Content Composer

Studio

Foundation 23.1

Written by: Documentation Team, R&D
Date: Monday, February 13, 2023

Documentation Notice

Information in this document is subject to change without notice. The software described in this document is furnished only under a separate license agreement and may only be used or copied according to the terms of such agreement. It is against the law to copy the software except as specifically allowed in the license agreement. This document or accompanying materials may contain certain information which is confidential information of Hyland Software, Inc. and its affiliates, and which may be subject to the confidentiality provisions agreed to by you.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright law, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Hyland Software, Inc. or one of its affiliates.

Hyland, HXP, OnBase, Alfresco, Nuxeo, and product names are registered and/or unregistered trademarks of Hyland Software, Inc. and its affiliates in the United States and other countries. All other trademarks, service marks, trade names and products of other companies are the property of their respective owners.

© 2023 Hyland Software, Inc. and its affiliates.

The information in this document may contain technology as defined by the Export Administration Regulations (EAR) and could be subject to the Export Control Laws of the U.S. Government including for the EAR and trade and economic sanctions maintained by the Office of Foreign Assets Control as well as the export controls laws of your entity’s local jurisdiction. Transfer of such technology by any means to a foreign person, whether in the United States or abroad, could require export licensing or other approval from the U.S. Government and the export authority of your entity’s jurisdiction. You are responsible for ensuring that you have any required approvals prior to export.

Table of Contents

Content Composer Studio **51**
Content Composer Studio 51
User Interface *51*
Overview 51
Editor Window 51
Main Menu 51
Quick Access Toolbar 52
Navigator Pane 53
About the Navigator Pane 53
Displaying the Navigator Pane 53
About the Folder View Tab 53
Filter 53
About the Filter Tab 53
Using the Filter 54
Search 54
About the Search Tab 54
Performing a Full-Text Search 54
Navigator Toolbar 55
About the Navigator Toolbar 55
Displaying Objects on Multiple Pages 55
Refresh 56
Show Filterlist 56
Last View 56
Importing Objects 56
Displaying Locked Objects 57
Displaying Work Versions 57
Managing Scheduled Transfers 58
Displaying Transfer History 59

Revoking a Transfer 59
Displaying Deleted Objects 60
Restoring an Object 60
Deleting an Object Permanently 60
Context Menu of the Navigator Pane 61
Object Inspector Pane 62
About the Object Inspector Pane 62
Displaying the Object Inspector Pane 62
Context Menu of the Object Inspector Pane 63
Output Pane 63
About the Output Pane 63
Displaying the Output Pane 63
Context Menu of the Output Pane 63
Toolbox Pane 63
About the Toolbox Pane 63
Displaying the Toolbox Pane 63
System Font 64
Administration *64*
Odin Views 64
About Odin Views 64
Accessing Odin Views 64
Process View 64
About the Process View 64
Opening the Process View 64
Context Menu of the Process View 65
Context Menu of the Jobs of a Process 65
Stack View 66
About the Stack View 66
Opening the Stack View 66
Context Menu of the Stack View 66

Context Menu of the Jobs of Stack View 67
Context Menu of the Envelopes of Stack View 68
Context Menu of the Jobs of Envelopes View 69
Open Job View 69
About the Open Job View 69
Opening the Open Jobs View 69
Context Menu of the Open Jobs View 70
Open Envelopes View 70
About the Open Envelopes View 70
Opening the Open Envelopes View 70
Context Menu of the Open Envelopes View 71
XPS Viewer 71
Odin Views Tools 71
Overview of Odin Statuses 72
Query Assistant 74
About Query Assistant 74
Accessing the Query Assistant 74
Run a Predefined Query 74
DB Alias Administration 74
About DB Alias Administration 74
Accessing DB Alias Administration 74
Creating a Database Alias 75
Modifying the Connection Settings 75
Connection Settings for MS SQL Server 75
Deleting a Database Alias 76
Exporting a Database Alias 76
Repository Import 77
About Repository Import 77
Accessing Repository Import 77

Importing Objects 77
System Synchronization 77
About System Synchronization 77
Accessing System Synchronization 77
Synchronize Two Systems 78
System Administration 78
About System Administration 78
Accessing System Administration 78
Creating a Database Connection 78
Creating a New System 80
Assigning a Word Template 80
Modifying a Database Connection 81
Defining a Transfer System 81
Deleting a System 82
Restoring a System 82
System Properties 83
Remote Control 83
About Remote Control 83
Accessing Remote Control 84
Control a Hosted Process 84
RuntimeRemoteControl Tools 84
Odin Import Procedure Manager 85
About Odin Import Procedure Manager 85
Accessing Odin Import Procedure Manager 85
Activating Odin Import Procedure Manager 85
Extending the Stored Procedure INSERT_JOB 86
Transfer data from a bundle to the ODIN_JOB table 87
Transfer Data from a Document to the ODIN_JOB Table 87
User Rights *87*
About User Rights 87

Profiles 88
About Profiles 88
Role profiles 88
Creating a Profile 88
Adding an Attribute to a Profile 88
Roles 89
About Roles 89
Creating a Role 89
Role Mappings 89
About Role Mappings 89
Creating a Role Mapping 89
Text Blocks *90*
About Text Blocks 90
Creating a Text Block 90
Text Block Properties 91
Text Block Editor Tools 96
Text Block Variables 97
About Text Block Variables 97
Inserting a Text Block Variable 97
Formatting a Text Block Variable 99
Special Variable Contents 100
Applying a Hyperlink as Content of a Variable 100
Applying a Line Feed, Tab, Non-Breaking Hyphen or Non-Breaking Space 100
Defining a Special Variable Content in the Property Formatting Mode 100
Defining an Input Mask 101
Defining Variable Rules 103
Manual Variables 104
Variable Properties 105
Image Variables 118
About Image Variables 118

Inserting an Image Variable 118
Hyperlink in an image variable 119
Applying a URL as Source of an Image Variable 120
Base64 coded images 120
Dynamic Charts 123
About Dynamic Charts 123
Predefine a Dynamic Chart 123
Chart properties 124
Assigning a Data Model 129
Inserting a Child Text Block 130
Inserting a Picture 130
Applying Text Block and Variable Properties 130
Multiple Text Blocks 132
About multiple text blocks 132
Defining a Multiple Text Block 132
Multiple insertions of a text block using a script 132
The Content Control EmptyLine 133
About the Content Control EmptyLine 133
Activating the Developer Tools in Microsoft Word 133
Inserting an EmptyLine Content Control 134
Variable Pools 134
Variable Text Blocks 135
About Variable Text Blocks 135
Defining a Variable Text Block 135
Attribute Inheritance 136
About Attribute Inheritance 136
Activating Inheritance 136
Property Transfer 137
About Property Transfer 137
Transfer Properties 137

Reports 137
Generating the "Structure of a text block" Report 137
Generating the "Used data connections" Report 138
Documents *138*
About Documents 138
Dynamic Documents 138
Creating a Dynamic Document 138
Dynamic Document Editor Tools 139
Inserting a Text Block 140
Inserting a Picture 140
Assigning a Data Model 141
Property Transfer 141
About Property Transfer 141
Transfer Properties 141
Testing a Document 142
Assigning an External Document 142
Using the Document Preview 142
Variable Documents 143
About Variable Documents 143
Defining a Variable Document 144
The Content Control CopyLabel 145
About the Content Control CopyLabel 145
Activating the Developer Tools in Microsoft Word 145
Creating the Content Control CopyLabel 145
Groups and Actions 145
About Groups and Actions 145
Creating a Group 146
Creating an Action 146
Reports 147

Generating the "Structure of a document" Report 147
Generating the "Used data connections" Report 147
Static Documents 148
Creating a Static Document 148
Static Word Documents 148
Static TIFF Documents 149
Static Document Editor Tools 150
Global Variables 150
About Global Variables 150
Creating a Global Variable 150
Properties of Global Variables 150
Document Properties 151
Bundles *158*
About Bundles 158
Creating a Bundle 158
Bundle Properties 158
Bundle Editor Tools 161
Creating a Document Copy 161
Assigning a Data Model 162
Prevent a User from Printing a Bundle 162
Groups and Actions 163
About Groups and Actions 163
Creating a Group 163
Creating an Action 164
Dynamic Bundles 164
About Dynamic Bundles 164
Creating a Dynamic Bundle 164
Testing a Bundle 165
Testing a Bundle 165
Assigning an External Document 166
Identical Archive Attributes for all Documents in a Bundle 166

Transfer Properties ... 167
Reports ... *167*
About Reports ... 167
Generating the Text Blocks Report ... 168
Generating the Unused Objects Report ... 168
Generating the Used Data Connections Report ... 170
Scripts and Conditions ... *171*
About Scripts and Conditions ... 171
Creating Scripts and Conditions ... 171
Creating a Script ... 171
Creating a Condition ... 172
Script and Condition Editor Tools ... 173
Code Completion ... 174
Setting a Bookmark ... 175
Displaying Context Information ... 175
Creating a Script Parameter ... 176
Adding a comment ... 176
Assigning a Script to an Object ... 176
Assigning a Condition to an Object ... 177
Debugging ... 177
Debugging a Script or a Condition ... 177
Debugger Tools ... 178
SOAP Web Service ... 179
Inspecting a SOAP Web Service ... 179
SOAP Web Service Pane ... 179
MonaLisa Functions ... 180
Data Types ... 180
Operators ... 182
Control Structures ... 183
About Control Structures ... 183

CheckEvery 183
CheckFirst 184
If-Condition 185
IfElse-Condition 185
While-Loops 186
Region 186
Exception Handling 186
About Exception Handling 186
TryFinally 186
TryOnError 187
TryOnErrorFinally 187
Standard Functions 189
Append 189
Beep 189
BigInteger 189
Bof 190
CallSystem 190
Close 192
CloseDatabase 192
CloseSQL 192
Commit 193
Contains 193
Contains_Exactly 194
Count 194
CreateGuid 195
CreatePool 196
Decimal 196
Delete 197
DeletePool 197

EncryptString 198
EndsWith 198
Eof 198
Execute 199
Find 200
FindValue 200
First 202
Float 202
Format 203
FormatStr 205
FreeObject 205
GetBlobField 206
GetConnection 206
GetField 207
GetLookupValue 208
GetObject 208
GetVariable 209
GetSystemOId 209
Index 210
InitDatabase 210
InitDataset 210
Insert 211
Integer 211
Last 212
Max 212
Min 213
Move 213
Next 214
OpenSQL 214

Parse 215
PrepareSQL 216
Prev 218
Protocol 219
PutField 219
Raise 220
Read 220
RegisterLib 223
ReRaise 223
Rollback 223
Round 224
Search 224
SelFirstRecord 225
SelNextRecord 225
SelRecordCount 226
SelRecordNumber 226
SelVariable 227
SelVariableEx 227
SelVariableXPath 228
SelVariableXsDateTime 229
SelVariableXsFloat 230
SetExecSQL 230
SetPoolVariable 237
SetSQL 237
Sleep 238
SplitStr 238
StartTransaction 239
StartsWith 239
StoredProcedure 240

StrCompare 240
StrDelChar 241
StrIndexOf 242
StrLastIndexOf 243
String 243
StringListAdd 244
StringListClear 244
StringListCount 245
StringListGetCommaText 245
StringListGetText 245
StringListGetValue 246
StringListIndexOf 246
StringListItemValues 247
StringListLine 248
StringListLoad 248
StringListParams 249
StringListSave 249
StringListSetCommaText 255
StringListSetValue 255
StringListSort 256
StrLeft 256
StrLen 256
StrLower 257
StrPos 257
StrReplace 258
StrReverse 259
StrRight 259
StrTok 260
StrToLen 260

StrUpper 261
StrWord 262
StrWordCount 262
SubString 263
Trim 263
TrimEnd 264
TrimStart 264
Trunc 265
UnRegisterLib 265
Update 266
VerifyPool 266
Version 266
Sample Script: Database Access 267
Xdata Functions 268
About Xdata Functions 268
Transaction Control 268
About Transaction Control 268
StartTransaction 269
Commit 269
Rollback 269
AddDataRecord 269
ApplyDataRecord 270
ApplyDataSet 270
DefineContainer 270
Eof 271
ExecuteQuery 271
First 272
GetField 272
Last 272

Message ... 273
Next ... 273
PrepareParameter ... 273
Prev ... 274
RowCount ... 274
StartProcedure ... 275
WorkWithQuery ... 275
Date Functions ... 275
About MLDate ... 275
Format Specifiers ... 276
Functions of the Class MLDate ... 276
CurrentDate ... **276**
CurrentDateToString ... **277**
DateStringToInt64 ... **277**
DayOfWeek ... **278**
DaysInMonth ... **278**
FormatDateTimeString ... **278**
IncDate ... **279**
Int64ToDateString ... **280**
IsLeapYear ... **280**
WeekOfYear ... **280**
File Processing ... 281
About MLDos ... 281
Functions of the Class MLDos ... 281
ChangeExtension ... **281**
Combine ... **281**
CopyFile ... **282**
CreateTextFile ... **282**
CreateDirectory ... **283**
DeleteDirectory ... **283**
DeleteFile ... **284**

DeleteFiles **284**
DirectoryExists **284**
ExecuteSynchron **285**
FileAppendText **285**
FileExists **291**
FileReadAllBytes **291**
FileReadAllText **292**
FileSize **297**
GetCommandLine **297**
GetCommandLineArgByName **298**
GetCommandLineArgs **298**
GetCurrentDirectory **298**
GetDirectories **298**
GetDirectoryName **299**
GetDirectoryRoot **299**
GetEnvironmentVariable **300**
GetExtension **300**
GetFileName **300**
GetFileNameWithoutExtension **301**
GetFiles **301**
GetFileSystemEntries **302**
GetFullPath **302**
GetLogicalDrives **302**
GetParentDirectory **303**
GetPathRoot **303**
GetRandomFileName **303**
GetTempFileName **304**
GetTempPath **304**
HasExtension **304**
IsPathRooted **305**
MoveDirectory **305**
MoveFile **305**
ReplaceFile **306**

SetCurrentDirectory **306**
Properties of the Class MLDos 307
Functions of the Class MonaLisaTextFile 307
Append **307**
Close **312**
Create **313**
Eof **318**
Flush **318**
Open **318**
ReadLine **324**
ReadToEnd **324**
Write **324**
WriteFormat **325**
WriteLine **325**
WriteLineFormat **325**
Properties of the Class MonaLisaTextFile 326
Examples 326
XML Processing 329
About MLXmlDocument 329
Additional Functions of the Class MLXmlDocument 329
GetAttribute **329**
GetAttributeOfNode **330**
GetNamespaceManager **330**
GetXpathValue **330**
Load **331**
LoadXml **331**
SetAttribute **332**
SetAttributeOfNode **332**
Sample Scripts 333
Sample Script: Reading and Modifying Attributes **333**
Sample Script: Create an XML File **336**
Ini-File Processing 337

About MLIniFile 337
Functions of the Class MLInifile 337
Close **337**
CopySection **338**
DeleteKey **338**
EraseSection **339**
GetSectionNames **339**
Open **340**
ReadBool **340**
ReadInteger **341**
ReadSectionToCommatext **341**
ReadString **342**
WriteBool **342**
WriteInteger **343**
WriteSectionFromCommatext **343**
WriteString **344**
MWS Functions 345
About MLMwsClient 345
Functions of the Class MLMwsClient 345
General Information **345**
ClearCache **345**
Doc_GetFile_Mime **345**
Doc_SetFile_Mime **346**
GetList **346**
Obj_GetStructure **347**
Obj_SetStructure **348**
Obj_Toggle **348**
Ping **349**
Process_Close **349**
Process_Create **350**
Process_Delete **351**
Process_Forward **351**

Process_GetInfo **351**
Process_GetLastError **352**
Process_Open **352**
Process_SetData **353**
Process_Start **353**
Srv_GetInfo **354**
Class Helper Functions 355
LoadBase64StringFromBinFile **355**
SaveBase64StringToBinFile **355**
Properties of the Class MLMwsClient 356
Sample Script MLMwsClient 357
Sample Script: Create a Process 364
SOAP Webservices 365
About SOAP Web Services 365
Requirements 365
Properties of the Class MLWsClient 366
Functions of the Class MLWsClient 367
CallMethod **367**
DefineWebProxy **367**
GetArrayLength **368**
GetEnumValue **369**
GetObjectFromArray **369**
GetObjectStringFromArray **369**
GetPropertyNameByIndex **370**
GetPropertyValueByIndex **371**
GetPropertyValueByIndexAsObject **371**
GetPropertyValueByName **372**
GetPropertyValueByNameAsObject **373**
GetRefParamNameByIndex **373**
GetRefParamValueByIndex **374**
GetRefParamValueByIndexAsObject **375**

GetRefParamValueByName ... **375**
GetRefParamValueByNameAsObject ... **376**
GetWebserviceProxyTypeFromString ... **377**
IsArrayObject ... **377**
IsObjectNull ... **378**
SetParameter ... **378**
Example Login / Logout ... **379**
Example: Calling Web Services ... **383**
User Attributes ... 396
About MLUserRepository ... 396
Functions of the Class MLUserRepository ... 396
GetCurrentUser ... **396**
GetUser ... **397**
Functions of the MonaLisa User Object ... 397
GetAttribute ... **397**
HasPermission ... **398**
HasObjectPermission ... **404**
IsInRole ... **407**
Sample Script MLUserRepository ... 408
Configuration Files ... 409
Archive Functions ... 410
About MLPerceptiveArchive ... 410
Functions of the Class MLPerceptiveArchive ... 410
AddDocumentPage ... **410**
AddToWorkflow ... **410**
Connect ... **411**
CreateDocument ... **411**
CreateDocumentWithFields ... **412**
Disconnect ... **413**
GetDocumentInformation ... **413**
GetDocumentViewResult ... **413**

GetDocumentViews **414**
GetFolderContentViewResult **414**
GetFolderViewResult **414**
GetFolderViews **415**
GetPageInformation **415**
GetPageFile **416**
GetWorkflowQueues **416**
MoveDocumentToFolder **418**
SearchForDocument **418**
UpdateDocumentDateAttribute **418**
UpdateDocumentFlagAttribute **419**
UpdateDocumentKey **420**
UpdateDocumentNumberAttribute **420**
UpdateDocumentStringAttribute **420**
Properties of the Class MLPerceptiveArchive 421
Sample Script MLPerceptiveArchive 421
Sample Script - Retrieve Files from Archive 426
Array Functions 428
About Array Functions 428
Properties of the Class MLArrays 428
Functions of the Class MLArrays 428
AddToArray **428**
Clear **429**
GetArray **429**
GetAtIndex **429**
InitFromArrayObject **429**
InsertIntoArray **430**
Object Functions 430
About Object Functions 430
Functions of the Class MLObjects 430
CreateNewInstanceOfType **430**

GetEnumValue **431**
GetModule **431**
GetTypeInModule **432**
GetTypeOfObject **432**
GetTypeOfObjectAsString **432**
MLIdAutomation 433
About MLIdAutomation 433
Functions of the Class MLIdAutomation 433
Code128 ..... **433**
Code128a **434**
Code128b **434**
Code128c **434**
Interleaved2of5 ..... **435**
Interleaved2of5Mod10 ..... **435**
Sample Script MLIdAutomation **436**
OnBase Document API 436
About MLOnBaseDocumentApi 436
Functions of the Class MLOnBaseDocumentApi 436
AddFile **436**
ArchiveFile **437**
ArchivePdfFile **437**
ClearKeywords **437**
FinishArchivePdfFile **438**
LoadKeywordsFromFile **438**
SetDocumentComment **438**
SetDocumentDate **438**
SetKeyword **439**
SetKeywordDate **439**
SetKeywordDateTime **440**
SetKeywordNumber **440**
SetLicenseFilePath **440**
SetOnBaseDocumentApiUrl **441**

SetOnBaseDocumentApiUrlFromConfiguration **441**
SetProfilePath **441**
SetRequestTimeout **442**
StartArchivePdfFile **442**
Sample Script: Convert one Input File into one PDF File **442**
Sample Script: Combine Multiple Input Files into one PDF File **443**
Sample Script: Send one File to the OnBase Archive **444**
HTTP API 445
About HTTP API 445
MLHttpClient 445
About MLHttpClient **445**
AddDefaultHeader **446**
ConvertFromBase64 ..... **446**
ConvertToBase64 ..... **446**
CreateRequest **447**
Delete **447**
Get **447**
GetCookie **448**
GetDefaultHeader **448**
GetDefaultHeaderCount **448**
Post **449**
Put **449**
RemoveDefaultHeader **449**
SetBaseAddress **450**
SetCookie **450**
SetIdpTokenMinValidPeriodMinutes **450**
SetRequestTimeout **451**
UseOAuth2ClientCredentials **451**
UseOAuth2PasswordGrant **452**
MLHttpRequest 452
About MLHttpRequest **452**
AddHeader **453**

SetByteContent **453**
SetByteContentFromFile **454**
SetStringContent **454**
SetUri **455**
MLHttpResponse 455
About MLHttpResponse **455**
GetByteContent **455**
GetHeader **456**
GetHeaderCount **456**
GetReasonPhrase **456**
GetStatusCode **457**
GetStringContent **457**
IsSuccessStatusCode **457**
SaveByteContentToFile **457**
Sample Script HTTP API 458
JSON API 459
About JSON API 459
MLJson 459
About MLJson **459**
AddArray **459**
AddObject **460**
AddPropertyValue **460**
Clear **461**
Exists **461**
GetArray **461**
GetObject **461**
GetStringValue **462**
LoadArrayFromFile **462**
LoadArrayFromString **462**
LoadFile **463**
LoadString **463**
Remove **463**

SaveToFile **464**
SetPropertyValue **464**
ToString **464**
MLJsonArray 465
About MLJsonArray **465**
AddArray **465**
AddElementValue **465**
AddObject **466**
Clear **466**
GetArray **466**
GetCount **467**
GetObject **467**
GetStringValue **467**
Remove **468**
SaveToFile **468**
SetElementValue **468**
ToString **469**
MLJsonObject 469
About MLJsonObject **469**
AddArray **469**
AddObject **469**
AddPropertyValue **470**
Clear **470**
Exists **470**
GetArray **471**
GetObject **471**
GetStringValue **471**
Remove **472**
SetPropertyValue **472**
ToString **472**
Sample Script JSON API: Create Array 473
Sample Script JSON API: Create Object 474

Sample Script JSON API: Read Data ... 475
Properties and Functions of Contexts ... 477
Context FlatFileConverter ... 477
Properties of the Context FlatFileConverter ... 477
Sample Script for context FlatFileConverter ... 477
Context ManualVariable ... 479
About the Context ManualVariable ... 479
Properties of the Context ManualVariable ... 479
Functions of the Context ManualVariable ... 483
AddItem ... **483**
ClearItems ... **484**
ItemExists ... **484**
Context ModusBatch ... 484
About the Context ModusBatch ... 484
Properties of the Context ModusBatch ... 484
Functions of the Context ModusBatch ... 485
AssignData ... **485**
DeleteDirectory ... **485**
GetData ... **486**
SetDataContext ... **486**
Context MwsProcess ... 486
About the Context MwsProcess ... 486
Properties of the Context MwsProcess ... 487
Functions of the Context MwsProcess ... 489
GetActivityStatus ... **489**
GetForwardInformation ... **491**
GetObjectSelectionParameters ... **491**
GetOptions ... **491**
IsInformationRequested ... **492**
SetObjectSelectionParameters ... **492**
Context OdinArchive ... 492

About the Context OdinArchive ... 492
Properties of the Context OdinArchive ... 492
Functions of the Context OdinArchive ... 496
HasData ... 496
Context OdinConvert ... 497
About the Context OdinConvert ... 497
Properties of the Context OdinConvert ... 497
Functions of the Context OdinConvert ... 505
HasData ... 505
Context OdinCustomProcess ... 505
About the Context OdinCustomProcess ... 505
Properties of the Context OdinCustomProcess ... 506
Functions of the Context OdinCustomProcess ... 510
HasData ... 510
Context OdinDbTask ... 510
About the Context OdinDbTask ... 510
Properties of the Context OdinDbTask ... 510
Functions of the Context OdinDbTask ... 514
HasData ... 514
Context OdinEmail ... 515
About the Context OdinEmail ... 515
Properties of the Context OdinEmail ... 515
Functions of the Context OdinEmail ... 523
AddAttachment ... 523
AddBodyHtmlImage ... 523
AddToHeader ... 524
ClearAttachments ... 524
ClearBodyHtmlImages ... 525
ClearCertificates ... 525
GetAttachments ... 525

LoadBodyFromFile ... **525**
LoadCertificateFromCertFile ... **526**
LoadCertificateFromSignedFile ... **526**
RemoveAttachment ... **526**
RemoveFromHeader ... **526**
Send ... **527**
SetCredentials ... **527**
SetSmtpServer ... **527**
Sample Script OdinEmail ... **528**
Context OdinEmailConvert ... 529
Properties of the Context OdinEmailConvert ... 529
Functions of the Context OdinEmailConvert ... 534
SetPdfSigner ... **534**
Functions of IOdinStreamDocument Interface ... 535
AddBookmark ... **535**
EditPage ... **535**
GetInputBin ... **536**
Image ... **536**
IsDuplex ... **537**
IsPortrait ... **537**
Line ... **537**
Pagecount ... **545**
Post ... **546**
Rectangle ... **546**
SetFont ... **562**
SetInputBin ... **562**
TextOut ... **562**
Context OdinFilter ... 570
About the Context OdinFilter ... 570
Functions of the FilterObject ... 571
About the Filter Object ... **571**
DefinePostageDefinitionByName ... **571**

DefinePostageDefinitionByOId **571**
DefinePrinterByName **572**
DefinePrinterByOId **572**
EnumEnvelopes **572**
Gen_Stack **573**
GetParam **574**
RunFilter **574**
SetParam **575**
Overview of Possible Parameters for the Functions GetParam and SetParam 575
Sample Script: Simple Filter Including Stack Generation 577
Functions of the Envelope Object 579
About the Envelope Object **579**
AppendVJob **579**
CalcEnvelopePages **579**
CalcEnvelopePapers **580**
CalcEnvelopePostage **580**
CalcEnvelopeWeight **580**
Commit **581**
Count **581**
CreateEnv **581**
EndTimer **581**
Eof **582**
First **582**
GetField **582**
GotoJobTillWeight **583**
GotoRecNum **583**
Init **583**
InsertVJob **584**
Last **584**
Next **585**
Prev **585**
ReadNextEnvelope **585**

Recnum **585**
Rollback **586**
SetField **586**
Split **586**
StartTimer **586**
StartTransaction **587**
Sample Script: Simple Enveloping Filter with Stack Generation 587
Context OdinPrint 592
Properties of the Context OdinPrint 592
Functions of the Context OdinPrint 596
HasData **596**
Context OdinRendition 596
About the Context OdinRendition 596
Properties of the Context OdinRendition 597
Functions of the Context OdinRendition 597
GetFieldNameEnumerator **597**
GetJobRow **597**
GetValue **598**
HasData **598**
SetValue **599**
Context OdinStream 600
About the Context OdinStream 600
Properties of the Context OdinStream 600
Functions of the IOdinStream-Interface 600
AppendDocument **600**
GetProperty **601**
LoadDocument **605**
LoadEnvelopes **605**
LoadJobs **606**
SetProperty **606**
Sample Script LoadEnvelopes / LoadJobs **606**

Functions of the Context OdinStream ... 608
HasData ... 608
Context OdinStreamDocument ... 608
About the Context OdinStreamDocument ... 608
Properties of the IOdinStreamDocument-Interface ... 608
Functions of IOdinStreamDocument-Interface ... 610
AddBookmark ... 610
AddComment ... 610
AddPJLCommand ... 611
EditPage ... 611
GetInputBin ... 612
Image ... 612
IsDuplex ... 613
IsPortrait ... 613
Line ... 613
Pagecount ... 621
Post ... 622
Rectangle ... 622
SetAfpCopyGroup ... 637
SetFont ... 638
SetInputBin ... 638
TextOut ... 639
Context OdinStreamPrint ... 647
Properties of the Context OdinStreamPrint ... 647
Functions of the Context OdinStreamPrint ... 654
HasData ... 654
Context OnError ... 655
Properties of the Context OnError ... 655
Context OnNavigateObject ... 655
Properties of the Context OnNavigateObject ... 655
Sample Script OnNavigateObject ... 656

Context OnProcessError ... 657
Properties of the Context OnProcessError ... 657
Context Process ... 659
About the Context Process ... 659
Properties of the Context Process ... 659
Context ProcessTimer ... 661
About the Context ProcessTimer ... 661
Properties of the Context ProcessTimer ... 661
Context SelectionQueryScript ... 662
Properties of the Context SelectionQueryScript ... 662
Context TextComponent ... 663
About the Context TextComponent ... 663
Properties of the Context TextComponent ... 663
Functions of the Context TextComponent ... 665
LoadExternalDocument ... 665
LoadExternalDocumentFromBytes ... 665
Navigate ... 666
Context TextDocument ... 666
About the Context TextDocument ... 666
Properties of the Context TextDocument ... 667
Functions of the Context TextDocument ... 667
LoadExternalDocument ... 667
LoadExternalDocumentFromBytes ... 668
Navigate ... 668
Context TextDocumentCollection ... 669
About the Context TextDocumentCollection ... 669
Properties of the Context TextDocumentCollection ... 669
Functions of the Context TextDocumentCollection ... 671
AddDocumentByName ... 671
AddDocumentByObjectId ... 675

AddDocumentCopyByName **678**
AddDocumentCopyByObjectID **681**
Navigate **683**
SetArchiveAttribute **684**
SetDataContext **684**
Context TextDocumentReference 684
About the Context TextDocumentReference 684
Properties of the Context TextDocumentReference 684
Functions of the Context TextDocumentReference 688
DefinePrinterByName **688**
DefinePrinterByOId **689**
SetArchiveAttribute **690**
Context TextVariable 690
About the Context TextVariable 690
Properties of the Context TextVariable 691
Functions of the Context TextVariable 702
Navigate **702**
Sample Script: Generate a Chart 703
Context XsdSetXmlData 704
About the Context XsdSetXmlData 704
Properties of the Context XsdSetXmlData 705
Sample Script XsdSetXmlData 705
Script Order 705
Alias Tables *708*
About Alias Tables 708
Creating an Alias Table 708
Lookup Tables *708*
About Lookup Tables 708
Creating a Lookup Table 708
Transfer Jobs *709*
About Transfer Jobs 709

Creating a Transfer Job 710
Transfer Status 711
Creating a Scheduled Transfer Job 711
Exporting Objects 712
Transfer Job Editor Tools 712
Value Helps *713*
About Value Helps 713
Static Value Helps 713
About Static Value Helps 713
Creating a Text Box with Input Mask Value Help 713
Creating a Drop-Down Value Help 715
Creating a Popup Calendar Value Help 716
Creating a Numeric Input Field Value Help 717
Creating a Multiline Text Box Value Help 717
Dynamic Value Helps 718
About Dynamic Value Helps 718
Word User Interfaces *718*
About Word User Interfaces 718
Creating a Word User Interface 718
Assigning a Word User Interface to a Document 719
Word Templates *719*
About Word Templates 719
Creating a Word Template 720
Modifying a Word Template 720
Exporting a Word Template 721
Processes *721*
About Processes 721
Creating a Process 721
Properties of a Standard Process 722
Properties of an MWS Process 722
Process Designer Tools 723
Process Designer Modules 724

Areas of a Process Module ... 724
Check ... 726
Condition ... 726
IF-Check ... 727
IF-ELSE-Check ... 727
Comment ... 727
Loop ... 727
Script Execution ... 727
Universal Timer ... 727
Position and Connect Process Designer Modules ... 728
About Positioning and Connecting Process Modules ... 728
Examples: Add Modules to a Process ... 728
Adding a Loop to a Process ... 728
Inserting a Script Execution in the Loop ... 729
Inserting Another Script Execution in the Loop ... 729
Inserting a Script Execution as Loop Successor ... 729
Moving a Script Between two Already Linked Scripts ... 731
Moving a Script to a Position Outside the Loop ... 731
Inserting a Script Between a Script and a Loop ... 731
Inserting a Script Back in a Loop ... 732
Arranging Conditions in Succession ... 732
Nest Conditions ... 733
Process Designer Components ... 734
About Process Designer Components ... 734
CsvConverter ... 734
About CsvConverter ... 734
Properties of CsvConverter ... 734
CsvConverter configuration ... 735
Simple Example ... 738
Advanced Example ... 739

Known Limitations ... 741
CsvConverter Functions ... 741
Init ... **741**
IsInitialized ... **741**
Run ... **742**
Sample Script CsvConverter ... **742**
DataSelectionManager ... 743
About DataSelectionManager ... 743
Properties of the DataSelectionManager ... 743
DataSelectionManager Functions ... 744
GetData ... **744**
GetDataExtended ... **744**
LoadSelection ... **745**
LoadSelectionByName ... **745**
LoadSelectionByOId ... **746**
WriteXml ... **746**
Sample Script DataSelectionManager ... **747**
DocumentCollectionManager ... 747
About DocumentCollectionManager ... 747
Properties of the DocumentCollectionManager ... 747
DocumentCollectionManager Functions ... 748
AssignData ... **748**
CreateDocuments ... **749**
DefinePrinterForBundleByName ... **749**
GetDataReferenceEnumerator ... **750**
GetManualVariables ... **751**
GetSelectionParameters ... **751**
LoadByAlias ... **751**
LoadById ... **751**
LoadByName ... **752**
PrintAndArchive ... **752**

ProcessData **753**
ResetDocuments **753**
SetManualVariables **753**
SetSelectionParameters **753**
Sample Script DocumentCollectionManager **754**
FlatFileConverter 754
About FlatFileConverter 754
Properties of FlatFileConverter 754
FlatFileConverter Events 755
FlatFileConverter Configuration 755
Defining Data Conversion 764
Process at runtime 766
Example 768
FlatFileConverter Functions 770
Init **770**
IsInitialized **771**
Run **771**
SampleScriptFlatFileConverter **771**
ModusBatchManager 772
About ModusBatchManager 772
Properties of the ModusBatchManager 772
ModusBatchManager Functions 772
DoWork **772**
FinishWork **772**
MwsActivityStatus 773
MwsAssistantManager 773
About MwsAssistantManager 773
Properties of the MwsAssistantManager 773
MwsAssistantManager Functions 773
DeleteExpiredSessions **773**

DeleteFinishedProcesses **774**
Sample Script MWSAssistantManager **774**
OdinArchive 775
About OdinArchive 775
Properties of OdinArchive 775
OdinArchive Events 776
OdinArchive Functions 777
ArchiveJob **777**
End **777**
GetCurrentStackRow **777**
GetEnvelopeEnumerator **779**
GetJobEnumerator **780**
GetNextStack **781**
HasEnvelopes **781**
Start **782**
Sample Script OdinArchive **782**
OdinConvert 784
About OdinConvert 784
Modifying the Output Tray List in a Compart Profile 784
Properties of OdinConvert 785
OdinConvert Events 791
OdinConvert Functions 791
ConvertDocument **791**
End **791**
GetCurrentStackRow **792**
GetDocumentEnumerator **793**
GetNextStack **794**
SetStackError **794**
Start **795**
Sample Script OdinConvert **795**
OdinCustomProcess 797

About OdinCustomProcess 797
Defining an Additional Stack Type 797
Properties of OdinCustomProcess 797
OdinCustomProcess Events 798
OdinCustomProcess Functions 799
End 799
GetCurrentStackRow 799
GetEnvelopeEnumerator 801
GetJobEnumerator 802
GetNextStack 803
HasEnvelopes 803
Process 804
ProcessJob 804
ProcessJobs 804
Start 805
Sample Script OdinCustomProcess 805
OdinDbTasks 808
About OdinDbTasks 808
Properties of OdinDbTasks 808
OdinDbTasks Functions 808
DeleteStack 808
DeleteProcessRecords 809
SetBackStackState 810
Sample Script OdinDbTasks 811
OdinEmail 811
About OdinEmail 811
Properties of OdinEmail 811
OdinEmail Functions 813
AddAttachment 813
AddBodyHtmlImage 813
AddToHeader 814

ClearAttachments **814**
ClearBodyHtmlImages **814**
ClearCertificates **815**
GetAttachments **815**
GetDeliveryNotificationOptions **815**
GetMailPriority **816**
GetSmtpDeliveryMethod **816**
LoadBodyFromFile **817**
LoadCertificateFromCertFile **817**
LoadCertificateFromSignedFile **817**
RemoveAttachment **818**
RemoveFromHeader **818**
Reset **818**
Send **819**
SetCredentials **819**
SetDeliveryNotificationOptions **819**
SetMailPriority **820**
SetSmtpDeliveryMethod **820**
SetSmtpServer **821**
Sample Script OdinEmail **821**
OdinFileConvert 824
About OdinFileConvert 824
Properties of OdinFileConvert 824
OdinFileConvert Functions 825
AddDocument **825**
AddFile **825**
CreatePdfSigner **825**
ConvertSingleFile **842**
ConvertXDocToHtml **849**
ConvertXDocToString **849**
ConvertXDocToText **850**
ConvertXDocToXps **855**

End ... **856**
LoadDocument ... **856**
NumberOfFilesConverted ... **857**
Start ... **857**
Sample Script OdinFileConvert: 1 file ... **863**
Sample Script OdinFileConvert: 1-n Files ... **865**
Sample Script OdinFileConvert: xDoc to XPS ... **866**
Convert Doc to Docx ... 867
ConvertDocToDocX ... **867**
EndDocConvert ... **867**
StartDocConvert ... **867**
OdinFilterManager ... 868
About OdinFilterManager ... 868
Properties of OdinFilterManager ... 868
OdinFilterManager Functions ... 868
Run ... **868**
Sample Script OdinFilterManager ... **868**
OdinOnBaseArchive ... 868
About OdinOnBaseArchive ... 868
Properties of OdinOnBaseArchive ... 869
OdinOnBaseArchive Functions ... 870
ArchiveStack ... **870**
OdinPrint ... 871
About OdinPrint ... 871
Properties of OdinPrint ... 871
OdinPrint Events ... 872
OdinPrint Functions ... 873
AddBatchPrintJob ... **873**
BatchJobCount ... **873**
CurrentPrinter ... **874**
End ... **874**

ExecuteBatchPrint ... **875**
GetCurrentStackRow ... **876**
GetNextStack ... **878**
GetPrinterEnumerator ... **878**
GetPrintJobEnumerator ... **879**
GetMailEnumerator ... **880**
GetMailJobEnumerator ... **880**
InitializeBatchPrint ... **881**
IsEmailPrinter ... **882**
IsPrinterActive ... **882**
PrintJob ... **882**
Start ... **883**
Sample Script OdinPrint ... **883**
OdinProcessImport ... 888
About OdinProcessImport ... 888
Properties of OdinProcessImport ... 888
OdinProcessImport Functions ... 890
ImportProcessFile ... **890**
SearchAndImportProcessFile ... **890**
Sample Script OdinProcessImport ... **891**
Sample Script OdinProcessImport error handling ... **891**
OdinRendition ... 892
About OdinRendition ... 892
Properties of OdinRendition ... 893
OdinRendition Events ... 894
OdinRendition Functions ... 895
CreateRenditionForProcess ... **895**
CreateRenditionForSelectedProcess ... **896**
Sample Script OdinRendition ... **896**
OdinStream ... 897
About OdinStream ... 897

Accessing the Virtual Database Tables ... 897
Setting OMR Definitions ... 897
OdinStream Output Files ... 898
Properties of OdinStream ... 904
OdinStream Events ... 918
OdinStream Functions ... 924
AddBookmark ... **924**
End ... **924**
GetCurrentStackRow ... **925**
GetEnvelopeEnumerator ... **927**
GetJobEnumerator ... **928**
GetNextStack ... **929**
SetStackError ... **929**
Start ... **930**
StreamDocument ... **930**
Sample Script OdinStream ... **931**
OdinStreamPrint ... 933
About OdinStreamPrint ... 933
Properties of OdinStreamPrint ... 933
OdinStreamPrint Events ... 934
OdinStreamPrint Functions ... 935
DefinePrinterByName ... **935**
DefinePrinterByOld ... **935**
End ... **935**
GetCurrentStackRow ... **936**
GetDocumentEnumerator ... **938**
GetNextStack ... **939**
PrintDocument ... **939**
SetStackError ... **940**
Start ... **940**
Sample Script OdinStreamPrint ... **941**

Transfer ... 942
About Transfer ... 942
Properties of Transfer ... 942
Transfer Functions ... 942
ExecuteTransfer ... **942**
GetNextTransfer ... **943**
XMLProcessingManager ... 943
About XMLProcessingManager ... 943
Properties of the XmlProcessingManager ... 943
XMLProcessingManager Functions ... 945
Cancel ... **945**
FileExists ... **945**
FileExistsEx ... **945**
GetJobAttribute ... **946**
NextJob ... **946**
StartProcessing ... **947**
Terminate ... **947**
XwsDataSelectionManager ... 947
About XwsDataSelectionManager ... 947
Properties of the XwsDataSelectionManager ... 947
XwsDataSelectionManagers Functions ... 948
GetData ... **948**
GetDataExtended ... **949**
Ping ... **949**
Sample Script XwsDataSelectionManager ... **950**
Query Definitions ... *951*
About Query Definitions ... 951
Creating a Query Definition ... 951
Adding a condition ... 952
Adding a Variable to a Condition ... 952
Adding a column ... 952

Properties of a Query Definition 953
Properties of Result Columns 953
Properties of a Condition 954
Properties of a Variable 954
OnBase Interfaces *955*
OnBase DIP File Configurations 955
About OnBase DIP File Configurations 955
Creating an OnBase DIP File Configuration 955
OnBase Document API Configurations 956
About OnBase Document API Configurations 956
Creating an OnBase Document API Configuration 956
Configuring Archiving Settings for a Document Template 957
Configuring Archiving Settings for a Bundle 958
Odin Processing 958
Sample Script: Archive Documents in the Component Odin Rendition 958
Maintaining OnBase Object IDs 963
Xdata *963*
About Xdata 963
Limitations on Calling BAPI Functions 964
Data Objects 964
About Data Objects 964
Creating a Data Object 965
Modifying a Data Object 965
Reading Scripts 966
About reading scripts 966
Creating a Reading Script 966
Modifying the SQL Query 967
Inserting Child Data Objects 967
Creating a Summary Node 968
Xdata Editor Tools for Data Objects 969

Empty Data Objects ... 969
Creating an Empty Data Object ... 969
Example of Use ... 970
Modifying or Deleting Datasets with an Empty Data Object ... 970
Executing a Stored Procedure with an Empty Data Object ... 970
Updating Data Objects ... 971
Properties of a Data Object ... 972
Properties of a Data Object Variable ... 973
Selections ... 974
About Selections ... 974
Defining the Start Parameters ... 974
List of Formatting Characters ... 975
Saving a Selection as a Schema ... 977
Xdata Editor Tools for Selections ... 977
Active Selections ... 978
Creating an Active Selection ... 978
Activating a Data Object ... 978
Refresh Data Objects in a Selection ... 979
Properties of an Active Selection ... 979
Testing Selections ... 979
Executing a Selection ... 979
Debug a selection ... 980
Using Test Sets ... 980
Recording a Data Retrieval ... 980
Replay a data retrieval ... 981
Missing Data Objects or Attributes ... 981
XSD Selections ... 981
Importing an XSD Schema ... 981
Exporting an XSD Schema ... 982
Comments in XSD Selections ... 982

Properties of XSD Selections ... 983
Execute an XSD selection ... 983
Debug an XSD selection ... 983
Supported XSD Schema Constructs ... 984
Rules for Data Objects ... **984**
Rules for Attributes ... **987**
Odin ... *990*
About Odin ... 990
Envelopes ... 990
About Envelopes ... 990
Creating an Envelope ... 990
Enclosures ... 990
About Enclosures ... 990
Creating an Enclosure ... 991
Postage Definitions ... 991
About Postage Definitions ... 991
Creating a Postage Definition ... 991
Forms ... 992
About Forms ... 992
Creating a Form ... 992
Printers ... 992
Email Printers ... 992
About Email Printers ... 992
Creating an Email Printer ... 992
Online Printers ... 993
About Online Printers ... 993
Creating an Online Printer ... 993
Creating an Online Printer for Local Printing ... 993
Outsourcing Printers ... 994
About Outsourcing Printers ... 994

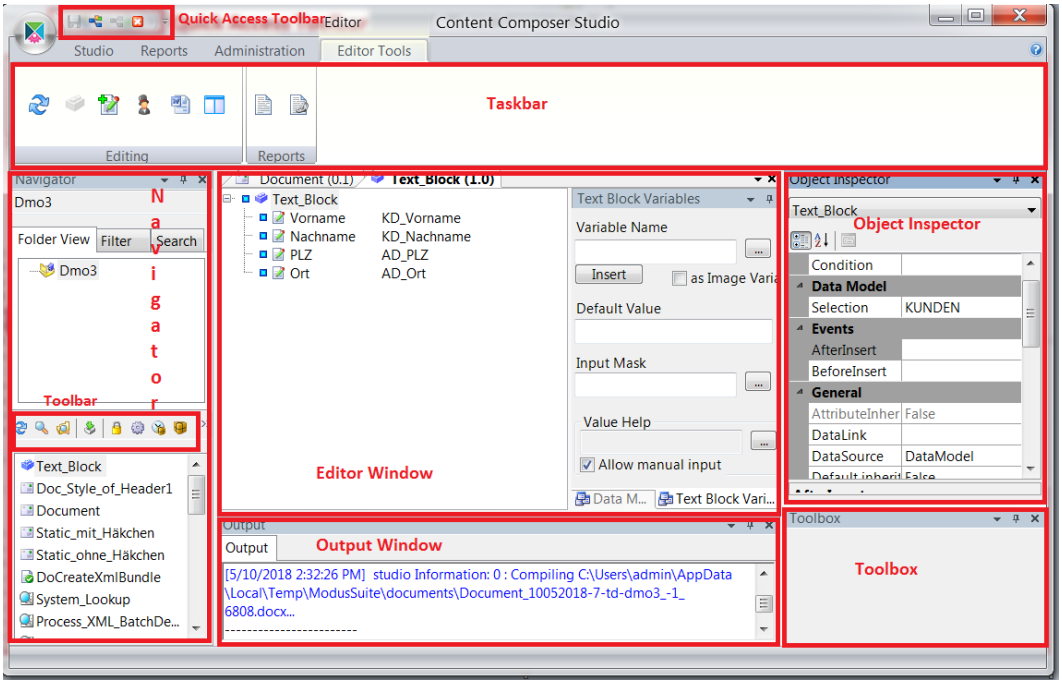
Creating an Outsourcing Printer 994
Printer Properties 995
Properties of an Email Printer 995
Properties of an Online Printer 997
Properties of an Outsourcing Printer 999
Tray Properties 1000
Properties of an Email Printer Tray 1000
Properties of an Online Printer Tray 1001
Properties of an Outsourcing Printer Tray 1002
Versioning *1003*
About Versioning 1003
Version Types 1004
Creating a New Work Version from a Release Version 1004
Creating a New Work Version from a Work Version 1005
Creating a New Release Version from a Work Version 1005
Modifying a Work Version Without Creating a New Version 1006
Displaying the Object History 1006
Reactivating an Older Version 1006
Transfer Versioned Objects to Other Systems 1007

Content Composer Studio

Content Composer Studio

User Interface

Overview



Editor Window

In the **Editor** window, you modify all objects in Content Composer Studio. It is possible to open numerous objects simultaneously. Each object opens in its own tab.

Main Menu

To access the main menu, click the four-colored icon in the top left-hand corner.

The main menu offers the following options.

Menu item	Description
New	Opens a submenu to select a new object to be created.
Create New Version	This menu item is only available when you have a versioning license installed. For more information, refer to Create a new version
Reactivate Version	This menu item is only available when you have a versioning license installed. For more information, refer to Show Object History
Save	Saves the current object.
Save All	Saves all open objects.
Close Current Object	Closes the current object.
Close All Objects	Closes all open objects.
Print	Enables you to print conditions, scripts and processes.
Print Preview	This menu item is available for conditions, scripts and processes and displays them in a print preview.
Clear Studio and Runtime Cache	If you modified an object, for example a document or a selection, and you test the modifications without saving the object first, a new compilation of the object will be created. If you close the object without saving, the new compilation remains in the cache. So, if the object is executed again, it is the cached compilation that executes. In this case, you should use the function **Clear Cache** to execute the object in its original state.
Debug Manager	The debug manager administrates scripts in the debug service. It allows you to remove them from the service.

Quick Access Toolbar

The quick access toolbar holds a number of buttons used to call frequently used functions.

Note You can use the menu to the right to hide or show these buttons.

Icon	Menu item	Description
	Save	Saves the current object.
	Create New Version	This menu item is only available when version control has been licensed accordingly. For more information, refer to Create a new version.
	Reactivate Version	This menu item is only available when version control has been licensed accordingly. For more information, refer to Reactivate an older version.
	Close Current Object	Closes the current object.

Navigator Pane

About the Navigator Pane

The **Navigator** pane enables you to administrate all Content Composer Studio objects.

The **Navigator** pane contains the following tabs.

- Folder View
- Filter
- Search

Displaying the Navigator Pane

If the **Navigator** pane is not visible, complete the following step.

- In the **Studio** tab, click the **Navigator** button.

About the Folder View Tab

The upper pane of the **Navigator** shows the folder structure. The lower pane shows all objects of the folder selected in the upper pane.

Filter

About the Filter Tab

In the **Filter** tab, you can locate objects with names containing a specified text pattern.

Using the Filter

To use the filter, complete the following steps.

1. In the **Navigator**, click the **Filter** tab.
2. In the **Filter** tab, select one of the following options.
    - **starts with** Object names with the text pattern**.**
    - **ends with** Object names end with the text pattern**.**
    - **contains**Object names contain the text pattern**.**
3. In the input field, type your text pattern.
   Note As you are typing, the filter result in the lower pane refreshes with each keystroke.

Search

About the Search Tab

The **Search** tab enables you to perform a full-text search to locate objects that contain a specified text string.

Depending on database workload, changes of text block content or document text may be found after a delay caused by the update of the full-text database index.

Full-Text Search requires a specific index in your database system (SQL Full Text or Oracle Text). For more information, refer to the *Content Composer Advanced Design and Setup Guide* and contact your database administrator.

The original release of the Oracle 10g database does not support full-text search in DOCX files. When using this database product, the full-text search in DOCX files related to text blocks and documents does not produce the expected results.

Performing a Full-Text Search

To perform a full-text search, complete the following steps.

1. In the **Navigator**, click the **Search** tab. The objects matching your search display in the lower pane.
2. In the **Search** tab, select the object types you want the full-text search to regard.
   Note Depending on the scanned object type, the search performs either fuzzy matching or exact matching.

Object type	Search technique
Text Block Content	Exact Matching

Document Text	Exact Matching
Object Description	Exact Matching
Version Identifier	Fuzzy Matching
Script	Fuzzy Matching
Condition	Fuzzy Matching

Example

If you choose "future" as your search string, the search only detects the occurrence of the **exact** string "future" in text blocks and documents.

In object descriptions, scripts, and conditions, the search also finds additional word forms like "futures", as the search bases on **fuzzy matching**.

3. In the **Search for** box, type your search string and click the **Search** button.
   Note The search does not support the use of wildcard characters.

Navigator Toolbar

About the Navigator Toolbar

The **Navigator** toolbar offers various functions.

Displaying Objects on Multiple Pages

By default, all objects in the **Navigator** display on one page.

To display objects on a number of pages, complete the following steps.

1. In the top-left corner, click the main menu icon and click the **Options** button in the bottom right.
2. In the **Options** dialog box, in the left pane, click **General Settings**.
3. In the right pane, select the **Activate paging for the Navigator**check box.
4. In the **Page Size** box, type the number of objects to display on one page and click **OK**.
5. To page through the objects, on the **Navigator** toolbar, click the **Previous Page** button or the **Next Page** button.

Refresh

To go back to the standard view, for example after executing the function **Is Used By** or after displaying locked objects, complete the following step.

- On the **Navigator Toolbar**, click the **Refresh** button.

Show Filterlist

To use the filter list function, complete the following steps.

1. On the **Navigator**toolbar, click the **Show Filterlist** button.
2. In the **Filter Settings** dialog box, select the object types you want to display in the **Navigator** pane and click **OK**.

Last View

If you have just used the function **Is Used By** and have returned to the standard view, you can use this function to display the result of **Is Used By** again.

To display the result of **Is Used By** again, complete the following step.

- On the **Navigator Toolbar**, click the **Last View** button.

Importing Objects

This function enables you to import existing DOCX files as documents or text blocks.

To import DOCX files, complete the following steps.

1. On the **Navigator**toolbar, click the **Import Objects** button.
2. In the **Import Objects** dialog box, complete of the following substeps.
    1. Select one of the following options as target object type.
        - Text Block
        - Document
    2. For documents as target object type, select one of the following document types.
        - Dynamic Word Document
        - Static Word Document
    3. Optional. If you use the function from a subfolder, to link the imported objects current folder, select the **Create links to the current folder** check box.
    4. Optional. In the **Version identifier** box, modify the default text.

5. Click the button to the right of the **Import Directory** box.
6. In the **Search folder** dialog box, select the folder from which you want to import the DOCX files and click **OK**.
7. Click **Start**.

Notes

- The objects created receive the name of the respective import file.
- If the name of the import file contains invalid characters for object names, for example spaces, these are automatically replaced by the underline character.
- When an object has been imported, the suffix of the corresponding DOCX file is renamed to **.docx_done**.
- If you cannot import an object, the name of the DOCX file remains unchanged.
- Before each object is imported, the system checks whether an object with that name already exists.
  If this is the case, a dialog box displays and you can decide if you want to overwrite the existing object.
  If version control is licensed, confirming the dialog box message results in a new work version of the object.
  If the new document type is not the same as that of the existing object, the object will not import.

Displaying Locked Objects

If Content Composer Studio terminated unexpectedly while objects were still being edited, these objects remain locked.

To unlock locked objects, complete the following steps.

1. To display a list of locked objects in the bottom pane of the **Navigator**, on the **Navigator**toolbar, click the **Locked Objects** button.
2. Select the objects you want to unlock.
3. In the bottom pane of the **Navigator**, right-click and click **Unlock**.

Displaying Work Versions

Show All

This menu item is only available when you have a versioning license installed.

To display a list of all the objects whose highest version is a work version, complete the following step.

- On the **Navigator**toolbar, click the **Work Versions** button.

The **Work versions** dialog box offers the following functions.

- Create release version

  To create a new release version from a work version, complete the following steps.

  1. In the **Work versions** dialog box, select the preferred objects.
  2. Click the **Create Release Version** button.
  3. Optional. In the **Version Label** dialog box, in the **Label** box, type a label for the release version.
  4. Click **OK** and click **Close**.

- Add to new transfer job

  To add the selected objects to a new transfer job, complete the following steps.

  1. In the **Work versions** dialog box, select the preferred objects.
  2. Click the **Add to new transfer job** button and click **Close**.
  3. Start the transfer as described in Create a transfer job.

- Create release version and add to new transfer job

  To create release versions and add the selected objects to a new transfer job, complete the following steps.

  1. In the **Work versions** dialog box, select the preferred objects.
  2. Click the **Create release version and add to new transfer job** button
  3. In the **Create Release Version** dialog box, click **Yes**.
  4. **Optional**. In the **Version Label** dialog box, in the label box, type a label for the release version.
  5. Click **OK** and click **Close**.
  6. Start the transfer as described in Create a transfer job.

Notes

- The number of the release version sets automatically.
- You cannot create a release version for locked objects.

Managing Scheduled Transfers

This menu item is only available when you have a versioning license installed and when you possess the **Administration(12)** right.

This function lists all scheduled transfers in which the current system is the target system.

To manage scheduled transfers, complete the following steps.

1. On the **Navigator**toolbar, click the **Scheduled Transfers** button.
2. In the **Scheduled Transfers** dialog box, in the upper pane, click a transfer.
   Note The objects of the selected transfer display in the lower pane.
3. Optional. To edit a scheduled transfer, complete the following substeps.
   1. Click the **Edit** button.
   2. In the **Edit Scheduled Transfer** dialog box, complete one or more of the following substeps.
      1. Optional. To modify the start date, in the **Date** box, type a date or click the down arrow to select a date from the calendar.
      2. Optional. To modify the start time, in the **Time** list, select a time.
         Note You can specify the date to the nearest hour.
      3. Optional. In the **Status** list, modify the status.
      4. Optional. To set the write protection for the transferred objects in the target system, select the **Write Protected** option.
      5. Click **OK**.
4. Optional. To delete the selected scheduled transfer, click the **Delete** button.
5. Optional. To refresh the displayed properties of the scheduled transfer, click the **Refresh** button.

Displaying Transfer History

This function displays all transfers executed in which the current system was the target system.

To display the transfer history, complete the following steps.

1. On the **Navigator**toolbar, click the **Transfer History** button.
2. In the **Transfer History** dialog box, in the upper pane, click a transfer.
   Note The objects of the selected transfer display in the lower pane.

Revoking a Transfer

This function is only available when you have a versioning license installed and when you possess the **Administration(12)** right.

To revoke a transfer, complete the following steps.

1. On the **Navigator** toolbar, click the **Transfer History** button.
2. In the **Transfer History** dialog box, in the upper pane, click a transfer.

3. Click the **Revoke Transfer** button.

Notes

- Revoked transfers display in strike-through type in the list.
- You must revoke transfers in the order they were originally executed.
- If the transfer newly creates an object in a target system, revoking the transfer results in deleting the object from the target system.
- If the transfer restores an object from the recycle bin, the object is logically deleted and returned to the recycle bin when you revoke the transfer.

Displaying Deleted Objects

To display all logically deleted objects, complete the following step.

- On the **Navigator** toolbar, click the **Deleted Objects** button.
  Note The highest version of each object displays.

For more information, refer to the **Context menu of the Navigator**.

Restoring an Object

To restore an object, complete the following steps.

1. On the **Navigator**toolbar, click the **Deleted Objects** button.
2. In the **Deleted Objects** dialog box, select the objects you want to restore and click the **Restore Objects** button.
   Result
   The object is visible and editable in the Navigator again.

Deleting an Object Permanently

This function is only available when you possess the right **Delete Objects Permanently (37)**.

You cannot permanently delete an object if it is still used by a logically deleted object. You can only permanently delete an object if no current or previous version of another object uses the object you want to delete.

An object is classified as being used by another object, when:

- The object is a child object of another object.
  Example
  A text block is used by another text block.
- The object is referenced by another object.
  Example
  A script is executed by a text block event.

To permanently delete an object, complete the following steps.

1. On the **Navigator** toolbar, click the **Deleted Objects** button.
2. In the **Deleted Objects** dialog box, select the objects you want to delete permanently and click the **Permanently delete objects** button.
   Note This action deletes all previous versions of the object permanently from the database.

Context Menu of the Navigator Pane

The context menu in the lower pane of the **Navigator** provides specific object administration functions.

Menu item	Description
New	Opens a submenu to select a new object to be created.
Rename	Opens a dialog box to specify the new name for the object.
Delete (Link)	This menu item is only available in folders and deletes the selected object (link) from the folder.
Delete (Object)	Deletes the selected object. At first, this only means the object is deleted logically - it is marked as deleted in the database and no longer displayed in the **Navigator**. The object remains in the database with all its versions and, through the list of Deleted Objects can either be restored or permanently and physically deleted. As long as the object is still in the database, it is not possible to create a new object of the same type with the same name. **Version controlled objects can only be logically deleted if** • they are not being used by the current version of another object or • they are only being used by objects that themselves have been logically deleted. **Note** DB alias and folder objects are automatically physically deleted and cannot be restored.
Open	Opens the selected object. Alternatively, you can open all objects by double-clicking them.
Contains	Shows the first level objects used by the selected object.
Is Used By	Shows a list of objects using the selected object.

Used by Document	Shows a list of documents using the selected text block.
Copy	Copies the selected object to the clipboard.
Copy with Child Objects	Copies the object tree to the clipboard. A prefix and/or suffix for the new objects that will be created must be defined in the dialog box that is then displayed.
Add to new Transfer Job	Creates a new transfer job and adds the selected object to the transfer job. For more information, refer to Transfer Jobs.
Paste Link	This menu item is only available in folders and inserts the object you have copied as a link in the folder.
Paste	Inserts a an object you have copied.
Show Object History	Displays the complete version history of an object. For more information, refer to Show the object history. **Note** This menu item is only available when version control has been licensed accordingly.
Show Object Information	Lists various object information in the output pane.
Start in Content Composer Client	This menu item is available for documents and bundles and starts the selected object in the integrated Content Composer Client for test purposes.
Add to Debug Service	This menu item is only available for scripts and conditions. For more information, refer to Debug a script or a condition.
Remove from Debug Service	This menu item is only available for scripts and conditions. For more information, refer to Debug a script or a condition.

Object Inspector Pane

About the Object Inspector Pane

The **Object Inspector** pane enables you to edit properties and events of open objects.

Displaying the Object Inspector Pane

If the **Object Inspector** pane is not visible, complete the following step.

- In the **Studio** tab, click the **Object Inspector** button.

Context Menu of the Object Inspector Pane

The context menu of the **Object Inspector** pane offers the following options.

Menu item	Description
Reset	Resets the value of the property to the default value or deletes the value of the property.
Create	Creates a new script, condition or variable object definition.
Edit	Opens the script or condition assigned to the property.

Output Pane

About the Output Pane

The **Output** pane displays warnings, status messages, and error messages as well as console output.

Displaying the Output Pane

If the **Output** pane is not visible, complete the following step.

- In the **Studio** tab, click the **Output** button.

Context Menu of the Output Pane

The context menu of the **Output** pane offers the following options.

Menu item	Description
Clear all	Deletes all entries in the **Output** pane.
Copy	Copies the text selected in the **Output** pane to the clipboard.
Save output	Allows you save the content of the **Output** pane to a TXT file.

Toolbox Pane

About the Toolbox Pane

The **Toolbox** provides tools you need for modifying certain objects. Depending on the selected object, the toolbox can also be empty.

Displaying the Toolbox Pane

If the **Toolbox**pane is not visible, complete the following step.

- In the **Studio** tab, click the **Toolbox** button.

System Font

Content Composer uses the Windows system font **Segoe UI** to display all interface texts.

This font is capable of displaying all the special characters for European languages.

Note The Segoe UI font cannot be used to display certain mathematical special characters.

Administration

Odin Views

About Odin Views

The Odin views display printed processes, stacks, open jobs, and open envelopes.

To access the Odin Views, the user must possess the right **Odin Views in Studio (48)**.

Accessing Odin Views

To access the Odin views, complete the following step.

- On the **Administration** tab, click the **Odin Views** button, then click one of the submenu items.

Process View

About the Process View

The process view displays all pending print processes in the upper pane. The query **ViewProcess** defines the data that is shown.

The jobs of a process display in the lower pane. The query **ViewProcessJobs** defines the data that is shown.

Opening the Process View

To open the process view, complete the following steps.

1. On the **Administration** tab, click the **Odin Views** button and click **Processes**.
2. In the **Select Odin Database** dialog box, select a database and click **OK**.
   Note The **Select Odin Database** dialog box only displays if an Odin view is not open yet.
3. Optional. To copy the contents of a cell, click the cell and press CTRL + C.
4. Optional. To display the jobs of a process in the lower pane, in the upper pane, double-click a process.

Context Menu of the Process View

Menu item	Description
Lock Process	Sets the status of the process to **locked**.
Unlock Process	Resets the field ODP_Locked_by so that processing of the dataset can continue.
Reset status to	Sets the status of the process to **Imported** or **Rendition created**.
Set status to Delete	Sets the status of the process to **Delete**.
Creating Stack from Open Jobs	Creates a stack for printing or for deleting.
Show jobs of process	Displays the jobs of the process in the bottom window.
Show error view	Opens another pane to show any errors if applicable.
Select Columns	Opens a dialog window in which you can select the columns to display.
Sort by multiple columns	Opens a dialog window in which you can select the columns for sorting list results.
Refresh	Refreshes the process list.

Context Menu of the Jobs of a Process

Menu item	Description
Show print stream	Opens the corresponding document in the XPS viewer. For more information, refer to XPS Viewer.
Show Stack	Shows the corresponding stack.
Select Printer	Allows you to modify the printer.
Create Stack	Creates a stack for printing or for deleting. To do this, the user must have the **Create Odin stack for deleting/printing in the Odin view Processes (3002)** right.

Select all	Selects all jobs of the specified process.
Show error view	Opens another pane to show any errors if applicable.
Select Columns	Opens a dialog window in which you can select the columns to display.
Sort by multiple columns	Opens a dialog window in which you can select the columns for sorting list results.
Refresh	Refreshes the view.

Stack View

About the Stack View

The stack view displays print stacks. The query **ViewStacks** defines the data shown.

The jobs of a stack display in the lower pane. The query **ViewStackEditJobs** defines the data that is shown.

Opening the Stack View

To open the stack view, complete the following steps.

1. On the **Administration** tab, click the **Odin Views** button and click **Stacks**.
2. In the **Select Odin Database** dialog box, select a database and click **OK**.
   Note The **Select Odin Database** dialog box only displays if an Odin view is not open yet.
3. Optional. To copy the contents of a cell, click the cell and press CTRL + C.
4. Optional. To open a stack, double-click the stack row.
5. Optional. To show the print stream in the integrated XPS viewer, complete the following substeps.
    1. Double-click the stack.
    2. In the stack window, in the **Jobs of stack** pane, right-click and click **Show Print Stream**.

Context Menu of the Stack View

Menu item	Description
Reset Stack	Sets the status to w**ait**.

Release Stack	The term or function release stack refers to the action of removing the stack reference in jobs and envelopes, thereby freeing them up to be processed by a new stack.
Delete Stack	Sets the status of a stack to **delete.**
Lock Stack	Sets the status to **locked**.
Edit Stack	Displays the stack in a new tab.
Unlock Stack	Resets the field **ODS_Locked_by** so that a dataset can be further processed.
Reset Stack for Streaming	Sets the stack type to **streaming**.
Reset Stack To	Sets the status of a stack to the type selected.
Show Stream	Opens the print stream in an XPS viewer. For more information, refer to **XPS Viewer**.
Mark Stack	Sets a marker for the selected stack in the list header.
Show Stack	Displays the stack in a new tab.
Select all	Selects all stacks in the view.
Show error view	Opens another window to show any errors if applicable.
Select Columns	Opens a dialog window in which you can select the columns to display.
Sort by multiple columns	Opens a dialog window in which you can select the columns for sorting list results.
Refresh	Refreshes the view.

Context Menu of the Jobs of Stack View

Menu item	Description
Show Jobs of Envelope	Displays the jobs of the envelope in the bottom pane.

Show Stack	Shows the stack in the upper pane.
Show error view	Opens another window to show any errors if applicable.
Remove envelope from stack	Removes the envelope from the stack. To do this, the user must have the **Remove Odin envelope from stack (3511)** right. In addition, the stack must be set to **locked**.
Select Columns	Opens a dialog window in which you can select the columns to display.
Sort by multiple columns	Opens a dialog window in which you can select the columns for sorting list results.
Refresh	Refreshes the view.

Context Menu of the Envelopes of Stack View

Menu item	Description
Select Printer	Allows you to change the printer.
Reset Job To	Sets the stack to the selected status.
Show print stream	Opens the print stream in an XPS viewer.
Show Stack	Shows the stack in the upper pane.
Show error view	Opens another pane to show any errors if applicable.
Remove Job	Removes the job from the stack. To do this, you must have the **Remove Odin job from stack/envelope (3512)** right. In addition, the stack must be set to **locked**.
Select Columns	Opens a dialog window in which you can select the columns to display.
Sort by multiple columns	Opens a dialog window in which you can select the columns for sorting list results.

Refresh	Refreshes the view.

Context Menu of the Jobs of Envelopes View

Menu item	Description
Select Printer	Allows you to change the printer.
Reset Job To	Sets the stack to the status selected.
Show print stream	Opens the print stream in an XPS viewer.
Show Stack	Shows the stack in the upper pane.
Show error view	Opens another pane to show any errors if applicable.
Remove Job	Removes the job from the stack. To do this, you must have the **Remove Odin job from stack/envelope (3512)** right. In addition, the stack must be set to **locked**.
Select Columns	Opens a dialog window in which you can select the columns to display.
Sort by multiple columns	Opens a dialog window in which you can select the columns for sorting list results.
Refresh	Refreshes the view.

Open Job View

About the Open Job View

The open job view displays all open jobs. The query **ViewOpenJobs** defines the data that is shown.

Opening the Open Jobs View

To open the open jobs view, complete the following steps.

1. On the **Administration** tab, click the **Odin Views** button and click **Open Jobs**.

2. In the **Select Odin Database** dialog box, select a database and click **OK**.
   Note The **Select Odin Database** dialog box only displays if an Odin view is not open yet.
3. Optional. To copy the contents of a cell, click the cell and press CTRL + C.
4. Optional. To show the print stream in the integrated XPS viewer, right-click the open job and click **Show Print Stream**.

Context Menu of the Open Jobs View

Menu item	Description
Show print stream	Opens the print stream in an XPS viewer.
Creating stack	Creates a stack for printing or for deleting. To do this, you must have the **Create Odin stack for deleting/printing in the Odin view*Open Jobs*(3005)** right.
Show error view	Opens another pane to show any errors if applicable.
Select Columns	Opens a dialog window in which you can select the columns to display.
Sort by multiple columns	Opens a dialog window in which you can select the columns for sorting list results.
Refresh	Refreshes the view.

Open Envelopes View

About the Open Envelopes View

This view displays all open envelopes. The query **ViewOpenEnvelopes** defines the data that is shown.

Opening the Open Envelopes View

To open the open envelopes view, complete the following steps.

1. On the **Administration** tab, click the **Odin Views** button and click **Open Envelopes**.
2. In the **Select Odin Database** dialog box, select a database and click **OK**.
   Note The **Select Odin Database** dialog box only displays if an Odin view is not open yet.
3. Optional. To copy the contents of a cell, click the cell and press CTRL + C.

Context Menu of the Open Envelopes View

Menu item	Description
Show Jobs of Envelope	Displays the jobs of the envelope in the bottom window. The query **ViewOpenEnvelopesJobs** defines the data shown.
Show error view	Opens another pane to show any errors if applicable.
Select all	Selects all jobs of the specified envelope.
Refresh	Refreshes the view.
Select Columns	Opens a dialog window in which you can select the columns to display.
Sort by multiple columns	Opens a dialog window in which you can select the columns for sorting list results.
Release	Releases the envelope and deletes the reference in the jobs, thereby creating open jobs. **Note** The term **Release** refers to the action of logically destroying the envelope and freeing up the jobs to be packed in other logical envelopes.

XPS Viewer

The XPS Viewer allows you to view generated documents from the Odin views **Open Jobs** and **Stacks**and enables local printing of the document as well as searching for text in the document.

Odin Views Tools

The functions in this menu are available to all views.

Icon	Name	Description
	Refresh View	Refreshes the view.
	Previous Page	Scrolls one page back.
	Next Page	Scrolls one page forward.

	Select Filter	Allows you to modify the filter.
	Change active filter conditions	Allows you to modify the parameters of the current filter.
	Show View Information	The information window displays the parameters of the current query. The field **DataProviderDefinition** shows the name of the query definition. For more information, refer to Query Definitions.

Overview of Odin Statuses

Processes

ODP_STATE	Meaning
0	Imported
1	Rendition created
2	Locked
3	Delete

Stack Status

ODS_STATE	Meaning
0	Wait
1	In Progress
2	On Hold (printed)
3	Locked (in the event of an error)
4	Delete

Stack Type

ODS_STACKTYPE	Meaning
0	Print (default)
10 *	Postage Optimization
20 *	Streaming
30 *	Convert
40 *	StreamPrint
50	Archive
60	Transfer (Example of a Custom Process)
99	Finished

* = The stack type cannot be set and will result in an error if set in a script.

Jobs

ODJ_STATE	Meaning
0	Created (job was inserted into the database)
1	Rendition created
2	Error creating rendition
3	Printed
4	Error while printing

Envelopes

ODE_STATE	Meaning
0	Created
1	IT Franking

11	In print
12	Printed
13	Error while printing

Query Assistant

About Query Assistant

The Query Assistant enables you to run predefined queries. For more information on queries, see the Query Definitions.

To access the Query Assistant, the user must possess the right **Query Assistant (49)**.

Accessing the Query Assistant

To access the Query Assistant, complete the following step.

- On the **Administration** tab, click the **Query Assistant** button.

Run a Predefined Query

To run a predefined query, complete the following steps.

1. In the **Select the required query definition** window, select a query definition from the list and click **Next**.
2. In the **Select the required condition** window, select a condition and click **Next**.
3. Optional. If the query definition contains variables, in the **Enter the parameter values** window, enter the values.
4. Click **Next**.

DB Alias Administration

About DB Alias Administration

In a database alias, you specify the connection information to a data source, for example, to a database.

To access the DB Alias Administration, the user must possess the right **DB Alias Administration (50)**.

Accessing DB Alias Administration

To access the DB Alias Administration, complete the following step.

- On the **Administration** tab, click the **DB Alias Administration** button.

Creating a Database Alias

To create a database alias, complete the following steps.

1. On the **DB Alias Administration** tab, click the **Create New DB Alias** button.
2. In the **Create New DB Alias** dialog box, type a name for the DB alias and click **OK**.
3. In the **Connection Settings** dialog box, complete the following substeps.
    1. Under **Data Provider**, select the preferred data provider.
    2. Under **SQL Template**, click the **Standard Value** button.
    3. Under **SQL Field Template**, click the **Standard Value** button.
    4. Click the ellipsis button to the right of the **Connection String** box.
    5. In the **Data Link Properties** dialog box, specify the connection settings and click **OK**.
    Note For information about the settings for a connection to an MS SQL Server, refer to Connection settings for MS SQL Server.
    6. Optional. In the **Connection Settings** dialog box, select the **Encrypt Password** check box.
    8. To check your settings, in the **Connection Settings** dialog box, click the **Check Connection** button.
    9. Click **OK**.

Modifying the Connection Settings

To modify the connections of a DB alias, complete the following steps.

1. In the **DB Alias Administration** window, right-click a DB alias and click **Connection Settings**.
2. In the **Data Link Properties** dialog box, modify the connection settings and click **OK**.
Note For information about the settings for a connection to an MS SQL Server, refer to Connection settings for MS SQL Server.
3. In the **Connection Settings** dialog box, click the **Check Connection** button.

Connection Settings for MS SQL Server

To specify connection settings for a Microsoft SQL Server, in the **Connection Properties** dialog box, complete the following steps.

1. On the **Common** tab, from the **Select or enter a server name** drop-down menu, select a server or type a server name.
2. Complete one of the following substeps.
    - To use Windows authentication, select the **Use Windows Authentication** option.
    - To specify user and password, complete the following substeps.

1. Select the **Use SQL Server Authentication** option.
2. In the **User name** box, type a user name.
3. In the **Password** box, type the password.

3. Complete one of the following substeps.
   - From the **Select or enter a database name** drop-down menu, select a database or type a database name.
   - To attach a database file, complete the following substeps.
     1. Click the **Browse** button to the right of the **Attach a database file** box.
     2. In the **Select SQL Server Database File** dialog box, select a database file and click **Open**.
     3. In the **Logical name** box, type a name.
4. Optional. To use encryption for the database connection, complete the following substeps.
   Note On your **MSSQL Server**, you need to specify the appropriate settings for data encryption.
   1. In the **Connection Properties** dialog box, click the **Advanced** tab.
   2. Double-click the **Encrypt** entry.
   3. In the **Edit Property Value** dialog box, from the **Property Value** drop-down menu, select **True** and click **OK**.
5. To check your settings, click the **Test Connection** button.

Deleting a Database Alias

You can only delete aliases that are not in use.

To delete a DB alias, complete the following steps.

1. In the **DB Alias Administration** window, select the aliases you want to delete.
2. On the **DB Alias Administration Tools** tab, click the **Delete DB Alias** button.
   Note The selected alias is first marked for deletion. After saving, the alias is then deleted permanently.

Exporting a Database Alias

You can export a database alias to an XML file and then load the database alias from that file instead of from the database.

For information on how to load a database alias from a file, see "Loading a Database Alias from a File" in the *Content Composer Advanced Design and Setup Guide*.

Complete the following steps.

1. In the **DB Alias Administration** window, select the aliases you want to export.
2. On the **DB Alias Administration Tools** tab, click the **Export DB Alias as XML** button.
3. In the **Save As** dialog box, enter a name for the XML file and click **Save**.

Repository Import

About Repository Import

Repository Import enables you to import previously exported objects. For information about exporting objects, see the Export objects.

To access Repository Import, the user must possess the right **Transfer systems (10)**.

Accessing Repository Import

To access Repository Import, complete the following step.

- On the **Administration** tab, click the **Repository Import** button.

Importing Objects

Note To allow the import of work versions, open the **System Administration** and set the **Import Work Versions** property of the appropriate system to `True`.

To import objects into the repository, complete the following steps.

1. In the **Repository Import** window, in the **Specification of the import directory,** type the name of the directory from which you want to import the objects.
   Note The directory must be a subdirectory below the configured RWS work directory. The default setting for this directory is [*drive:*]\ContentComposer\Admin\rws_work\.
2. Optional. If objects should be flagged as write-protected in the target system, select the **Set Write Protection** check box.
3. On the **Repository Import** tab, click the **Execute** button.

System Synchronization

About System Synchronization

System Synchronization enables you to compare the objects in two systems and to synchronize these systems.

To access System Synchronization, the user must possess the right **Synchronize systems (11)**.

Accessing System Synchronization

To access System Synchronization, complete the following step.

- On the **Administration** tab, click the **System Synchronization** button.

Synchronize Two Systems

To synchronize two systems, complete the following steps.

1. On the **System Synchronization** tab, select the systems you want to compare.
2. To display the list of objects in both systems, on the **System Synchronization** tab, click the **Compare Systems** button.
   - Objects found in the one system but missing in the other display in red. You can mark them for deletion.
   - Objects with different object names in each system display in blue.
   - Objects that differ in version number display in black italics.
   - Objects that differ in version number and name display in blue italics.
3. Optional. To mark an object for deletion, right click the object and click **Delete**.
4. On the **System Synchronization** tab, click the **Execute Synchronization** button.

System Administration

About System Administration

System Administration enables you to create and administrate systems, to manage the dependencies between systems, and to create database connections.

To access the System Administration, the user must possess the right **System Administration (51)**.

Accessing System Administration

To use this function, the user must possess the right **System Administration (51)**.

To access System Administration, complete the following step.

- On the **Administration** tab, click the **System Administration** button.

Creating a Database Connection

To create a database connection, complete the following steps.

1. From the **Toolbox**, drag a **New DB Alias** module to the System Administration **Editor** window.
2. In the **Create New DB Alias** dialog box, type a name for the alias and click **OK**.
3. In the **Connection Settings** dialog box, complete the following substeps.
   1. Under **Data Provider**, select the preferred data provider.
   2. Click the ellipsis button to the right of the **Connection String** box.

3. Complete one of the following substeps.

   - For Native SQL

     To specify connection settings for a Microsoft SQL Server, in the **Connection Properties** dialog box, complete the following steps.

     1. On the **Common** tab, from the **Select or enter a server name** drop-down menu, select a server or type a server name.
     2. Complete one of the following substeps.
        - To use Windows authentication, select the **Use Windows Authentication** option.
        - To specify user and password, complete the following substeps.
          1. Select the **Use SQL Server Authentication** option.
          2. In the **User name** box, type a user name.
          3. In the **Password** box, type the password.
     3. Complete one of the following substeps.
        - From the **Select or enter a database name** drop-down menu, select a database or type a database name.
        - To attach a database file, complete the following substeps.
          1. Click the **Browse** button to the right of the **Attach a database file** box.
          2. In the **Select SQL Server Database File** dialog box, select a database file and click **Open**.
          3. In the **Logical name** box, type a name.
     4. Optional. To use encryption for the database connection, complete the following substeps.
        Note On your **MSSQL Server**, you need to specify the appropriate settings for data encryption.
          1. In the **Connection Properties** dialog box, click the **Advanced** tab.
          2. Double-click the **Encrypt** entry.
          3. In the **Edit Property Value** dialog box, from the **Property Value** drop-down menu, select **True** and click **OK**.
     5. To check your settings, click the **Test Connection** button.

- For Microsoft Oracle
    1. On the **Common** tab, in the **Enter a server name** box, type a server name.
    2. In the **User name** box, type a user name.
    3. In the **Password** box, type a password
    4. To check your settings, click the **Test Connection** button.
    5. Optional. Click the **Advanced** tab and change the required settings.

4. Optional. In the **Connection Settings** dialog box, select the **Encrypt Password** check box.
5. To check your settings, in the **Connection Settings** dialog box, click the **Check Connection** button.
6. Click **OK**.

Creating a New System

To create a new system, complete the following steps.

1. From the **Toolbox**, drag a **New System** module to the System Administration **Editor** window.
2. In the **Create New System** dialog box, in the **Name** box, type a name for the system.
3. In the **SystemObjectID** box, type an object id for the system and click **OK**.
   Note You can only edit the system object ID at this stage of creating the system. The ID must be unique.
4. To link the system to a DB alias, in the **Editor** window, move the pointer over the system, click the red dot and drag the pointer to a DB alias.

Notes

- After saving a link to a DB alias, no further editing is possible.
- You cannot move existing systems from one database to another.
- On saving, the system is assigned a **DefaultAliasTable** and a **DefaultLookupTable**. Both tables are created in the new system.

Assigning a Word Template

By default, each system uses the Normal.dotm specified in the configuration file to open or generate documents.

Document Access or Generation via...	Configuration File

Studio	Composer.Studio.exe.config
Client	Composer.MWS.exe.config If WCF console is used: Composer.WCF.Console.exe.config.
Custom Service	Custom Service configuration file For more information, refer to "Install a Windows service " in Content Composer Configuration Guide.

To use a different Normal.dotm for a specific system, assign a Word template to the system in the system administration.

To assign a Word template to a system, complete the following steps.

1. Open the system to which you want to assign the Word template.
2. Optional. Create a new Word template.
3. Open the system administration.
4. In the **Editor** window of the system administration, click the current system.
5. In the **Object Inspector** pane, click the **Word Template** property and click the ellipsis button next to the property.
6. In the **Select an object** dialog box, select the preferred Word template and click **Select**.

Modifying a Database Connection

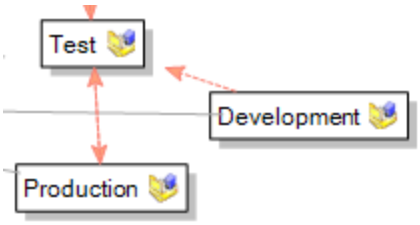
To modify the connection settings of a database connection, complete the following steps.

1. In the **System Administration Editor** window, double-click the database connection you want to modify.
2. In the **Connection Settings** dialog box, modify the settings.
   Note For information about the settings for a connection to an MS SQL Server, refer to Connection settings for MS SQL Server.

Defining a Transfer System

Transfer systems enable you to copy objects from a source system to one or more target systems with the **Navigator** function Add to new transfer job. You can also use Transfer Jobs

Red arrows depict the transfer of one system to another.



To define a transfer system, complete one of the following steps.

- In the **System Administration Editor** window, move the pointer over the source system, click the red dot and drag the pointer to the target system.
- Complete the following substeps.
  1. In the **Editor** window of the system administration, double-click the source system.
  2. In the **Transfer Targets** dialog box, select one or more target systems and click **OK**.

Deleting a System

To delete a system, complete the following steps.

1. Open the system administration.
2. In the **Editor** window of the system administration, click the system that you want to delete.
3. In the **System Administration** tab, click the **Delete Object** button.
   Notes The selected system is first marked for deletion. After saving, the system is then deleted permanently.
   You cannot delete the opened system.

Restoring a System

Use the **Restore Systems** function to search for systems that are stored in the database but do not currently display in the system administration.

To search for systems in a database, complete the following steps.

1. Optional. If not already available, create a DB alias for the database you want to search and save the system administration.
2. In the **System Administration** tab, click the **Restore Systems** button.
3. In the **Unconfigured Systems** dialog box, select one or more systems that you want to add to the system administration and click **Restore**.

System Properties

Property	Description
DB Alias	Database containing the system.
DefaultAliasTable	Default system alias table.
DefaultLookupTable	Default system lookup table.
Import Work Versions	TRUE = Work versions are imported. FALSE = Work versions are not imported.
Description	Optional explanatory text.
LastModified	Date and time of last modification.
Read-only Transfer	TRUE = Objects are transferred with write protection. FALSE = Objects are transferred without write protection.
Tag	Optional text. You can read the text specified here by a MonaLisa function at runtime.
Word Template	Standard Word template for this system.
ObjectName	Unique system name.
SystemObjectID	Unique system ID.

Remote Control

About Remote Control

Remote Control enables you to monitor and control services, processes, and sub-processes on various computers. For information about configuration, see the *Content Composer Advanced Design and Setup Guide*.

To access Remote Control, the user must possess the right **Remote Control (52)**.

Notes

- The logged in Windows user must have sufficient rights to gain access to remote computers.
- Users without Administrator rights should not be permitted to start the Remote Control module.
- When working on operating systems with User Account Control (UAC), an error message indicating "HTTP could not register URL" might occur due to missing permissions.
  To resolve this issue, execute the following command **as an administrator**. Use the value specified in the error message as *[Port]*.

netsh http add urlacl url=http://+:[*Port*]/ user=[DOMAIN]\[UserName]

Accessing Remote Control

To access Remote Control, complete the following step.

- On the **Administration** tab, click the **Remote Control** button.

Control a Hosted Process

To control a hosted process, complete the following steps.

1. In the **RuntimeRemoteControl** window, click the process you want to start.
2. In the **RuntimeRemoteControl Tools** tab, click the **Start Service** button.
3. Optional. To refresh the view automatically after any status modification, complete the following substeps.
   1. In the **RuntimeRemoteControl Tools** tab, click the **Activate automatic refresh** button.
   2. In the **Editor** window, observe the messages in the **Status** column.

RuntimeRemoteControl Tools

Icon	Name	Description
	Start Service	Starts the service.
	Stop Service	Stops the service.
	Pause Service	Pauses the service.

	Resume Service	Resumes the service.
	Refresh	Refreshes the view.
	Clear Cache	Clears the repository cache.
	Activate automatic refresh	The view refreshes automatically after any status modification.
	Deactivate automatic refresh	The view does not refresh automatically after any status modification.

Odin Import Procedure Manager

About Odin Import Procedure Manager

The Odin Import Procedure Manager enables you to create a SQL script that you can use to extend the stored procedure INSERT_JOB.

Note The stored procedure INSERT_JOB inserts job data into the table ODIN_JOB. This data is available for further processing.

To access the Odin Import Procedure Manager, the user must possess the right **Odin Import Procedure Manager (53)**.

Accessing Odin Import Procedure Manager

To access Odin Import Procedure Manager, complete the following step.

- On the **Administration** tab, click the **Odin Import Procedure Manager** button.

Note If the Odin Import Procedure Manager is not visible on the **Administration** tab, refer to Activate Odin Import Procedure Manager.

Activating Odin Import Procedure Manager

If the Odin Import Procedure Manager is not available in the **Administration**tab, complete the following steps.

1. Close **Content Composer Studio**.
2. From the **%Composerdir%** directory, open **studioprofile.xml** with a text editor that supports UTF-

8.

3. Find the following line.
   ```
   <ModuleInfo
   AssemblyFile="ModusSuite.Studio.OdinImportProcedureManager.dll" />
   ```
4. Remove the comment character **<!--** at the beginning of the line and **-->** at the end of the line.
5. Save and close the file.
6. Start **Content Composer Studio**.

Extending the Stored Procedure INSERT_JOB

To extend the stored procedure INSERT_JOB, complete the following steps.

1. From the **%Composerdir%** directory, open **odinparameter.config** with a text editor.
2. At the end of the file, add one or more lines containing the parameter name, alias, and, default value.
   Example
   ```
   <parameter name="ODJ_ORDERNR" type="string" direction="i" size="10"
   alias="ORDER_NR" defaultvalue="0"/>
   ```
3. Save and close the file.
4. Open the database administration where the Odin tables are located, for example SQL Server Management Studio.
5. Add one or more data fields to the **ODIN_JOB** table.
   Note The data field names must match the parameter names **(parameter name)** in the **odinparameter.config** file.
6. In **ContentComposer Studio**, on the **Administration** tab, click **Odin Import Procedure Manager** .
7. In the **Editor** window of the Odin Import Procedure Manager, click the preferred database connection.
8. In the **Path to OdinParameterConfig file** box, type the path to the **odinparameter.config** file.
9. In the **Output file** box, type the path and the name of the output file, for example, **C:\NewParam.sql**.
10. To generate the SQL script defined in the previous step, on the **Odin Import Procedure Manager** tab, click **Generate** .
11. In the database administration where the Odin tables are located, run the SQL script.

Transfer data from a bundle to the ODIN_JOB table

To transfer customer-specific data for each document in a bundle to the ODIN_JOB table, complete the following steps.

1. Open the bundle from which you want to transfer data to the **ODIN_JOB** table.
2. In the **Object Inspector** pane, assign a script to the **InitializePrintAndArchive** event.
3. In the script, use the function **SetArchiveAttribute** to assign data for each data field you added in the odinparameter.config file.

   Example
   ```
   SetArchiveAttribute("ORDER_NR","1234567890")
   ```
   Note The first parameter (in the ORDER_NR example above) must correspond to the alias used in the odinparameter.config file.

Transfer Data from a Document to the ODIN_JOB Table

To transfer customer-specific data from a document to the ODIN_JOB table, complete the following steps.

1. Open the bundle that contains the respective document.
2. In the **Editor** window of the bundle, click the document.
3. In the **Object Inspector** pane, assign a script to the **DefinitionArchiving** event.
4. In the script, use the function **SetArchiveAttribute** to assign data for each data field you added in the **odinparameter.config** file.
   Example
   SetArchiveAttribute("ORDER_NR","1234567890").
   Note The first parameter (in the example above **ORDER_NR**) must correspond to the **alias** used in the **odinparameter.config** file.

User Rights

About User Rights

You can manage Content Composer users either in the Windows user administration tool or in LDAP.

A user must first authenticate before starting a Content Composer program. This is done by logging in using the Windows user or by typing the user name and password.

For more information, refer to "Studio start parameters" in the *Content Composer Advanced Design and Setup Help*.

Profiles

About Profiles

Profiles enable you to collate user-specific attributes.

If you use LDAP, the attributes of the LDAP user are mapped to the corresponding attributes.

Role profiles

If you create a profile for an LDAP role, do not use the same attributes as for a user profile.

In this case, the system would overwrite the role profile attributes with the assigned or default values of the user profile attributes.

Creating a Profile

To create a profile, complete the following steps.

1. In the bottom pane of the **Navigator**, right-click and click **New** > **Profile**.
2. In the **Create New Profile** dialog box, type a name for the profile and click **OK**.

Adding an Attribute to a Profile

To add an attribute to a profile, complete the following steps.

1. On the **Identity Editor Tools** tab, click the **Add Attribute** button.
2. In the **CreateNew Profile Attribute** dialog box, type a name for the attribute and click **OK**.
   Note You cannot add the attributes **UserId** and **UserRoleNames** to profiles because they are automatically created on authentication.
3. In the **Object Inspector** pane, click the **Mapping** property and type the name of the attribute from the **Active Directory**.
4. Optional. In the **Object Inspector** pane, type a value for the property **Value**.
   Note This value is assigned to the attribute when the **Mapping** property is not populated or the entry defined there does not exist.
5. Optional. To define which profile is used, edit the **UserRepository.config** file.
   For more information, see the *Content Composer Advanced Design and Setup Help*.
6. To enforce your modifications, restart the service **Composer.Core**.

Roles

About Roles

Roles enable you to assign rights and privileges.

Creating a Role

To create a role, complete the following steps.

1. In the bottom pane of the Navigator, right-click and click **New** > **Role**.
2. In the **Create New Role** dialog box, type a name for the role and click **OK**.
3. From the **Navigator**, drag the preferred profile to the **Editor** window.
   Note If you modify the profile after you added it to a role, remove and re-add the profile to the role. Otherwise, the changes will not be taken into account.
4. Optional. To allow or deny a global right, in the **Editor** window, in the **Global Rights** tab, select or clear the corresponding check box in the **Assign** column.
   Note You can use the context menu in the **Global Rights** tab to quickly allow or deny all rights.
5. Optional. To allow or deny an object right, in the **Editor** window, in the **Object Rights** tab, select or clear the corresponding check box in the columns **Create, Read, Edit,** and **Delete**.
   Note You can use the context menu in the **Object Rights** tab to quickly allow or deny all rights.
6. Optional. To allow or deny access to a system, in the **Editor** window, in the **System Rights** tab, select or clear the corresponding check box in the **Allowed** column.
   Note You can use the context menu in the **System Rights** pane to quickly allow or deny access to all systems.
7. To enforce your modifications, restart the service **Composer.Core**.
   Note To disable or to enable user role access control, edit the **Composer**.**core.exe.config** file and the **Composer.MWS.exe.config** file, and any existing Windows Service Host configuration file. For more information, see the *Content Composer Advanced Design and Setup Guide*. With user role access control enabled, access is denied for each system for which a role is not configured explicitly to have access.

Role Mappings

About Role Mappings

Role mappings enable you to define user groups.

Creating a Role Mapping

To create a role mapping, complete the following steps.

1. In the bottom pane of the **Navigator**, right-click and click **New** > **Role Mapping**.
2. In the **Create New Role Mapping** dialog box, type a name for the role mapping and click **OK**.

3. To add a group, on the **Identity Editor Tools** tab, click the **New Group** button.
4. In the **Create New Group** dialog box, type a name for the group and click **OK**.
   Note The user group name must correspond to an existing group name in Windows or LDAP.
5. From the **Navigator**, drag the preferred role to the group in the **Editor** window.
6. Optional. To define which role mapping is used, edit the **UserRepository.config** file.
   Note For more information,see "Configure User Management" in the *Content Composer Advanced Design and Setup Help*.
7. To enforce your modifications, restart the service **Composer.Core**.

Note If a user is assigned to several groups, the user's rights are an accumulation of the rights assigned to the respective groups.

Text Blocks

About Text Blocks

Text blocks are integral parts of a document. As a rule, they comprise of a text fragment containing both standard fixed text and/or text variables.

You can nest text blocks within each other to any depth.

Creating a Text Block

To create a text block, complete the following steps.

1. In the bottom pane of the **Navigator**, right-click and click **New** > **Text Block**.
2. In the **Create New Text Block** dialog box, type a name for the text block and click **OK**.
3. Optional. To add text to the text block, perform the following substeps.
   1. On the **Editor Tools** tab, click the **Show Word Document** button.
   2. In **Microsoft Word**, type the content of the text block.
   3. In **Content Composer Studio**, save your changes.
4. Optional. **Insert variables**.
5. Optional. **Assign a data model**.
   Note To remove an existing data link, right-click and click **Remove Data Link.**
6. Optional. **Insert child text blocks**.
7. Optional. In the **Object Inspector** pane, edit the **properties** of the text block.
8. Optional. To modify the status of the text block, right-click the text block and select one of the following menu items.

- **Always active**. Sets the value of the **TextInsertionRule** property to **required**.
- **Activate**. Sets the value of the **TextInsertionRule** property to **activated**.
- **Deactivate**. Sets the value of the **TextInsertionRule** property to **deactivated**.
- **Disabled**. Sets the value of the **TextInsertionRule** property to **disabled**.

For more information,see the **General text block properties**.

Text Block Properties

Show All

Conditions

Property	Description
Condition	If a condition is assigned and returns the result `True`, the text block inserts into the document. If the condition returns the result `False`, the text block does not insert into the document. For more information, refer to Assign a condition to an object.

Data Model

Property	Description
Selection	Stores the associated selection.

Events

Event	Description
AfterInsert	The script defined here runs after the text block inserts into the document. Context: TextComponent
BeforeInsert	This script runs before the text block inserts into the document. Context: TextComponent

General

<table>
<tr><th>Property</th><th>Description</th></tr>
<tr><td>AttributeInheritance</td><td>Can only be modified through the context menu of a text block during document editing.
For more information, refer to Inherit Text Block Attributes.</td></tr>
<tr><td>Expandable</td><td>Defines if the user can see the text block structure and is allowed to activate or deactivate its objects.</td></tr>
<tr><td>DataLink</td><td>Path to the data object.</td></tr>
<tr><td>DataSource</td><td>
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>None</td><td>No data assigned.</td></tr>
<tr><td>Pool</td><td>Data retrieves from the pool assigned variable.</td></tr>
<tr><td>Data Model</td><td>Data retrieves from the assigned data model.
Linking to a data object also controls the number of times the object is inserted.
If no data exists in the address specified, the text block does not insert.</td></tr>
<tr><td>XPath</td><td>Data retrieves from the expression defined in the XPathExpression property.
This returns a number of nodes that determine how often the text block inserts.</td></tr>
</table>
</td></tr>
<tr><td>Multiple</td><td>Is set automatically dependent on the assigned data object.
TRUE = 1:N relationship.
FALSE = 1:1 relationship.
For more information, refer to Multiple Text Blocks.</td></tr>
<tr><td>Visible</td><td>Defines whether the text block is visible to end users.</td></tr>
<tr><td>TextBlockProtection</td><td>TRUE = Users cannot make any modifications to the text block.
FALSE = Users can make modifications to the text block.</td></tr>
</table>

TextInsertionRule

Value	Description
Activated	No abort is forced if the text block cannot be inserted during document generation.
Deactivated	The text block is deactivated and the user cannot activate it.
Disabled	The text block does not insert into the document.
Required	Document generation aborts if the text block cannot be inserted.

Title

The name of the text block the user sees in the client.

Default inheritance value

Value	Description
True	The text block inserts into a document or another text block with the property **AttributeInheritance=True** The properties of the inserted text block always correspond to those of the original text block and the user cannot modify the properties.
False	The text block inserts into a document or another text block with the property **AttributeInheritance=False** The user can modify the properties of the inserted text block.

For more information, refer to Inherit Text Block Attributes.

XPathExpression

To assign an XPath expression, complete the following steps.

1. In the Object Inspector, click the **XPatchExpression** property and click the ellipsis button next to the property.
2. In the **Open** dialog box, select an XML file.
3. In the **XPath Evaluator** dialog box, in the **Sample XML tree** box, right-click the variable that you want to assign to the text block variable, then click **Copy XPath**.
4. In the **XPath Expression** box**,** right-click and click **Paste**.
5. Click the **Evaluate** button.

6. Check the result in the **XPath Result** box.
7. Click the **Apply** button.

Examples

Adding Values with XPath

The function Sum returns the sum of all values of a node.

Note The comma character is not recognized, that means a dot (decimal point) must be used as the decimal separator in the XML file (U.S. spelling).

Syntax:

Sum(PARTNER/ACCOUNTS/MOV_assets)

sum(//ACCOUNT[1]/MOVEMENTS/MOV_assets)

//	Only looks for the node ACCOUNT irrespective of the parent node, that means looks for the node wherever it is found. There is no need to define the path to the node.
ACCOUNT[1]	Only the first node with the name ACCOUNT is read.
/MOVEMENTS	Child node
/MOV_assets	Element (with values)

Several instances of the node ACCOUNT exist, but only the sum of the values of a node element for one ACCOUNT node is to be calculated.

The node ACCOUNT is only run through once, but all the child nodes (MOVEMENTS) are evaluated.

The values are read by a multiple text block (1:n relationship) in Content Composer.

As such, the nodes all have the same structure and just different values.

The return value is a normal decimal value (for example $100.33) and therefore of the data type **Double**.

Other Useful Functions

Concat()	Allows you to combine strings to individually defined rules. Thus, contents read

	from elements can be combined with leading characters. **Example:** concat(Postbox ,PARTNER/ADDRESS/ADD_POB) **Return value:** Postbox 08 15
Count()	Counts the number of specified nodes found in the XML file. This is useful for 1:n relationships, when several nodes of the same name are found (multiple text blocks). **Example:** Count(PARTNER/ACCOUNTS)
Translate()	Replaces all lower cases with upper cases in the content read from the element name. **Example:** Translate(//ADD_country_ name,abcdefghijklmopqrstuvwxyz,ABCDEFGHIJKLMNOPQRSTUVWXYZ)
Character: //	Reads the content of all elements of the specified name, irrespective of where the element is found in the XML file. There is no need to define a particular path as only the element is looked for. **Example:** //MOV_liabilities

Object definitions

Property	Description
VariableObjectDefinition	Contains the settings for variable text blocks. To create a variable object definition, complete one of the following steps. • Drop an alias table on the text block.

- Right-click the **VariableObjectDefinition** property and click **Create**.

The following additional properties come into view. These properties determine which text block inserts.

- AttributeName
- DataLink
- DataSource
- LookupSource
- LookupKey
- XPathExpression

For more information, refer to Variable Objects.

Text Block Editor Tools

Icon	Name	Description
	Refresh	Refreshes text block structure if you modified child text blocks or selections while the text block is still open in editing mode. If you made modifications to a text block the function remains deactivated, refresh is done on saving.
	Open Text Block	Opens the selected text block.
	Add Variable	Inserts a new variable at the current cursor position.
	Apply Text Block and Variable Properties	For details, refer to **Apply text block and variable properties**.
	Show Word Document	Brings the Microsoft Word document of a text block to the foreground.
	Switch View	Switches between full screen mode and split screen mode with Word.
	Structure of a text block	For details, see the Text Block Structure.

	Used data connections	For details, see the Used Data Links.

Text Block Variables

About Text Block Variables

Text block variables are containers for variable data, for example, name and address.

Apart from text data, text block variables contain either image data or dynamic data charts.

Variable type	Content	Insertion method
Text	• Variable text data	As text block variable • in Content Composer Studio • in Microsoft Word For more information, refer to Insert a text block variable
Image	• Path and file name of the image or • Link to the image as variable value. For more information, refer to About image variables	As text block variable with image variable property • in Content Composer Studio • in Microsoft Word For more information, refer to Insert an image variable
Chart	• Chart that draws at runtime depending on given data. For more information, refer to About dynamic charts	As text block variable with image variable property and ChartDefinition properties in Content Composer Studio For more information, refer to Predefine a dynamic chart

Inserting a Text Block Variable

Show All

Two methods are available to insert variables into a text block.

- In Content Composer Studio

To insert a text block variable, complete the following steps.

1. Open the text block.
2. In the **Editor** window, below the bottom-right pane, click **Text Block Variables**.
3. Enter a name for the variable by completing one of the following options.
   - In the **Text Block Variables** pane, in the **Variable Name** box, type a name for the variable.
   - In the **Text Block Variables** pane, click the ellipsis button next to the **Variable Name** to open a dialog box where you can select a predefined variable.
     For more information, refer to **Define variable rules**.
4. Optional. Select the **as Image Variable** check box. For more information, refer to **Insert an image variable**.
5. Optional. In the **Default Value** box, type a default value for the variable. This value is used to populate the variable if no data field is defined.
6. Optional. To enter an input mask for the variable, complete one of the following options.
   - In the **Input Mask** box, type an input mask for the variable. For more information, refer to **Input masks**.
   - Click the ellipsis button next to the **Input Mask** box to open a dialog box and select a predefined input mask.
7. Optional. To assign a value help, perform the following substeps.
   1. Click the ellipsis button next to the **Value Help** box.
   2. In the **Select an Object** dialog box, in the lower pane, click one of the listed value help objects.
   3. Click **Select**.

   For more information, refer to **Value Helps**.
8. Optional. Clear the **Allow manual input** check box. This specifies whether a text block variable can be overwritten or whether it has to be filled by selecting a predefined value from a value help.
9. Click the **Insert** button. The variable is inserted at the corresponding cursor position in the Microsoft Word document.
10. Optional. To modify the order of variables, drag and drop the variable to the preferred position in the text block structure.
11. Optional. Check the position of the variable in Word and modify the position if necessary.
    Note Modifying the variable position in Word has no effect on the variable position in the text

block editor.

12. Optional. In the **Object Inspector** pane, edit the **properties** of the variable.
13. Optional. To modify the status of the variable, right-click and select one of the following menu items.

- **Required**. Sets the value of the **VariableType** property to **required**.
- **Optional**. Sets the value of the **VariableType** property to **optional**.
- **Free**. Sets the value of the **VariableType** property to **free**.

For more information, refer to **General text block properties**.

- In Microsoft Word

To insert a text block variable, complete the following steps.

1. Open the text block.
2. On the **Editor Tools** tab, click the **Show Word Document** button.
3. In Word, position the cursor to where you want to insert the variable.
4. In Word, on the **Content Composer** tab, click the **Insert Text Variable** button.
5. In the **Insert Text Variable** dialog box, in the **Variable Name** box, type a name for the variable.
6. Optional. In the **Default Value** box, type a default value for the variable.
7. Click **OK** to confirm.

Formatting a Text Block Variable

If the formatting of a variable in a text block should be passed to the document, complete the following steps.

1. In Word, select the content control of the variable.
2. Format the variable.

The following variable formats are passed on to the generated document.

- Font style
- Font color
- Highlight color
- Font style bold
- Font style italic
- Font style underlined
- Font style strike through

Special Variable Contents

Applying a Hyperlink as Content of a Variable

To apply a hyperlink as content of a standard variable, complete the following step.

- Format the variable content as follows.
  ```
  The visible content of the variable in the
  document|http://www.hylandcom
  ```

Notes

- The variable content consists of two parts, the visible content and the URL. The two parts are separated by a pipe sign ("|").
- In the generated document, the user can open the link by the key combination **CTRL + left mouse button** on the variable content.

Applying a Line Feed, Tab, Non-Breaking Hyphen or Non-Breaking Space

The following keywords in variable contents enable you to insert line feeds, tabs, non-breaking hyphens and non-breaking spaces.

Variable content	is converted to
\line	Line feed
|CRLF|	Line feed
\tab	Tab
\~	Non-breaking space
_	Non-breaking hyphen

Defining a Special Variable Content in the Property Formatting Mode

You can also define the special variable contents in the variable property **Formatting Mode** as **Type-Specific Formatting.**

To do this, complete the following steps.

1. In the **Editor** window, select the variable.
2. In the **Object Inspector**, click the property **Formatting Mode** and click the ellipsis button next to the input box.

3. In the **Output Formatting** dialog box, select the formatting mode **Type-Specific Formatting.**
4. In the field **Rule**, define the required formatting mode.
   Example
   To force a line feed after the variable content, type **%s\line**.

Note When using the WebClient, all line feed in variable content are displayed as #xA; when read from XML files. This particular XML-Parser behavior complies withW3C recommendationsfor XML and is described there as **Attribute-Value Normalization**.

Defining an Input Mask

Show All

Input masks are used to define input formats for variables, for example date.

An input mask is defined by the use of three parameters, separated from each other by a semicolon.

To define an input mask, complete the following steps.

1. In the **Editor** window, select the variable.
2. In the **Object Inspector** pane, type the input mask in the input box next to the property **Input Mask**.
3. The first value is used to define formatting characters.
   The following table is a list of the characters that can be used to define an input mask and their meaning.
   Formatting Characters

   This table displays the characters that you can use to define an input mask.

Characters	Description
!	Specifies that optional characters display as leading spaces. If "!" character is not specified, optional characters display as trailing spaces in the mask.
>	Converts all following characters to lower case until the end of the mask is reached or the character "<" is detected. **Note**: This input mask character is currently not supported by the Web Client.
<	Converts all following characters to upper case until the end of the mask is reached or the character ">" is detected. **Note**: This input mask character is currently not supported by the Web Client.

<>	If both characters are used together, no case sensitivity check is made. The data specified is formatted in exactly the same way a user types it.
\	The character following "\" can be used as a placeholder in the input mask.
L l	Valid entries are (A-Z, a-z). An entry is not compulsory.
A a	Only alphanumeric characters are valid (A-Z, a-z, 0-9). An entry is not compulsory.
C c	Any character is allowed. An entry is not compulsory. **Note**: The following characters are currently not supported by the Web Client. • - • / • (•) • . • : • Space • + • , • @ • [•] • “ • ‘
0 9	Only a numeric character is allowed. An entry is not compulsory.
#	Only a numeric character or a plus or minus sign is allowed. An entry is not compulsory. **Note**: The plus and minus signs are currently not supported by the Web

	Client.
:	Used to separate hours, minutes, and seconds in defining a time.
/	Used to separate the date entries day, month, and year from each other.
;	Used to separate the three mask parameters from each other.
_	The underline character _ automatically inserts a space in the text. The cursor skips over these characters when you type data.

Notes

- You can use any character which is not listed in the above table as a first parameter of the mask and as placeholder for input. These characters are inserted automatically and skipped by the cursor on data entry.
- You can also use the special mask characters listed in above table as placeholders as long as you specify the "\" character first.

4. Following this, use the character **1** to specify that any defined separators, for example slash or space, be passed on to the document.
   Using **0** means that separators specified in the formatting rule are not passed on.
5. As the third parameter, define the character to be used in the input box to display spaces or data not specified.

Mask Templates

To select a predefined input mask template, click the ellipsis button to the right of the **Input Mask** field.

Defining Variable Rules

Show All

To store frequently used input masks in a lookup table, complete the following steps.

System Administration

1. Open **System Administration**.
2. In the **System Administration** window, click the system for which you are defining the input masks.
3. In the **Object Inspector** pane, check which alias table is assigned to the property

DefaultAliasTable.

4. Optional. To modify the alias table, perform the following substeps.

    1. In the **Object Inspector** pane, click the property **DefaultAliasTable** and then the ellipsis button next to the input box.
    2. In the **Select an Object** dialog box, in the lower pane, click one of the listed alias table objects.
    3. Click **Select**.
    4. Save and close the **System Administration**.

Alias table

1. Open the alias table you have assigned as the **DefaultAliasTable**.
2. On the **Lookup Alias Table** tab, click the **New Key** button.
3. In the **Create New Key** dialog box, type **variablestemplate** as name of the key and click **OK**.
4. In the editor window, click the key **variablestemplate.**
5. **O**n the **Lookup Alias Table**tab, click the **New Reference** button.
6. In the **Select an Object** dialog box, in the lower pane, select a lookup table.
   Note You may have to create a new lookup table beforehand.
7. Save and close the alias table.

Lookup table

1. Open the lookup table which you assigned to the **variablestemplate** key.
2. Define the default variable rules in the following way.
   VarName=VarDefault|"VarInput".ExampleVar1=Value1|"!\(99999\)99999;1;_"Var2=Value2|"cc-99999;1;_"Var3=Value3|"000\000\00;1;_"
3. Save and close the lookup table.

Now, when you insert a new variable and click the ellipsis button next to the field **Variable Name**, the entries in the lookup table display for selection.

Manual Variables

An input dialog box is displayed for variables whose property Input Type allow a manual input during document generation.

If a data source is assigned the values read from the source are displayed as a default value.

Variable Properties

Show All

Conditions

Property	Description
Condition	If a condition is assigned and returns the result `True`, the variable inserts into the text block. If the condition returns the result `False`, the variable does not insert into the text block. For more information, refer to Assign a condition to an object.

Events

Event	Description
AfterInsert	This script executes after the variable is inserted into the text block. Context: TextVariable
BeforeInsert	This script executes before the variable is inserted into the text block. Context: TextVariable
OnGetManualVariable	When manual variables are queried, this event is executed for each variable. Input Type: **Always** or **DataConditional** Context: ManualVariable Within the script, the settings of the value help assigned to the text block variable can be queried or changed to allow dynamic value help. In order to pass dynamic value help objects to the client, the returned Manual Variable XML must include the manvar2 element. For more information, refer to the *Content Composer Configuration Guide*.

General

Property	Description
Attribute Name	Attribute name (database field or pool variable)

ChartDefinition	For more information, refer to Chart properties.
DataLink	Path to the data object or the name of a variable pool.
DataSource	<table><tr><th>Value</th><th>Description</th></tr><tr><td>None</td><td>If the property InputType specifies manual input, the variable is populated with this result. If the property InputType specifies no manual input, the variable is populated with the default value.</td></tr><tr><td>Pool</td><td>The variable is populated by the value from the pool variable assigned.</td></tr><tr><td>Data Model</td><td>The variable is populated by the value from the data model assigned.</td></tr><tr><td>XPath</td><td>The variable is populated by the value returned by the XPath expression defined in the property **XPathExpression**.</td></tr></table>
DefaultValue	Value applied if the variable is not populated. **Default values for a user** To populate a variable with values pertaining to the current user, use a default value according to the following syntax. `[ModusUser.AttributeName]` Using the value `[ModusUser.UserId]`, for example, you can make the default value for an input field the name of the currently logged in user. You can find an overview of already available attributes in the options dialog which is reached through the **Main Menu button - Options - User Data**. The property **InputType** must be set to the value **Always**.
Formatting Mode	Defines the formatting of the variable in the document. The following two formatting types are available. String Splitting To fill the content of a variable with filling characters, for example leading zeros, to a required length, use the formatting mode **String Splitting**. Start position and number of places play a role, when only a substring of the variable is to

be used.

Start position=3 and number of places=2 would mean using 2 characters starting from the third character.

Define a character as the filling character that will be used to fill the string to the specified length.

The default value for a filling character is a space.

Typespecific Formatting

Typespecific formatting allows you to define rules as to how variables are formatted when they are inserted into a document.

So, for example, the value 123 can be formatted and inserted as $123.00.

Use the **Data Type** to define the type of variable.

You can use the field **Rule** to define the formatting rule.

The Formatting rule contains simple characters and format identifiers. Simple characters are inserted into the result string as is. Format identifiers begin with "%" and assign a specific format to the result.

A format identifier has the following syntax:

%[width].[prec] type

After the "%" sign, the following definitions may be made:

- an optional width definition [width],
- an optional precision definition, that is denoted by a leading period [.prec] and
- a character that denotes the conversion type (type).

Width and precision are denoted by numbers (for example "%.10d). The width specification denotes the minimum length of the field for conversion.

If the result string is shorter than the minimum length. it is stocked up with spaces. As a rule, spaces are inserted before the value to produce a right alignment.

For all floating point formats, the characters for the decimal and thousand separators are taken from the DecimalSeparator and ThousandSeparator settings in the Windows Control Panel.

Example:

for the **value = 100**

the formatting rule **%.2m** returns

$100.00 as the result.

Conversion type

The following character are available for the conversion type and irrespective of upper or lower case.

Conversion Characters

Character	Meaning
d	**Decimal** The argument must be an integer value. The value is converted to a String comprised of decimal numbers. If the Format-String contains a precision specification, the resulting String must have at least the minimum number of decimal places. If it has less, the String is padded to the left with zeros to reach the minimum length.
e	**Scientific Notation** The argument must be a floating point value (float). The value is converted to a String in the form of "-d,ddd…E+ddd". If the number is negative, the String begins with a minus sign. A number is always set before a decimal separator.
.	The total number of places in the result String (including the number before the decimal comma) is determined by the precision specifier in the Format-String. If there is no specification, the default precision to 15 places is used. A plus or minus sign and at least three decimal places always follow the exponent "E" in a String.
f	**Fixed** The argument must be a floating point value (float). The value is converted to a String according to the following convention: "-ddd,ddd…". If the number is negative, the String begins with a minus sign. The number of decimal places is determined by the precision specification in the Format-String.

	If there is no specification, the default precision to 2 decimal places is used.
g	**General** The argument must be a floating point value (float). The value is converted to the shortest decimal string according to the fixed or scientific notation format. The number of significant places in the result string is defined by the precision specifier in the Format-String. If there is no specification, the default precision to 15 places is used. Trailing zeros are removed from the result string. A decimal point is only displayed when needed. The fixed-point format is used for the resulting string, if the number of digits before the decimal point lesser than or equal to the specified precision and the value is greater than or equal to 0.00001. Scientific notation is used in all cases.
n	**Number** The argument must be a floating point value (float). The value is converted to a String according to the following convention: "-d.ddd.ddd,ddd...". Format "n" is the same as format "f", except that the result string contains a thousands separator.
m	**Currency** The argument must be a floating point value (float). The value is converted to a string that represents an amount of currency. The conversion is based on the Windows global variables for CurrencyString, CurrencyFormat, NegCurrFormat, ThousandSeparator, DecimalSeparator und CurrencyDecimals. These variables are initialized with the settings stored in the

	Windows Control Panel tab Regional Settings, Currency. N.B. If the Format-String contains a precision specifier, this will override the value in the global variable CurrencyDecimals.
s	**String** The argument must be a character, a string or a PChar value. The String or, as the case may be, the character, is inserted instead of the format identifier. If the Format-String contains a precision specifier, this will determine the maximum length of the resulting strings. If the argument is a string that exceeds the maximum length, the string is truncated.
x	**Hexadecimal** The argument must be an integer value. The value is converted to a String comprising of hexadecimal digits. If the Format-String contains a precision specification, the resulting String must have at least the minimum number of digits. If it has less digits, the String is padded to the left with zeros to reach the minimum length.

Define the formatting rule in the In the field **Computation** to specify a computation formula, for example *10.

The placeholder for the value in the computation formula is always **%s**, because it is passed on as a string to the computation.

Input Mask

For more information, refer to Input Masks.

Input Type

Value	Description
None	No manual input
Always	Manual input is possible

	DataConditional: If the variable content is empty although a data source was assigned, manual input is possible.
	For more information, refer to Manual Variables.
Prompt	Prompt text for manual variables.
ValueHelp	For more information, refer to Value Helps.
ValueHelpOverwritable	Evaluated when a value help is assigned.

Value	Description
True	A value can be specified manually instead of making a selection from the value help.
False	Only a value from the value help can be selected.

Variable Type

Value	Description
Free	Free variable If this variable is not populated, it has no bearing on document generation.
Required	Variable is required If this variable is not populated, document generation is aborted. Required variables with manual input are displayed in red in the input dialog.
Optional	Variable is optional If this variable is not populated, a log entry is made to this effect.

XPathExpression

To assign an XPath expression, complete the following steps.

1. In the Object Inspector, click the **XPatchExpression** property and click the ellipsis button next to the property.
2. In the **Open** dialog box, select an XML file.
3. In the **XPath Evaluator** dialog box, in the **Sample XML tree** box, right-click

the variable that you want to assign to the text block variable, then click **Copy XPath**.

4. In the **XPath Expression** box**,** right-click and click **Paste**.
5. Click the **Evaluate** button.
6. Check the result in the **XPath Result** box.
7. Click the **Apply** button.

Examples

Adding Values with XPath

The function Sum returns the sum of all values of a node.

Note The comma character is not recognized, that means a dot (decimal point) must be used as the decimal separator in the XML file (U.S. spelling).

Syntax:

Sum(PARTNER/ACCOUNTS/MOV_assets)

sum(//ACCOUNT[1]/MOVEMENTS/MOV_assets)

//	Only looks for the node ACCOUNT irrespective of the parent node, that means looks for the node wherever it is found. There is no need to define the path to the node.
ACCOUNT[1]	Only the first node with the name ACCOUNT is read.
/MOVEMENTS	Child node
/MOV_assets	Element (with values)

Several instances of the node ACCOUNT exist, but only the sum of the values of a node element for one ACCOUNT node is to be calculated.

The node ACCOUNT is only run through once, but all the child nodes (MOVEMENTS) are evaluated.

The values are read by a multiple text block (1:n relationship) in Content Composer.

As such, the nodes all have the same structure and just different values.

The return value is a normal decimal value (for example $100.33) and therefore of the data type **Double**.

Other Useful Functions

Concat()	Allows you to combine strings to individually defined rules. Thus, contents read from elements can be combined with leading characters. **Example:** concat(Postbox ,PARTNER/ADDRESS/ADD_POB) **Return value:** Postbox 08 15
Count()	Counts the number of specified nodes found in the XML file. This is useful for 1:n relationships, when several nodes of the same name are found (multiple text blocks). **Example:** Count(PARTNER/ACCOUNTS)
Translate()	Replaces all lower cases with upper cases in the content read from the element name. **Example:** Translate(//ADD_country_ name,abcdefghijklmopqrstuvwxyz,ABCDEFGHIJKLMNOPQRSTUVWXYZ)
Character: //	Reads the content of all elements of the specified name, irrespective of where the element is found in the XML file. There is no need to define a particular path as only the element is looked for. **Example:** //MOV_liabilities

Lookup

Property	Description
LookupFormattingType	Output formatting after condition and lookup table evaluation. The following two formatting types are available.

String Splitting

To fill the content of a variable with filling characters, for example leading zeros, to a required length, use the formatting mode **String Splitting**.

Start position and number of places play a role, when only a substring of the variable is to be used.

Start position=3 and number of places=2 would mean using 2 characters starting from the third character.

Define a character as the filling character that will be used to fill the string to the specified length.

The default value for a filling character is a space.

Typespecific Formatting

Typespecific formatting allows you to define rules as to how variables are formatted when they are inserted into a document.

So, for example, the value 123 can be formatted and inserted as $123.00.

Use the **Data Type** to define the type of variable.

You can use the field **Rule** to define the formatting rule.

The Formatting rule contains simple characters and format identifiers. Simple characters are inserted into the result string as is. Format identifiers begin with "%" and assign a specific format to the result.

A format identifier has the following syntax:

%[width].[prec] type

After the "%" sign, the following definitions may be made:

- an optional width definition [width],
- an optional precision definition, that is denoted by a leading period [.prec] and
- a character that denotes the conversion type (type).

Width and precision are denoted by numbers (for example "%.10d). The width specification denotes the minimum length of the field for conversion.

If the result string is shorter than the minimum length. it is stocked up with spaces. As a rule, spaces are inserted before the value to produce a right alignment.

For all floating point formats, the characters for the decimal and thousand separators are taken from the DecimalSeparator and ThousandSeparator settings in the Windows Control Panel.

Example:

for the **value = 100**

the formatting rule **%.2m** returns

$100.00 as the result.

Conversion type

The following character are available for the conversion type and irrespective of upper or lower case.

Conversion Characters

Character	Meaning
d	**Decimal** The argument must be an integer value. The value is converted to a String comprised of decimal numbers. If the Format-String contains a precision specification, the resulting String must have at least the minimum number of decimal places. If it has less, the String is padded to the left with zeros to reach the minimum length.
e	**Scientific Notation** The argument must be a floating point value (float). The value is converted to a String in the form of "-d,ddd…E+ddd". If the number is negative, the String begins with a minus sign. A number is always set before a decimal separator.
.	The total number of places in the result String (including the number before the decimal comma) is determined by the precision specifier in the Format-String. If there is no specification, the default precision to 15 places is used. A plus or minus sign and at least three decimal places always follow the exponent "E" in a String.
f	**Fixed** The argument must be a floating point value (float). The value is converted to a String according to the following convention: "-ddd,ddd…".

	If the number is negative, the String begins with a minus sign. The number of decimal places is determined by the precision specification in the Format-String. If there is no specification, the default precision to 2 decimal places is used.
g	**General** The argument must be a floating point value (float). The value is converted to the shortest decimal string according to the fixed or scientific notation format. The number of significant places in the result string is defined by the precision specifier in the Format-String. If there is no specification, the default precision to 15 places is used. Trailing zeros are removed from the result string. A decimal point is only displayed when needed. The fixed-point format is used for the resulting string, if the number of digits before the decimal point lesser than or equal to the specified precision and the value is greater than or equal to 0.00001. Scientific notation is used in all cases.
n	**Number** The argument must be a floating point value (float). The value is converted to a String according to the following convention: "-d.ddd.ddd,ddd…". Format "n" is the same as format "f", except that the result string contains a thousands separator.
m	**Currency** The argument must be a floating point value (float). The value is converted to a string that represents an amount of currency.

	The conversion is based on the Windows global variables for CurrencyString, CurrencyFormat, NegCurrFormat, ThousandSeparator, DecimalSeparator und CurrencyDecimals. These variables are initialized with the settings stored in the Windows Control Panel tab Regional Settings, Currency. N.B. If the Format-String contains a precision specifier, this will override the value in the global variable CurrencyDecimals.
s	**String** The argument must be a character, a string or a PChar value. The String or, as the case may be, the character, is inserted instead of the format identifier. If the Format-String contains a precision specifier, this will determine the maximum length of the resulting strings. If the argument is a string that exceeds the maximum length, the string is truncated.
x	**Hexadecimal** The argument must be an integer value. The value is converted to a String comprising of hexadecimal digits. If the Format-String contains a precision specification, the resulting String must have at least the minimum number of digits. If it has less digits, the String is padded to the left with zeros to reach the minimum length.

Define the formatting rule in the In the field **Computation** to specify a computation formula, for example *10.

The placeholder for the value in the computation formula is always **%s**, because it is passed on as a string to the computation.

LookupSource	Object to be looked up in.
LookupDefault	Default value when no value is found in the lookup table.
LookupKey	Key value to pass.

	Note Use `%s` to pass the content of the variable.
LookupValue Condition	Condition evaluated after the look up. If the condition returns the result `True`, the variable inserts into the text block. If the condition returns the result `False`, the variable does not insert into the text block.

Image Variables

About Image Variables

Image variables contain the path and file name of the image or a link to the image as variable value.

This image will then insert into the text box dynamically when the user generates the document.

Example

- `C:\Logo.jpg`

Notes

- If a variable contains a link value, the image inserts as a link (not embedded).
- If the document created is then modified and saved, the image is downloaded and saved in the document (embedded).

Supported image file types

- JPEG
- GIF
- TIFF
- PNG

Inserting an Image Variable

Show All

Two methods are available to insert image variables into a text block.

- In Content Composer Studio

To insert an image variable, complete the following steps.

1. Open the text block.
2. In the **Editor** window, below the bottom-right pane, click **Text Block Variables**.

3. In the **Text Block Variables** pane, in the **Variable Name** box, type a name for the variable.
4. Select the **as Image Variable** check box.
5. Optional. In the **Default Value** box**,** type the path and file name of the image or a link to the image. This value is used to populate the variable if no data field is defined.
6. Click the **Insert** button. In **Microsoft Word**, the variable is inserted in a text box that you can position freely.
7. Optional. Drag the variable to the preferred position in the text block structure to modify the order of variables.
8. Optional. In **Word**, modify the position of the variable if necessary.
   Note Modifying the variable position in Word has no effect on the variable position in the text block editor.
9. Optional. In **Content Composer Studio**, in the **Object Inspector** pane, edit the properties of the variable.

- In Microsoft Word

To insert an image variable, complete the following steps.

1. Open the text block.
2. On the **Editor Tools** tab, click the **Show Word Document** button.
3. In Word, on the **Content composer** tab, click **Insert Text Variable** > **Image Variable**.
4. In the **Insert Image Variable** dialog box, in the **Variable Name** box, type a name for the variable.
5. Optional. In the **Default Value** box, type the path and file name of the image or a link to the image. This value is used to populate the variable if no data field is defined.
6. Click **OK** to confirm. The variable is inserted in a text box that you can position freely.
7. Optional. In **Content ComposerStudio**, drag the variable to the preferred position in the text block structure to modify the order of variables.
8. Optional. In **Content ComposerStudio**, in the **Object Inspector** pane, edit the properties of the variable.

Hyperlink in an image variable

If an image variable should contain a hyperlink, the variable content should be formatted as follows:

C:\Logo.jpg|http://www.hyland.com

The variable content consists of two parts: the name and path of the image file and the URL. The two parts are separated by a pipe sign ("|"). Defining **http://** is mandatory.

The link can be opened in the generated document by the key combination **CTRL + left mouse button** on the image.

Note The way characters in variable content display in generated documents is defined by the scope of the respective font being used. For this reason certain mathematical special characters cannot be displayed if they are not supported by the font used.

Applying a URL as Source of an Image Variable

To apply a URL as a source of an image variable, complete the following step.

- Apply the URL as content of the image variable.

  Example

  https://www.hyland.com/assets/hyland-careers/images/logo.png

To apply a URL as the source of an image variable and assign a hyperlink to the image, complete the following step.

- Apply the URL and the hyperlink, separated by a pipe sign ("|"), as content of the image variable.

  Examples

  ```
  https://www.hyland.com/assets/hyland-
  careers/images/logo.png|https://www.hyland.com
  ```

  ```
  https://www.hyland.com/assets/hyland-
  careers/images/logo.png|https://www.hyland.com
  ```

Notes

- Defining **http://** or **https://** is mandatory.
- In the generated document, the user can open the link by performing the key combination **CTRL + left mouse button** on the image.

Note If the reference image file does not exist, the image file saved in **%ComposerDir%/PDC_ ImageVariable_FileNotFound.png** includes as a substitute into the DOCX file.

Base64 coded images

Here, instead of specifying the path to the image, you have to specify the actual Base64 coded string as the variable value.

To make sure the value is recognized as Base64, the string is embedded in an identification string as the following example shows.

ModBASE64Image*[Base64-Bildcode]*ModBASE64Image

The Base64 image code cannot contain line breaks.

Example

<?xml version="1.0" encoding="UTF-8"?>

<PARTNER xsi:noNamespaceSchemaLocation="Partner_Immobilie.xsd"

<ADRESSEN ADR_Par_Num="0816" ADR_Lfd_Num="1" ADR_Gueltig="J" ADR_Typ="D" ADR_ Bezeichnung="Hauptadresse" ADR_Strasse="Hauptstraße 1a" ADR_Postfach="" ADR_PLZ="70000" ADR_Ort="Stuttgart" />

<ANREDE ANR_Anredecode="2" ANR_Adressanrede="Frau" ANR_Briefanrede=" Frau" ANR_ Persoenlich="Frau" ANR_Adressanrede_E="Mrs." SAL_LetterSal_D="Frau" Sal_Personal_D="Frau" ANR_ Adressanrede_F="Mme" ANR_Briefanrede_F="Madame" ANR_Persoenlich_F="Madame" />

<TITEL TIT_Titlecode="5" TIT_Description="Dr."

<RE_Code="220-D-986" IM_LivingSpace="200" IM_Location="Cannes" IM_Price="1.000.000" IM_ Title="Villa Konstantin" IM_PictureFile="c:\PerceptiveSoftware\PDCSuite\data\XMLData\villa_ konstantin.jpg" />

<RE_Code="110-A-679" IM_LivingSpace="180" IM_Location="Nizza" IM_Price="800.000" IM_Title="Villa Belle Vue" IM_PictureFile="c:\PerceptiveSoftware\modusSuite\data\XMLData\villa_ bellevue.jpg"></Immobilien>

<RE_Code="132-D-684" IM_LivingSpace="230" IM_Location="Miami" IM_Price="500.000" IM_Title="Villa Miami Beach" IM_
PictureFile="ModBASE64Image*/9j/4AAQSkZJRgABAQAAAQABAAD//gA7Q1JFQVRPUjogZ2QtanBlZyB
2MS4wICh1c2luZyBJSkcgSlBFRyB2NjIpLCBxdWFsaXR5ID0gNzcK/9sAQwAHBQYGBgUHBgYGCAgH
CQsSDAsKCgsXEBENEhsXHBwaFxoZHSEqJB0fKCAZGiUyJSgsLS8wLx0jNDg0LjcqLi8u/9sAQwEICAg
LCgsWDAwWLh4aHi4uLi4uLi4uLi4uLi4uLi4uLi4uLi4uLi4uLi4uLi4uLi4uLi4uLi4uLi4u/8IAEQgAlQDIA
wEiAAIRAQMRAf/EABsAAQADAAMBAAAAAAAAAAAAAAEBQYBAgMH/8QAGgEBAAMBAQEAAAA
AAAAAAAAAAAECBAMFBv/aAAwDAQACEAMQAAABpx6WUAAAAAAAAAAAA7X1Zz6xrpgJAAAFnX
Q6iQAABYeXC8Ry7UvtjGnYu+AqdXcdufzxusd0rGbmZW3zpc6S0YLv9AyNX0X5V9XqOHT503OW08699
C86zgUvVWrip+zp/P0e0Wx5+b9HOtM+kwV1zn7vTxxttPs5iZTXdLyvYYLe4HtS81GW1XO0Dyq4to3vzj6
P81h9LoNBnqXjS4cu1eLinuKziLWn0XmafNxz4uutTH1/k09nkmml1qfniG/zFONzN+com40eETH0fOZtE/Qf
b5whr6CDNrbWxKaXxtxaK00VHXe1otKH19TQMK618ho5gAAAAD0qtbCP28D1JXh17681nC7drxD85kaH
WbA7RNM5ehlijrAAAAAC8rdPg08end52tzxyeB2mLVEkTXuJZddNfHCj0sgAAADngW1jmLfDp0A83WjyO
LVjyuO8T6cS0xF7SBTID0MmfG7gAAAAAtaq14dNEPG9AB269ieL0AzDh62LPjvzAAAAAWhn66Tk8feBx6
CJ/UvTxjkWzo9jF//EACsQAAICAQMCBQQCAwAAAAAAAAIDAQQFABESMDMQEyAhMhQVIjEjNCRD
YP/aAAgBAQABBQL/AIuIkpDFWSi3VOrPRCIaOJjaegFUzg0MDX61hhgrdyxFZN639VPQD5x22dz11V8z2
5aNTJmwEwFKqlQvSt4WKaBufbKmm4pExKWQ9GKUMTjaerVXjdVi64iWMqTFqqVV4/D7dV1YxaZCnW
Oy0MZVGGYusUPQaXVsWECePpQABwiP1oULfrEOab8qw1VYbYY3jltVPN8jjH3rIsJVSLNiNYoydduLY1
GPqvrTmI/hj4Hbs+ePuOKiIjMvassQ5jUXBicjZKV16rWsJf68Ldjy04X+zmv6eDAeF65NWazJanfbNZJZNq
RWsTrFCSbtuXCj6zJatOuHEduazyePsOIKCHMpYc4geKbhRGRsjLK9Kq8rDFkvW+/hfqb1cY9aHZK4h9b
HW/piLIU9gyladXbEHcRIUkM5GpGrdqDuIyqCH7jU1krarEryVXb7jU1bygyFOyVVhZGuxKryK9eOFo6+TDj
ORq6fdm0abNEV2bH+SGTrSHWiJKYrM0ioZaq1eRAlcxdCAepfECgVggA1NTzNGHGetT22IoGBaK1bzoA
TM24mTj20f46Hbc3gAN/fVD5gMskNwmNonffSpkFN5uOFOjUpdqFsjRxGnR79WqmNeSvUCMeM94vYv
RIB/PqIX5jNuLPQfz/2JL8BKC8bIw1PUq2ASKneczxP3JPuIxucqjfyz1xbri3VqyyvPVoT+fjMb+Ad30ZX59W
h3fSHd9GV+fVod30h3fRIfn1aHd9IR/Jt4c9S/bWRPmz/xAAlEQACAgEDAwQDAAAAAAAAAAAAAQIRAx
IgIRATMgQwMUFCUXH/2gAIAQMBAT8B91K/b7cquukU0jS2OtomlolGxxaO2xRbMOG5cjVmaK1C8Y5tcjVm
RLfTT22Tp+FjbHGeG5cjqvmaNeSvUCMeM94vYvRIB/PqIX5jNuLPQfz/2JL8BKC8bIw1PUq2ASKneczxP3JPu
qkaGKDojBtmW1Lf6eH5dV1nC3vw5XenpJWhcIssyTalvw+a3ZfLfi8luy+R/8QAKBEAAgECBgIBBAMAAA
AAAAAAAAECAxEhMQQSIjJBUWFxgZGh/9oADAMBAAIRAxEAPwBAT8B91K/b7cquukU0jS2OtomlolGxxaO2xRbMOG5cjVmaK1C8Y5tcjVmRLfTT22Tp+FjbHGeG5cjqvmaNeSvUCMeM94vYvRIB/PqIX5jNuLPQfz/2JL8BKC8bIw1PUq2ASKneczxP3JPuIxucqjfyz1xbri3VqyyvPVoT+fjMb+Ad30ZX59W
```
```

AAAAAAAQIAAxEEEhMgITEQMwUiIzAyQlFx/9oACAECAQE/AfyIgO4DfrbcHaMTSLZb8+HYM95qqoi1A
011hqqBBVDSm4XuCopF5rrGqAC8xuNCU7DsxaxUz4+u+jzGH3Iz/VlAi9mUbZeZU/fiBTe8pW5vB0YSMg
n9T5SgzNmXkQIephlVaQENMFs0akGN4tIA3mgsNJSLRaIBvNBY70qQsxg0yeDP9hCgRUpqYKSnrexsI+
H131GNooRJqLGqAmNUFuJRtkFt+Lq/xHk+aWJyLbfiaK5S3hTYw8nzRooVud+J9Z3Yb1jfifWd2E9U/8QA
NxAAAgECAgcGAwYHAAAAAAAAAQIAAxESISIwMTJBUXEQEyAjM2GBkaEEFEJSYnJAQ2BjgpKi/9o
ACAEBAAY/Av6LAAuTLkqsXGQcXLVXFI294Ry1N7iZjsufwid5a8Q4MOHUr1nwj/uOoz2CcrcRMziExFbR
aiLZivOYKguJQooui18Wc3D855ZKmdxbTvaebdmnpfWfd6PGeZdzMkw9DEUm6k5GDpCSlyfeE0dFphGQ
G8ZmpbqZoAoYaLDPh7wNXuW5Qk07Ae8wLlftIqjREKPUJULsmKmxU32iBg7tU4T+Z/tE771OMJ/TeOyG
zT13+cL1DibDClJsJMbvKgZTwII/3BB0hPfNvQGVz+uU0RyoPKN3jYrHbPsplR12hYe8qs3teX7TTQ6T7ek
b9s/yEqVPxXtFHdF7xahTATwnVI6oLmei/wAoUqDCxWE0Bd56X/Mpj7QmFcYtPhCBSbNoBK1vzym6qSo
yNox4Ez7KJURdpWKDTZRxM0t383atRR5ijOF6hsLTBTbSvGDi6N9Jctf4Q3uvKd/RbZsnmgo09SfeKN8uc
8y6NPVILu76LXMALEdRPUhSgMz+IzEM14iOBcMRsMVBdnjV69bAw2CYa4IP5p6l/hBSpgrTGZ94Ffe4wv
QY4OEGM2PEa+wE4S94xa0ps6g7VygAXLlDcXMcgZ7JprfK8x0tnIy2vPObc4gXNxxl2F+spuFsrcjxi3OkuX
WftgTjLE7xzhRN7ZLa4QKuVhmYELaDb0N9vtNxj8IyMDc7sS4UPPTyvc5zOm15mj/KC+0+2yX12mOF5u
zJR2r0mIbQLzLwA60CKv6fCkHSWtmuXgI4606F2mK1svARiZbC+XGY23jBLhiDPVaZVfpN8fKAZNf21xH
gz7F8KdNceniXp4U6a49PEvhXp/CL27Ju/Xs//8QAKhABAAICAgAEBQUBAQAAAAAAAQARITFBUTBhc
ZEQgaHB0SBAseHw8WD/2gAIAQEAAT8h/wDFuWRQEAeQmDOeV6P++DvBKQHbCM7tU+DUQc0wS
4IqkqUktVIge8AisbPO7r8eD9D/AJmj/dT/AHO/Ap0c0qMZ4EPtZI5JnUXMNbZl7lt4ai/CwLT/AL2Kun5sm4O
vCq+UXRFFYeYoICtBLYY8xbojMRdpKAF+/BYQX2cVVQUU9F4Z/CrEOjvaRj5gGWUOqHoUWloJWkI2K
GgZO9EFAx8vhmEC91MCDCcEa0TLIF4IlykELPePzLKtUgapr3qmuMA9Zi1lsJPCtsD7ynqEIhgLhhPLPvT
Fo4iUBeNxWHJcMBIcIUCtquWKKsUeRmb7MkNQHN9oG3Z+Owx+TT46nKi+UQRd6Oon7k+JWh6D2mi
std5iFF0ZY4gw5w8PUH5wD5Ew5miMj45TG4LDgqebQyx6nQlbFdnmPJzDutwQiRc2Klmi+iA0R9l/V1F36z
3yxHSV1LdGlqoLptVtRQD5IzKI1Y2QqrAtXJCBzB2QS8/QgFRQpEBV5HJEOHtBhvcEWwwqCBdfaMuor
RXpKI2Y7yOy1KJW0TIGLhRtOolMC4oWMCWLyEZGJLuk0EuBNUFE/KK6KaLp8e+JeiKLaeUswDNZ5iX
U3Reo7ntNNOJXKxYL2ATo/LDYK1jqiGeV6ka4345OCXFjDAwJtjoP3lZiebfaFmwMxXeD2TpfDZMN7Bfz4
mQqQZ6vmBUlZvpLLwYPEfBXPjc3uIa5AekaAvwvLFfR8q6JxHzWXpbY84sE9N7iZdtAMxmVLb3Egr1ykJ
hguRVnohlWnnxmnK0ecB19U0wlBo+GFnLjR4EgiXo6+FzcpGUJ4rAEoAfpC69l+0YGtcrh5cRRCxOE+DiD
osFmPF2pMXep16R+i1xKwIKNzyvUp3kz1QQnV9D8T+9myrKut5YpBeaeM3UC/wBACgJ5/D6R/c5n1z9z
GfTP7NtvpKlSpUqZq+GeqJUaf2mZJBxVfif/2gAMAwEAAgADAAAAEPPPPPPPPPPPPPPvfPPPMfPPP
Pfrn+/cP59tc41TJT59UlP9GQ3uAznTfTrjyF8S3jPPPPPPPwthLij/ADzzzzzE8odtjLzzzzyzj6/r64/zzzzyn77
7zwHzzzzzyKACN/wL//EACARAQACAgMAAgMAAAAAAAAAAAAEAESExECBBMIFRkfD/2gAIAQMBAT
8Q+VNCIjT1RN9U8iuKSbl5ubiEl5rGrUUrhQojm6EGLWNtN4lK+k8k33HEDL4rgBY+Q/CapN/uVq4HsE
QAqCUwkJsgFPArYaIICljq9vYxnRlhMpwnc2qYgtPf72W7qC6IdWZYYEgad7FX1zuxK4Wyu6T44SgagwS
3BYX8Qdc7PizqEJvn//xAAkEQEAAgIBBAEFAQAAAAAAAAABABEhMXEQIEFRMGGBseHwkf/aAAgBA
gEBPxD5D8A4sqxBnMpwnc2qYgtPf72W7qC6IdWZYYEgad7FX1zuxK4Wyu6T44SgagwS
3BYX8Qdc7PizqEJvn//xAAkEQEAAgIBBAEFAQAAAAAAAAABABEhMXEQIEFRMGGBseHwkf/aAAgBA
gEBPxD5D8A4sqxBnMpwnc2qYgtPf72W7qC6IdWZYYEgad7FX1zuxK4Wyu6T44SgagwS
3BYX8Qdc7PizqEJvn/9oACAEBAAE/EP8A4A5sqxBnMpwnc2qYgtPf72W7qC6IdWZYYEgad7FX1zuxK4W
y

snY95XcEqdXf+M3uczyALPtnYuNx++I/pGXnxjt5EK2twpuYhDmnP/eaB9FFGu70XConM/tmzDI0i91xWBT
T6MAEXQ6xVME8vJNI31p3jjUUNlBt48mAlBVrwLD9nDRDdZVEMWN6GIHjvuZLIZOHmFOjvyecKvvFzh
BODyc02ELaX5BXF9lKVumMJ81NC3JsLeiLs7879GQ5pRWn5w3UmWe9DjNqxViAcmCspAKvcefjNZfFb
kjb5Cpbq8YAmzWbe48/GDW/pYMEJIieN4KEC0dPC5ti9PEI1PUD1HVyd+6Gep6OE3QyNScl74TSRnBd
31MFi3YZsj7uO2l3sPw4t8gS4hkVmGOPB168ZbZtsc3XD2mAmy8NCC6fNwsa+Ohs8n1zDnwFzbx7nCK
EickFQ/HPxgpbIChPAI8/bBBwxI22g88/GBWVI6GhTEYOCPVdH5yCDUztXa/E++JrdXL5Wnlf2xWjKcAZa
d+euIgoZrv9dTZbynTHkISjl6YVFLJVYieUXWc210Rh3Aw/GC3S0gl6HIvwYJ+5OStPwlyYSmScxp+ZjNnc
ZaIpPEJiIsMSMGj1XJsSx7K7bkyYAbcxKe7z9/rQgKchyUBkJTQ1zcFYCBzOi8zv4uD1SgeABOmUN5po
D5MRgDpa6XwBMJDhG2BXAEpQdBwRfWPUJbDW/OCU2nKvSXNIMXAjXI7FfeEJuoBL/ePrQyAEGQqf
xhPU7uBZ6Aw4Afb/AAzohj2P/ZgerIHWLrATUqu5ip3yTSz3mhsH7ZKQV0aXj+Pq0iFLnRUQXsP/AL+lNN
Rv4XDt6C+/OKfSQ57h9s6kMKHIYgVYY+drqmch7+ozUvG8riOVgG+Cf2GEOxqW3rfn9EEQDEjTV9nzkt
S3ogdQ6HX74bmpu7YpMRFZTIEHCDG4SnwT+0xCBlzu/JcRM8GgXw7+sknMXtn+v0eCECmAAAAHAd
Mr+3x/ww+Z/VGy3+F/StU4+s+U/Vyf7z9Lp9vrFGPHCuue2e2e2FauV6RonjP7TJLblCbTHAVO/wD0yhEk
At6M/9k=*ModBASE64Image"></Immobilien>

</PARTNER>

The following constellations regarding image files are not supported.

- Specifying a Base64 coded string in the dialog for manual variables, if the image variable is defined as a manual variable.
- Specifying a Base64 coded string as a default value for an image variable

Dynamic Charts

About Dynamic Charts

Dynamic charts draw charts at runtime depending on the given data and insert them in the document.

To insert a chart in a document, predefine the chart in the text block and then use a script to assign the data points.

Predefine a Dynamic Chart

To predefine a dynamic chart, complete the following steps.

1. Open the text block.
2. In the **Editor** window, below the bottom-right pane, click **Text Block Variables**.
3. In the **Text Block Variables** pane, in the **Variable Name** box, type a name for the chart container.
4. Select the **as Image Variable** check box.
5. Click the **Insert** button. In **Microsoft Word**, the chart container inserts as a text box that you can position freely.
6. Optional. In **Word**, resize the chart container and modify the position.
7. In **Content Composer Studio**, in the **Object Inspector** pane, in the **General** section, click the **ChartDefinition** property and click the ellipsis button next to the property.

8. To update the **ChartDefinition** properties, complete the following substeps.
    1. On the **General** tab, set the properties.
    2. On the **Area** tab, add one or more areas and set the properties.
    3. On the **Series** tab, add one or more series and set the properties.
       Note The **Chart Definition** editor uses random values to depict the appearance of the chart.
    4. On the **Legends** tab, set the properties
9. In the **Object Inspector** pane, assign a script to the **BeforeInsert** event.
10. In this script, assign the data points to the chart.
    For more information, refer to About the context TextVariable and Sample Script: Generate a Chart.

Chart properties

General tab

Property	Meaning
Title	Chart title, font, and color.
Position	Title position.
Alignment	Title alignment.
Shadow	Shadow offset and color.
Gradient	Gradient style and color.

Gradient Style	Meaning
Center	Gradient from the center outwards.
DiagonalLeft	Gradient diagonally from left to right.
DiagonalRight	Gradient diagonally from right to left.
HorizontalCenter	Gradient horizontally from the center outwards.
LeftRight	Gradient from left to right.

None	No gradient.
TopBottom	Gradient from top to bottom.
VerticalCenter	Gradient vertically from the center outwards.

Border — Border line style, width, and color.

Border Line Style	Meaning
Dash	Dashed line.
DashDot	Line with a repeating dash-dot pattern.
DashDotDot	Line a repeating dash-dot-dot pattern.
Dot	Line with a repeating dot pattern.
NotSet	No line.
Solid	Solid line.

Dpi — DPI of the chart.

Area tab

Property	Meaning
Chart Area	Chart area.
xAxis	X-axis title, font, and color.
Alignment	X-axis alignment.
Format	X-axis format, such as 0.00 $.
yAxis	Y-axis title, font, and color.
Alignment	Y-axis alignment.

Format	Y-axis format, such as 0.00 $.
Display 3D	Displays the chart in 3D.
Rotate X	Rotates the chart at the X-axis.
Rotate Y	Rotates the chart at the Y-axis.
Set xAxis Min / Max	Use the settings for minimum and maximum value and interval for the X-axis defined below. **Notes** • If you select this option, data points outside the range of the minimum and maximum values do not display in the chart. • If you do not select this option, the width of the X-axis scales automatically.
xAxis min value	X-axis minimum value.
xAxis max value	X-axis maximum value.
Interval	Value interval for the X-axis.

Series tab

Property	Meaning
Series	Series.
Type	Series type and color. **Note** Not all series types are combinable.

Series Type	Meaning
Area	Area chart type.
Bar	Bar chart type.

BoxPlot	BoxPlot chart type.
Bubble	Bubble chart type.
Candlestick	Candlestick chart type.
Column	Column chart type.
Doughnut	Doughnut chart type.
ErrorBar	ErrorBar chart type.
FastLine	FastLine chart type.
FastPoint	FastPoint chart type.
Funnel	Funnel chart type.
Kagi	Kagi chart type.
Line	Line chart type.
Pie	Pie chart type.
Point	Point chart type.
PointAndFigure	PointAndFigure chart type.
Polar	Polar chart type.
Pyramid	Pyramid chart type.
Radar	Radar chart type.
Range	Range chart type.
RangeBar	RangeBar chart type.
RangeColumn	RangeColumn chart type.
Renko	Renko chart type.

Spline	Spline chart type.
SplineArea	SplineArea chart type.
SplineRange	SplineRange chart type.
StackedArea	StackedArea chart type.
StackedArea100	StackedArea100 chart type.
StackedBar	StackedBar chart type.
StackedBar100	StackedBar100 chart type.
StackedColumn	StackedColumn chart type.
StackedColumn100	StackedColumn100 chart type.
StepLine	StepLine chart type.
Stock	Stock chart type.
ThreeLineBreak	ThreeLineBreak chart type.

Label — Series label, font, and color.

Is visible in legend — Series label displays in the legend.

Width — Series width.

Drawing Styles — Series drawing style.

Drawing Style	Meaning
Cylinder	Data points show as cylinders.
Emboss	Data points show with an embossed effect.

LightToDark	Data points show with a light-to-dark effect.
Wedge	Data points show with a wedge effect.
Default	Data points show flat.

Legends tab

Property	Meaning
Legend	Legend title.
Position	Legend position.
Alignment	Legend alignment.
Use Legend	Show the legend.
In Chart Area	Show the legend in the chart area.

Assigning a Data Model

Assign a selection to the text block to populate text block variables with values from a database table or XML file.

To assign a data model to a text block, complete the following steps.

1. In the **Editor** window, below the bottom-right pane, click **Data Model**.
2. From the **Navigator**, drag the required selection to the left-hand pane of the text block window. The selection displays in the upper-right pane of the text block window.
   Note If the text block already has a selection defined, this operation overwrites the link.
3. In the upper-right pane of the text block window, click one of the data objects in the selection. The variables of the data object display in the bottom-right pane.
4. Drag the variables of the data object to the appropriate text block variables.
5. Optional. To remove an existing data link, right-click the variable or the text block and click **Remove Data Link**.
6. Optional. To delete the value of the **AttributeName** property, right-click the variable and click **Reset Attribute Name**.

Inserting a Child Text Block

To insert a child text block, complete the following steps.

1. Open the text block the child text block is to be inserted in.
2. From the **Navigator**, drag the preferred text block to the **Editor** window.
   Note If required, you can insert the same text block more than once.
3. Optional. In the **Object Inspector** pane, edit the properties of the child text block.
4. Optional. Check the position of the child text block in **MicrosoftWord** and modify the position if necessary.
   Note Right-click the text block, then click **Select in Word** to highlight the reference to the selected text block in the Word document.
5. Optional. To modify the status of the text block, in the **Editor** window, right-click the text block and select one of the following menu items.
   - **Required**. Sets the value of the **TextInsertionRule** property to **required**.
   - **Activate**. Sets the value of the **TextInsertionRule** property to **activated**.
   - **Deactivate**. Sets the value of the **TextInsertionRule** property to **deactivated**.
   - **Disabled**. Sets the value of the **TextInsertionRule** property to **disabled**.

   For more information, refer to General text block properties.
6. Optional. In the **Editor** window, right-click the text block and click **Inherit Attributes**.
   Note If this function is activated, the text block in the document inherits the properties of the original text block.
   For more information, refer to Activate inheritance.

Inserting a Picture

Make sure the picture file has the preferred image size before you insert it.

To insert a picture, complete the following steps.

1. On the **Editor Tools** tab, click the **Show Word Document** button.
2. In **MicrosoftWord**, on the **Insert** tab, under **Illustrations**, click the **Picture** button.
3. In the **Insert Picture** dialog box, select a picture file.
4. Click the **Insert** down arrow, then click **Link to File**.

Applying Text Block and Variable Properties

If you have altered properties of a text block (or of one of its variables) already in use by a document or another text block and you now want to transfer these changes to all or some objects that use the text block, complete the following steps.

1. Open the text block from where you want to roll out property changes to text block references.
2. In the **Editor** window, select the root text block or a text block variable.
   Note The Properties Wizard transfers only the properties of the selected text block or variable.
3. On the **Editor Tools** tab, click the **Apply Properties** button.
4. In the Wizard, on the **Objects Selection** page, complete the following substeps.
   1. Select the text blocks and document templates whose child objects you want to update with the altered properties. To select or deselect all objects, right-click a check box and choose the appropriate option from the context menu.
      Notes The list does not consider child objects with active inheritance property, it does not display objects with property **AttributeInheritance=True**.
      Locked objects entries are disabled. The locking user is indicated in the **Note** column. To verify an object has been unlocked, on the left top of the object list, click the **Refresh** button.
   2. Optional. To determine the creation of a new work version, behind an object, select the **New Work Version** check box. In the **New Version Label** field, enter an optional version label.
      Note These actions require an installed versioning license.
   3. Click the **Next** button.
5. In the Wizard, on the **Properties Selection** page, complete the following substeps.
   1. Select the properties to roll out. To select or deselect all properties, right-click a check box and choose the appropriate option from the context menu.
   2. Click the **Next** button.
6. In the Wizard, on the **Summary** page, complete the following substeps.
   1. Inspect the selected objects and properties.
   2. Optional. To adjust your selections, use the **Back** button and the **Next** button to navigate to the previous pages and back to the **Summary** page.
   3. Click the **Finish** button.
7. In the Wizard, on the **Result** page, inspect the indicated result.
8. To close the wizard, click the **Close** button.

Multiple Text Blocks

About multiple text blocks

Use multiple text blocks to model 1:n relationships in data structures in text blocks.

This enables you, for example, to insert several different bank account details in a document.

Notes

- You cannot use a variable text block as a multiple text block.
  Even with the property **Multiple** set to `True`, the text block only appears once in the document structure.
- XPATH is not suitable for use with multiple text blocks.
  When executing an XPATH expression, the following behavior displays.
  If multiple nodes exist in the data XML that match the specified XPATH expression - as is usually the case with a multiple text block - the value of the first node found is always used as the value of the variable, whether it is the first or n-th iteration.
  Solution
  For variables of multiple text blocks, set the DataSource to DataModel and specify the corresponding data link and attribute name.

Defining a Multiple Text Block

To define a multiple text block, complete the following steps.

1. Open the text block.
2. In the **Editor** window, below the bottom-right pane, click **Data Model**.
3. From the Navigator, drag the required selection to the left-hand pane of the text block window. The selection displays in the upper-right pane of the text block window.
4. From the upper-right pane of the text block window, drag the preferred 1:n node from the data structure onto the text block. The **Multiple** property changes to **True**.

When generating a document, the text block is inserted for each account found in the data structure.

Multiple insertions of a text block using a script

You can insert a text block as often as you want using a script.

To do this, complete the following steps.

1. Open the text block.
2. In the **Object Inspector** pane, complete the following steps.
    1. Set the **Multiple** property to `True`.
    2. Set the **DataSource** property to **None**.
    3. Assign a script to the **BeforeInsert** or **AfterInsert** event.

3. In this script, set the **Blockpool.Break** variable to `False`.
4. To signal the end of insertion iterations, set the **Blockpool.Break** variable to `True`.

Note If you set the **Blockpool.Break** variable to `False`, be sure to reset the variable to `True` at a later stage. If not, the script runs in an infinite loop.

The following example shows when and how you insert a text block multiple times using a script.

Example

Insert a text block a predefined number of times

1. Open the document.
2. In the **Object Inspector** pane, assign a script to the **Load** event.
3. In this script, set a SYSTEMPOOL variable.
   Example
   ```
   System.Counter = 0
   ```
4. Open the text block.
5. In the **Object Inspector** pane, assign a script to the **BeforeInsert** event.
6. Modify the script according to the following example.
   Example
   ```
   Blockpool.Break = False
   if (System.Counter < 10)
      System.Counter = System.Counter + 1
    else
      Blockpool.Break = True
    end-if
   ```

The Content Control EmptyLine

About the Content Control EmptyLine

The content control **EmptyLine** enables you to define an area in a text block or document that will be deleted on document generation if the variables contained within the section do not have values after data retrieval.

You can apply this practice to a single line, to several lines, or just to parts of a line.

If the content control contains an active text block, the whole content of the content control is visible.

If the text block contained in the content control is deactivated and none of the variables have values, the whole content control is removed from the document on generation.

Activating the Developer Tools in Microsoft Word

In Word, check if the **Developer** tab is available. If not, perform the following steps.

- **In Microsoft Word 2007**

1. Click the **Microsoft Office Button** > **Word Options.**
2. In the **Word Options** dialog box, click **Popular**.
3. Select **Show Developer tab in the ribbon**.

- **In Microsoft Word 2010 and 2013**

1. On the **File** tab, click the **Options** button.
2. In the **Word Options** dialog box, click **Customize Ribbon**.
3. In the **Main Tabs** list, select the **Developer** check box.

Inserting an EmptyLine Content Control

To insert an EmptyLine content control, complete the following steps.

1. In **MicrosoftWord**, position the cursor to the place where you want to insert the content control.
2. On the **Developer Tools** tab, in the **Control Elements** group**,** click the **Rich Text** button.
3. On the **Developer Tools** tab**,** in the **Control Elements**group**,** click the **Properties** button.
4. In the **Content Control Properties** dialog box, in the **Title** box, type a title.
5. In the **Tag** box, type **mc:EmptyLine**, then click **OK**.
6. Drag one or more variables into the content control.

Alternative method

1. Select a text segment that you want to enclose in the content control (including the final paragraph mark)
2. Create the content control as described above.

Variable Pools

Variable pools can be used, for example, to store the results of calculations.

They can also be used to pass variables on to connected systems such as Odin.

- The pool **MODUSUSER** provides user-specific information.
- The contextpool **BLOCKPOOL** provides access to text block variables.
- The **DOCUMENTPOOL** provides access to document variables.
- The **SYSTEMPOOL** is available to bundles.

Typing **Poolname.NewVariablename** in a script creates a new variable.

Variable Text Blocks

About Variable Text Blocks

Using scripts and conditions you can control whether a text block is inserted into a document.

In some cases, the text block to be inserted can be directly linked to a value in the data.

Example The text block to be inserted depends on the country of the recipient.

So that you do not have to predefine all text blocks that might possibly be required in the document structure and then activate or deactivate the required text block by way of conditions, it is possible to define a placeholder text block. This placeholder text block is replaced during document generation by the required text block depending on the data content.

Notes

- To detect any variable text blocks that were not found, check the MissedObjects bundle property in a script. The content of MissedObjects is cleared before every document generation.
- In general, the scripts assigned to the BeforeInsert and AfterInsert events of the placeholder text block are executed.
  If no scripts are assigned to the placeholder text block for these events, then the scripts of the inserted text block are executed.
- You can insert manual variables contained in the placeholder object in the input dialog box for manual entry and use these variables in scripts of the placeholder objects.
  However, the variables are not inserted into the generated document.
- You should not use manual variables in the text blocks to be inserted, because the input dialog box for manual entry does not display for variable objects.
  However, the variables are inserted into the generated document.

Defining a Variable Text Block

To define a text block object as a variable text block, complete the following steps.

1. Open the text block.
2. In the **Object Inspector** pane, right-click the **VariableObjectDefinition** property and click **Create**.
3. In the **Object Inspector** pane, in the **Object Definition** section, click the plus to the left of **VariableObjectDefinition**.
   As a result, the following additional properties display below the object definition.

- AttributeName
- DataLink
- DataSource
- LookupSource

- LookupKey
- XPathExpression

Note These properties define the data field that contains the name of the text block to be inserted. For more information, refer to Variable Properties.

4. Perform one of the following options.
   - In the **Object Inspector** pane, in the **Object Definition** section, type a value for the properties **AttributeName** and **DataLink.**
   - Perform the following substeps.
   1. In the **Object Inspector** pane, in the **Object Definition** section, click the **XPathExpression** property, then click the ellipsis button next to the property**.**
   2. In the **Open** dialog box, select an XML file.
   3. In the **XPath Evaluator** dialog box, in the **Sample XML tree** box, right-click the variable that you want to assign to the text block variable, then click **Copy XPath**.
   4. In the **XPath Expression** box**,** right-click and click **Paste**.
   5. Click the **Evaluate** button.
   6. In the **XPath Result** box, check the result.
   7. Click the **Apply** button.
5. Select a value for the property **DataSource**.
6. Optional. To define an alias table to look up the value retrieved, use the property **LookupSource**. For more information, refer to Create an alias table.

Attribute Inheritance

About Attribute Inheritance

Modifications to a text block have no effect on the properties of the same text block when it is used as a child text block in a document or a parent block. A text block may possess different properties for each different usage. Only the structure of the text block is reused (that means text block variables and nested text blocks). Text blocks that inherit their properties are the exception to this rule.

The following rules apply if inheritance is active for a child text block.

- The child text block symbol shows in red.
- You can no longer modify the properties of the child text block.
- The child text block always uses the properties of the original text block.

Activating Inheritance

To activate inheritance, complete the following steps.

1. Open the document or the text block that contains the child text block.
2. In the **Editor** window, right-click the child text block that should inherit its properties and click **Inherit Attributes**.

Property Transfer

About Property Transfer

- You can transfer properties both between text blocks and between variables. Possible target objects are any objects of the same type at all levels.
- You can transfer between objects inside the current object window as well as between any other object windows open. The objects must be of the same type.

Transfer Properties

To transfer properties between objects, perform the following steps.

1. Press `CTRL` and drag the object whose properties you want to transfer onto the target object.
   Note To arrange two opened text blocks side by side in the Editor Window, drag a text block tab into the center of the Editor Window, onto an arrow element of the Docking Manager.
2. Complete the following substeps if you selected text blocks with the same object IDs as source and target object. Only in this case the **Selection** dialog box appears.
   1. In the **Selection** dialog box, select one of the options to specify the transferred items.
      - Only text block properties
      - Only properties of the text block variables
      - Properties of the text block and its variables
        This option combines the two preceding options.
      - Properties of the text block and its child objects
        This option equals the preceding option and extends it to all child objects of the text block.
   2. Click **Apply**.

Reports

Generating the "Structure of a text block" Report

Content Composer allows you to document the attributes and variables of a text block and its child text blocks as a DOCX file, using the DOCX file specified in the configuration file **Composer.Studio.exe.config** as a template.

To generate a text block structure report, complete the following steps.

1. Open the text block.

2. On the **Editor Tools** tab, under **Reports**, click the **Generate report "Structure of document/text block"** button.
3. In the **Report Creation** dialog box, select a file or type the name of the report file.
4. To generate the report, click the **Save** button.
5. In the following dialog box, decide whether to open the DOCX file immediately.

Generating the "Used data connections" Report

Content Composer Studio allows you to save used data links of a text block as a CSV file.

To document the used data links of a text block, complete the following steps.

1. Open the text block.
2. On the **Editor Tools** tab, under **Reports**, click the **Generate report "Used Data Links"** button.
3. In the **Save Used Data Links Report** dialog box, select a file or type the name of the report file.
4. To generate the report, click the **Save** button.
5. In the following dialog box, decide whether to open the CSV file immediately.

Note When the report is saved as a CSV file, a special character separates the cells of the CSV file from each other. To determine this character, Content Composer uses the list separator specified in the Regional Settings section of the Windows Control Panel. We recommend not to change this setting.

Documents

About Documents

There are two document types available to the user.

Dynamic documents

As a rule, dynamic documents are comprised of one or more text blocks.

Static documents

Static documents are documents that, when created as an object, are imported as finished documents and cannot be edited. Static documents refer to Word, PDF, TIFF, and XPS documents.

Dynamic Documents

Creating a Dynamic Document

To create a dynamic document, complete the following steps.

1. In the bottom pane of the **Navigator**, right-click and click **New** > **Document**.
2. In the **Create New Document** dialog box, type a name for the document and click **OK**.

3. In the **Select Document Type** dialog box, select **Dynamic Word Document** and click **OK**.
4. Optional. To add text to the document, perform the following substeps.

1. On the **Editor Tools** tab, click the **Show Word Document** button.
2. In **MicrosoftWord**, type the content of the document.
3. In **Content Composer Studio**, save your modifications.

5. Optional. In the **Object Inspector** pane, edit the properties of the document.

Dynamic Document Editor Tools

Icon	Name	Description
	Refresh	Refreshes the document if child text blocks or selections have been modified while the document is still open in editing mode. If modifications have been made to the document the function remains deactivated, refresh is done on saving.
	Open Text Block	Opens the selected text block.
	Define New Group	For details, refer to Create a group.
	Show Word Document	Brings the Word document of the document to the foreground.
	Switch View	Splits the screen to show both Content Composer Studio and Word Windows.
	Compile	Compiles the document.
	Content Composer Client	Test the document using the integrated Content Composer Client.
	Structure of Document	For details, refer to Create the report "Structure of a document".
	Used Data Links	For details, refer to Create the report "Used data connections".

Inserting a Text Block

To insert a text block in a document, complete the following steps.

1. Open the document in which the text block is to be inserted.
2. From the **Navigator** pane, drag the preferred text block to the **Editor** window.

   Note If required, you can insert the same text block more than once.
3. Optional. In the **Object Inspector** pane, edit the properties of the text block.
4. Optional. In the **Object Inspector** pane, edit the properties of the text block variables.
5. Optional. Check the position of the text block in **MicrosoftWord** and modify the position if necessary.

   Note Right-click the text block, then click **Select in Word** to highlight the reference to the selected text block in the Word document.
6. Optional. To modify the status of the text block, in the **Editor** window, right-click the text block and select one of the following menu items.
   - **Required**. The value of the **TextInsertionRule** property is set to **required**.
   - **Activate**. The value of the **TextInsertionRule** property is set to **activated**.
   - **Deactivate**. The value of the **TextInsertionRule** property is set to **deactivated**.
   - **Disabled**. The value of the **TextInsertionRule** property is set to **disabled**.
7. Optional. In the **Editor** window, right-click the text block and click **Inherit Attributes**.

   Note If this function is activated, the text block in the document inherits the properties of the original text block.

   For more information, refer to Activate inheritance.

Inserting a Picture

Before inserting a picture, ensure the file has the preferred image size.

To insert a picture, complete the following steps.

1. On the **Editor Tools** tab, click the **Show Word Document** button.
2. In **MicrosoftWord**, on the **Insert** tab, under **Illustrations**, click the **Picture** button.
3. In the **Insert Picture** dialog box, select a picture file.
4. Click the **Insert** down arrow and then click **Link to File**.

Assigning a Data Model

Assign a selection to the document to populate text block variables with values from a database table or XML file.

To assign a data model to a document, complete the following steps.

1. In the **Editor** window, below the bottom-right pane, click **Data Model**.
2. From the **Navigator** pane, drag the required selection to the left-hand pane of the **Editor** window. The selection displays in the upper-right pane of the **Editor** window.

   Note If the document already has a selection defined, this operation overwrites the link.
3. In the upper-right pane of the **Editor** window, click one of the data objects in the selection. The variables of the data object display in the bottom-right pane.
4. Drag the variables of the data object to the appropriate text block variables.
5. Optional. To remove an existing data link, right-click the variable or the text block and click **Remove Data Link**.
6. Optional. To delete the value of the **AttributeName** property, right-click the variable and click **Reset Attribute Name**.

Property Transfer

About Property Transfer

- You can transfer properties both between text blocks and between variables. Possible target objects are any objects of the same type at all levels.
- You can transfer between objects inside the current object window as well as between any other object windows open. The objects must be of the same type.

Transfer Properties

To transfer properties between objects with different object IDs, complete the following step.

- Press CTRL and drag the object whose properties you want to transfer onto the target object.

To transfer properties between objects with the same object IDs, complete the following steps.

1. Press CTRL and drag the object whose properties you want to transfer onto the target object.
2. In the **Selection** dialog box, select one of the options.
3. Click the **Apply** button.

Testing a Document

To test a document, complete the following steps.

1. Open a document template.
2. On the **Editor Tools** tab, under **Test**, click the **Compile** button, and then check the **Output**pane for success or failure messages.
3. On the **Editor Tools** tab, under **Test**, click the **Content Composer Client** button. The document opens as a test bundle in a new tab.
4. Optional. To toggle the status of a text block, perform the following substeps.
1. In the **Editor** window, in the left pane, click the document.
2. In the right-pane, right-click one of the text blocks and select **Switch state**.
5. Optional. From the **Navigator** pane, drag additional text blocks to the right pane of the E**ditor** window.
6. Optional. To replace a text block, assign an external document.
7. On the **Processes** tab, click the **Create Document** button.
8. Double-click the document to open it.

Assigning an External Document

Use this function to replace existing text blocks in a document with external documents.

Note You can only assign an external document to text blocks without child objects.

To replace a text block with an external document, complete the following steps.

1. In the left pane, click the document.
2. In the right pane, right-click the text block and click **Assign external document**.
3. In the **Select external document** dialog box, select a document and click **Open**.
The text **(external document)** displays to the right of the text block.
4. Optional. To remove the external document, right-click the text block and click **Remove external document**.

Note Using the LoadExternalDocument function in a script overwrites the values set with this function for the corresponding object.

Using the Document Preview

To use the document preview, complete the following steps.

1. On the **Editor Tools** tab, under **Test**, click the **Content Composer Client** button.
   The document opens as a test bundle in a new tab.
2. On the **Processes** tab, click the **Create Document** button.
3. Double-click the document to open it.
4. In **Content Composer Studio**, in the document template **Editor** window, below the bottom-right pane, click **Document Preview**.
5. In the **Document Preview** pane, click the **Refresh** button.
6. Optional. In the **Editor** window, in the left pane, select a text block or variable in the document structure.
   The corresponding text highlights by a colored background in the **Document Preview** pane.
7. Optional. To print the document on a local printer, in the **Editor** window, in the **Document Preview** pane, click the **Print** button.
8. Optional. To open an external **Microsoft Word** document in the **Document Preview** pane, in the **Editor** window, in the **Document Preview** pane, click the **Open File** button.

Variable Documents

About Variable Documents

Using scripts and conditions you can control whether a document is inserted into a bundle.

In some cases, the document to be inserted can be directly linked to a value in the data.

Example The document to be inserted depends on the country of the recipient.

So that you do not have to predefine all documents that might possibly be required in the bundle structure and then activate or deactivate the required document by way of conditions, it is possible to define a placeholder document. This placeholder document is replaced during document generation by the required document depending on the data content.

Notes

- To detect any variable documents that were not found, check the MissedObjects bundle property in a script. The content of MissedObjects is cleared before every document generation.
- In general, the scripts assigned to the BeforeInsert and AfterInsert events of the placeholder document are executed.
  If no scripts are assigned to the placeholder document for these events, then the scripts of the inserted document are executed.

Defining a Variable Document

To define a document as a variable document, complete the following steps.

1. Open the document template.
2. In the **Object Inspector** pane, right-click the property **VariableObjectDefinition** and click **Create**.
3. In the **Object Inspector** pane, click the **plus** to the left of **VariableObjectDefinition**.
   As a result, the following additional properties display below the object definition.

- AttributeName
- DataLink
- DataSource
- LookupSource
- LookupKey
- XPathExpression

Note These properties define the data field that contains the name of the document to be inserted. For more information, refer to Variable Properties.

4. Perform one of the following actions.

- In the **Object Inspector** pane, type a value for the properties **AttributeName** and **DataLink.**
- Perform the following substeps.

1. In the **Object Inspector** pane, click the **XPathExpression** property and then click the ellipsis button next to the property.
2. In the **Open** dialog box, select an XML file.
3. In the **XPath Evaluator** dialog box, in the **Sample XML tree** box, right-click the variable that you want to assign to the text block variable and then click **Copy XPath**.
4. In the **XPath Expression** box, right-click and click **Paste**.
5. Click the **Evaluate** button.
6. Check the result in the **XPath Result** box.
7. Click the **Apply** button.

5. In the **Object Inspector,** select a value for the property **DataSource**.
6. Optional. To assign an alias table to look up the value retrieved, use the property **LookupSource**. For more information, refer to Create an alias table.

The Content Control CopyLabel

About the Content Control CopyLabel

The content control CopyLabel enables you to define an area in a document template to insert the value of the property CopyLabel when a document copy is generated.

Activating the Developer Tools in Microsoft Word

In Word, check if the **Developer** tab is available. If not, perform the following steps.

- **In Microsoft Word 2007**

1. Click the **Microsoft Office Button** > **Word Options.**
2. In the **Word Options** dialog box, click **Popular**.
3. Select **Show Developer tab in the ribbon**.

- **In Microsoft Word 2010 and 2013**

1. On the **File** tab, click the **Options** button.
2. In the **Word Options** dialog box, click **Customize Ribbon**.
3. In the **Main Tabs** list, select the **Developer** check box.

Creating the Content Control CopyLabel

To insert a CopyLabel content control, complete the following steps.

1. In **MicrosoftWord**, place the cursor where you want to insert the content control.
2. On the **Developer Tools** tab, under **Control Elements,** click the **Rich Text** button.
3. On the **Developer Tools** tab, under **Control Elements,** click the **Properties** button.
4. In the **Content Control Properties** dialog box, in the **Title** box, type a title.
5. In the **Tag** box, type **mp:CopyLabel** and then click **OK**.
6. Reposition the control to your preferred location.

Groups and Actions

About Groups and Actions

Groups and Actions restrict business user actions by way of predefined rules.

Groups

A group consists of two or more text blocks. The relationship type defines how many text blocks in the group the user can activate.

Actions

Actions define relationships between a primary text block and one or more secondary text blocks.

The status of secondary text blocks depends on the status of the primary text block.

Example

- The primary text block has the status **Activated**.
- The secondary text block has the status **Deactivated**.

If the user toggles the primary text block to **Activated**, the status of the secondary text block automatically changes to **Deactivated**.

Creating a Group

When you create a group, make sure that using multiple groups does not result in mutual exclusion. You risk mutual exclusion when using the same object in several groups. Also, do not apply a group rule that may compromise another rule.

To create a new group, complete the following steps.

1. Open a document template.
2. In the **Editor** window, below the bottom-right pane, click the **Groups/Actions**tab.
3. In the upper-right pane of the **Editor** window, right-click and click **New Group**.
4. To define the relationship, right-click the group and then click one of the following menu items.
   - **Minimum one text block**. At least one text block in the group must be activated.
   - **Exactly one text block**. Only one text block in the group can be activated.
   - **Maximum one text block**. Only one or no text block in the group can be activated.
5. From the **Editor** window, from the left pane, drag two or more text blocks to the group.

Creating an Action

When you create an action, make sure that the action does not compromise prevailing group logic when you combine it with other defined actions.

To create a new action, complete the following steps.

1. Open a document template.
2. In the **Editor** window, below the bottom-right pane, click the **Groups/Actions**tab.
3. In the **Editor** window, from the left pane, drag the primary text block to the bottom-right pane.

4. In the **Editor** window, from the left pane, drag the dependent text blocks on the first text block in the bottom-right pane.
5. In the bottom-right pane, right-click the text blocks and select one of the following menu items.

- Activated
- Deactivated
- Required
- Disabled

Reports

Generating the "Structure of a document" Report

Content Composer Studio allows you to save a record of the text blocks, child text blocks, attributes, and variables of a document as a DOCX file, using the DOCX file specified in the configuration file **Composer.Studio.exe.config** as a template.

To generate a document report, complete the following steps.

1. Open the document.
2. On the **Editor Tools** tab, under **Reports**, click the **Generate report "Structure of document/text block"** button.
3. In the **Report Creation** dialog box, select a file or type the name of the report file.
4. To generate the report, click the **Save** button.
5. In the following dialog box, decide whether to open the DOCX file immediately.

Generating the "Used data connections" Report

Content Composer Studio allows you to save used data links of a document as a CSV file.

To document the used data links of a document, complete the following steps.

1. Open the document.
2. On the **Editor Tools** tab, under **Reports**, click the **Generate report "Used Data Links"** button.
3. In the **Save Used Data Links report** dialog box, select a file or type the name of the report file.
4. To generate the report, click the **Save** button.
5. In the following dialog box, decide whether to open the CSV file immediately.

Note When the report is saved as a CSV file, a special character separates the cells of the CSV file from each other. To determine this character, Document Composition uses the list separator specified in the Regional Settings section of the Windows Control Panel. We recommend not to change this setting.

Static Documents

Creating a Static Document

Static documents cannot contain text blocks. You can edit static Word documents.

To create a static document, complete the following steps.

1. In the bottom pane of the **Navigator**, right-click and click **New** > **Document**.
2. In the **Create New Document** dialog box, type a name for the document and click **OK**.
3. In the **Select Document Type** dialog box, select one of the static document types and click **OK**.
4. Perform one of the following substeps.

- To load a document permanently, perform the following steps.

1. On the **Editor Tools** tab, click the **Load Document** button.
2. In the **Open** dialog box, select a file and click **Open**.

- To load a document during bundle generation, perform the following steps.

1. Create a new script.
2. Assign the **Context**TextDocument to the script.
3. Use the statement **LoadExternalDocument** to specify the document that you want to load.
4. Compile and save the script.
5. In the **Editor** window of the document, in the **Object Inspector** pane, click the **Load** property and then click the ellipsis button next to the property.
6. In the **Select an Object** dialog box, select the script that you just created and then click the **Select** button.

5. Optional. In the **Object Inspector** pane, edit the properties of the document.

6. Optional. On the **Editor Tools** tab, click the **Show Word Document** button and edit the static Word document.

Static Word Documents

Before importing static Word documents, indicate whether to apply its existing document styles or the document template used by Content Composer.

To apply the styles of the specified Word template to the static document, complete the following steps.

1. In Microsoft Word, open the static document.
2. On the **Developer Tools** tab**,** under **Templates,** click the **Document Template** button.
3. In the **Templates and Add-ins** dialog box, select the **Automatically update document styles** check box.
4. Save and close the file.
5. In **Content Composer Studio**, open the static document.
6. On the **Editor Tools** tab, click the **Load Document** button.
7. In the **Open** dialog box, select the document and click **Open**.

Note With the **Automatically update document styles** option activated, Content Composer Web Client will not properly display the updated styles in static DOCX documents.

This limitation **only** affects the Web Client view. The printed DOCX documents will show the updated styles.

Static TIFF Documents

Specific settings regarding page width, page height, X and Y position and scaling of static TIFF documents can be made at the bundle property level.

This is done using the function SetArchiveAttribute to set the archive variable **tifflayout**.

The variable is passed on to the column **ODJ_TIFFLAYOUT** in the table **ODIN_JOB** for further processing in Odin.

The Odin-Rendition component uses the settings when converting to XPS.

Syntax

SetArchiveAttribute("tifflayout", "Page Width/Page Height/X-Position/Y-Position/Scaling Factor")

Parameter

Parameter	Meaning
Page Width	Width in pixels
Page Height	Height in pixels
X-position	X-Position of the TIFF
Y-position	Y-Position of the TIFF

Scaling Factor	Optional specification of the scaling factor defined in percent. If no specification is made, the default factor of 100 is used.

Example

SetArchiveAttribute("tifflayout", "794.0/1123.0/0.0/0.0/90")

Static Document Editor Tools

Icon	Name	Description
	Load document	Opens a dialog window to select a finished document.
	Save document to file	Allows you to save the static Word document as a DOCX file.
	Show Word document	Shows the static Word document.

Global Variables

About Global Variables

You can use global document variables in all scripts that have access to the pool **DOCUMENTPOOL**.

Creating a Global Variable

To create a global variable, complete the following steps.

1. In the **Editor** window, below the bottom-right pane, click **Global Variables**.
2. In the right pane of the **Editor** window, right-click and click **New Variable**.
3. In the **Create New Global Variable** dialog box, type a name for the global variable and click **OK**.
4. Optional. In the **Object Inspector** pane, edit the properties of the global variable.

Properties of Global Variables

Property	Description
DefaultValue	Value applied, if the variable is empty.

Manual	TRUE = Manual input is possible. FALSE = Manual input is not possible.
Variable Type	Possible values • Boolean • Float • Long • String

Conditions

Property	Description
Condition	If a condition is assigned and returns the result `True`, the document inserts into the bundle. If the condition returns the result `False`, the document does not insert into the bundle. For more information, refer to Assign a condition to an object. **Notes** The script defined here is interpreted after the **BeforeInsert** script executes. The condition has access to document data collected by the corresponding DataReference.

Data Model

Property	Description
Selection	Stores the associated selection.

Events

Event	Description
AfterInsert	This event is set for document references in a bundle. The script runs after the document is inserted into the bundle.

	Context: TextDocumentReference
BeforeInsert	This event is set for document references in a bundle. The script runs before the document is inserted into the bundle. If InsertObject is `True` after the script has run, processing continues with the evaluation of the condition. **Context**: TextDocumentReference
DefinitionArchiving	This event is set for document references in a bundle and executes after the DefinitionGrouping event. You can use the script to set attributes for archiving. Use the SetArchiveAttribute function to define 1-n archive attributes that pass on to Odin for each job. The following two alternative ways of defining the archive attributes are supported. **Alternative 1** The attributes to be used are assigned in a string in the following form, for example: by building a string list and converting it to a string as an archive attribute string assigned to the `Content` variable. `Name=Value <CR><LF>Name=Value` In doing so, each individual attribute inserts into the archive dictionary. **Example** StringlistAdd(List,ArchiveAttribute1=Value1) StringlistAdd(List,FormatStr(%s%s, Content = String("List") **Alternative 2** All scripts executed in the context TextDocumentReference can call the SetArchiveAttribute function to insert attributes into the archive dictionary. The attributes of the archive dictionary pass to Odin for each job. **Example** SetArchiveAttribute("MyArchiveVar1","Value1") SetArchiveAttribute("MyArchiveVar2","Value2") **Context**: TextDocumentReference

	For more information, refer toIdentical archive attributes for all documents in a bundleandTransfer data from a document to the ODIN_JOB table.
DefinitionGrouping	This event is set for document references in a bundle and executes after the DefinitionSorting event. You can use the script to set attributes for grouping. A sorting criterion is defined here as a string with a maximum of 256 characters. This passes on to Odin as the Grouping attribute for each job. Two alternative ways of defining the group string are supported. **Alternative 1** The group string to be used is assigned to the `Content` variable. In this way, the Grouping property of the TextDocumentReference context assigns to the corresponding value. **Example** Content = FormatStr("%s%s", "myGrouping: ", SelVariable ("PARTNER", "PAR_FirstName", "")) **Alternative 2** All scripts executed in the TextDocumentReference context have read and write access to the Grouping property. The content of this property passes on to Odin as the attribute grouping for each job. As in Alternative 1, it thereby is possible to extend or overwrite a predefined grouping criterion in subsequent events. **Example of extending the grouping criterion** dummy_Group = Grouping Grouping = FormatStr("%s%s", "MyGroupimgAdd+,dummy_Group) **Context**: TextDocumentReference
DefinitionSorting	This event is set for document references in a bundle and executes after documents are created. You can use the script to set attributes for sorting. A sorting criterion is defined here as a string with a maximum of 256 characters. This passes on to Odin as the **Sorting** attribute for each job. Two alternative ways of defining the sort string are supported. **Alternative 1**

	The sort string to be used is assigned to the `Content` variable. The Sorting property of the TextDocumentReference context is assigned the corresponding value. **Example** Content = FormatStr("%s%s", "mySorting: ", SelVariable("PARTNER", "PAR_FirstName", "")) **Alternative 2** All scripts executed in the TextDocumentReference context have read and write access to the Sorting property. The content of this property passes on to Odin as the Sorting attribute for each job. As in Alternative 1, it thereby is possible to extend or overwrite a predefined sort criterion in subsequent events. **Example of extending the sort criterion** dummy_Sort = Sorting Sorting = FormatStr("%s%s", "MySortingAdd + ", dummy_Sort) **Context**: TextDocumentReference
Load	This script executes when a document is loaded. **Context**: TextDocument
Unload	This script executes after inserting all text blocks and before the **DefinitionSorting** event. **Context:** TextDocument

General

Property	Description
Archive	Specifies if the document is archived after it has been created. This property is available to documents in bundles.
CopyLabel	The text defined here inserts into the CopyLabel content control of a document copy. This property is available to document copies in bundles.

	For more information, refer to Create the Content Control CopyLabel.
CustomIndex	The value can be freely defined and used for subsequent queries. This property is available to documents in bundles.
CustomUIReference	The assigned Word User Interface. This property is available to documents in bundles.
DataID	This property stores the associated selection. This property is available to documents in bundles.
DocumentReference	Name of the document.
Expandable	Defines whether the user can see the document structure and is allowed to activate or deactivate its objects.
NumberOfCopies	Number of copies of this document to create. This property is available to documents in bundles.
OriginalID	Unique ID of the object. This property is available to document copies in bundles.
Password	If a password is defined, users can only edit text blocks with the property **TextComponentProtection=TRUE** if they know the password. This property is available to documents in bundles.
PrintDefinition	Use this property to define the printer, form and print time for the document. Specification of printer and form is optional. If the fields are not set, they must be defined by the business user in the Content Composer Client. In this case, activate the option **Changes allowed** and also the print time **unknown**. By choosing **local** as printing time, you set up the document 1. for direct printing 2. and in addition, for local printing. In the Content Composer Client, you are prompted to select printer and forms when choosing **Print and Archive**. This property is available to documents in bundles.

TextInsertionRule	<table><tr><th>Rule</th><th>Effect</th></tr><tr><td>Activated</td><td>If the document cannot be inserted during generation, no abort is forced.</td></tr><tr><td>Deactivated</td><td>The document is deactivated and the user cannot activate it.</td></tr><tr><td>Disabled</td><td>The document does not be insert into the bundle.</td></tr><tr><td>Required</td><td>Bundle generation aborts if the document cannot be inserted.</td></tr></table>
Title	The value defined here is the name of the document the user sees in the Content Composer Client.
Visible	Defines whether the document is visible to end users.

Object definitions

Property	Description
VariableObjectDefinition	Contains the settings for variable documents. To create a variable object definition, complete one of the following steps. • Drop an alias table on the document. • Right-click the **VariableObjectDefinition** property and click **Create**. The following additional properties come into view. These properties determine which document inserts. • AttributeName • DataLink • DataSource • LookupSource • LookupKey • XPathExpression For more information, refer to Variable objects.

OnBase Archive

Properties	Description
DIP File Configuration	The assigned OnBase DIP File Configuration. To assign an OnBase DIP File Configuration, complete one of the following steps. • Click the ellipsis button next to the property and select an **OnBase DIP File Configuration** in the **Select an Object** dialog. • To assign a new **OnBase DIP File Configuration** to be created, right-click the **OnBase Keyword Configuration** property and click **Create**.
Document Type	It is mandatory to specify an **OnBase Document Type** to archive a document in OnBase and set the **Archive** property to `True`. You must assign an **OnBase Document Type** to a single document, if the document is passed as a single file to the OnBase Hotfolder for archiving. To assign a **Document Type**, complete one of the following steps. 1. Click the ellipsis button next to the property 2. In the **Specify the OnBase Document Type** dialog box, in the **Name** field, type a name. **Note** We recommend to specify the name of the **OnBase ID** here. 3. In the **ID** field, type the **OnBase Document Type ID**. **Note** This **ID** is passed to the OnBase archive interface and must exist in your OnBase installation. 4. Click **Select**. 5. In the **Select an Object** dialog box, select an **OnBase Document API Configuration**. 6. Click **OK**.
Transfer single file to archive	The archival documents can pass as a combined PDF, individual file, or combined PDF and individual file. Possible values are `True` or `False`. This property is available to documents in bundles. Consider the following settings for the **Transfer single file to archive** property and the bundle property **Combine documents in a direct print stack**. • Bundle property is set to `True` and this Document property is set to `False`. This archival document is included in a combined PDF and passed to OnBase Archive Hotfolder. • Bundle property is set to `True` and this Document property is set to

`True`. This archival document is included in a combined PDF and additionally passed as an individual file to the OnBase archive Hotfolder.

- Bundle property is set to `False` and this Document property is set to `False`. This archival document is passed as an individual file to the OnBase archive.
- Bundle property is set to `False` and this Document property is set to `True`. This archival document is passed as an individual file to the OnBase archive.

Bundles

About Bundles

Bundles are logical collations of related documents. Collating business process related documents into a single object enables you to generate and archive documents in one common processing step.

In addition, you have the option to process copies of the original documents in this same single processing step mentioned above.

Creating a Bundle

To create a bundle, complete the following steps.

1. In the bottom pane of the **Navigator**, right-click and click **New** > **Bundle**.
2. In the **Create New Bundle** dialog box, type a name for the bundle and click **OK**.
3. Optional. From the **Navigator**, drag the preferred documents to the **Editor** window.
4. Optional. To modify the status of the document, in the **Editor** window, right-click the document and select one of the following menu items.
    - **Required**. Sets the value of the **TextInsertionRule** property to **required**.
    - **Activate**. Sets the value of the **TextInsertionRule** property to **activated**.
    - **Deactivate**. Sets the value of the **TextInsertionRule** property to **deactivated**.
    - **Disabled**. Sets the value of the **TextInsertionRule** property to **disabled**.
5. Optional. Create a document copy.
6. Optional. Assign a data model.

Bundle Properties

Events

In these scripts, you can access both the data of the data references assigned to the bundle and also the properties and functions of the **TextDocumentCollection** context.

Event	Description
AfterCreatingDocuments	This event executes at the end of bundle processing after all documents are created. **Context**: TextDocumentCollection
AfterDataRetrieval	The script defined here executes after data retrieval. **Context**: TextDocumentCollection
BeforeCreatingDocuments	The script defined here executes once at the start of a bundle. If, after the script has been executed, the InsertObject property is `True`, bundle processing continues with the InitializeDocumentCreation event. If it is `False`, bundle processing terminates without creating documents. **Context**: TextDocumentCollection
BeforePrintingAndArchiving	The script defined here executes once before printing and archiving the bundle begins. If, after the script has been executed, the AllowPrint property is `False`, a predefined logical error is thrown. **Context**: TextDocumentCollection
BeforePrintingDocument	The script defined here executes before the printing of each individual document. **Context**: TextDocumentReference
InitializeDocumentCreation	The script defined here executes after the BeforeCreatingDocuments event and before document generation. If, after the script has been executed, the InsertObject property is `True`, processing continues with the step generate documents. If it is `False`, bundle processing terminates without creating documents. **Context**: TextDocumentCollection
InitializePrintAndArchive	The script defined here executes once after the BeforePrintingAndArchiving event and adds archiving information defined in the variable content by the script to all documents contained in the bundle.

	For more information, refer to Identical archive attributes for all documents in a bundle and Transfer data from a bundle to the ODIN_JOB table. **Context**: TextDocumentCollection

General

Property	Description
DataID	Selection assigned to a bundle. Bundle scripts can access data collected by the selection.

OnBase Archive

Properties	Description
Combine documents in a direct print stack	• `True`. The archival documents contained in the Direct Print Stack are combined into one PDF file. The documents are added in the same order as in the bundle editor. This PDF file is then sent to the OnBase Hotfolder. **Note** Ensure that you specify a Document Type for a bundle. • `False`. The documents in the stack are passed as a single file to the OnBase Hotfolder irrespective of the value set for document property "Transfer single file to archive". **Note** Ensure that you specify a Document Type to a single document template.
DIP File Configuration	The assigned OnBase DIP File Configuration. To assign an OnBase DIP File Configuration, complete one of the following steps. • Click the ellipsis button next to the property and select an **OnBase DIP File Configuration** in the **Select an Object** dialog. • To assign a new **OnBase DIP File Configuration** tobe created, right-click the **DIP File Configuration** property and click **Create**.
Document Type	You must assign an **OnBase Document Type** to a bundle if the documents of the bundle are contained in a Direct Print stack, must be combined into one PDF file and archived in OnBase.

To assign a **Document Type**, complete the following steps.

1. Click the ellipsis button next to the property
2. In the **Specify the OnBase Document Type** dialog box, in the **Name** field, type a name.
   Note We recommend to specify the name of the **OnBase ID** here.
3. In the **ID** field, type the **OnBase Document Type ID**.
   Note This **ID** is passed to the OnBase archive interface and must exist in your OnBase installation.
4. Click **Select**.
5. In the **Select an Object** dialog box, select an **OnBase Document API Configuration**.
6. Click **OK**.

Bundle Editor Tools

Icon	Name	Description
	Open Document Template	Opens the selected document template.
	Create Copy	Inserts a copy of the selected document template.
	Define New Group	For details, refer to Groups.
	Compile	Compiles the bundle.
	Content Composer Client	Test the bundle using the integrated **Content Composer Client**.

Creating a Document Copy

To create a copy of a document in a bundle, complete the following steps.

1. Open the bundle.
2. In the **Editor** window of the bundle, click the document.

3. In the **Editor Tools** tab, click the **Create Copy** bottom.

   Result
   A copy of the document appears in the bundle, marked with a **C**.

Assigning a Data Model

A bundle can use several different selections. You can also assign an individual selection to each document in a bundle.

To assign a selection to a bundle, complete the following steps.

1. Open the bundle.
2. From the **Navigator**, drag the preferred selection onto the bundle in the **Editor** window.
   Note If the bundle already has a selection defined, this operation overwrites the link.
3. Optional. To remove an existing data link, right-click the bundle and click **Remove Data Link**.

To assign a selection to a document in a bundle, complete the following steps.

1. Open the bundle.
2. From the **Navigator**, drag the preferred selection onto a document in the **Editor** window.
   Note If the document already has a selection defined, this operation overwrites the link.
3. Optional. To remove an existing data link, right-click the document and click **Remove Data Link**.

Prevent a User from Printing a Bundle

To prevent a user from printing a bundle created by the user, complete the following steps.

1. Create a new script.
2. In the **Object Inspector** pane, click the **Context** property, then click the down arrow next to the property and select **TextDocumentCollection**.
3. In the script, set the **BlockBundle** property to `True`.
4. Optional. In the script, set a value for the **DestinationGroup** property.
5. Optional. In the script, set a value for the **DestinationUser** property.
6. Compile and save the script.
7. Open the bundle.
8. In the **Object Inspector** pane, assign the script to one of the following events.
   - AfterCreatingDocuments
   - AfterDataRetrieval

- BeforeCreatingDocuments
- InitializeDocumentCreation

Sample Script

```
BlockBundle = true
DestinationGroup = "Adminrole"
DestinationUser = "Administrator"
```

Result

When you click the **Print and Archive** button in a **Content Composer Client**, the **Forwarding** dialog box displays. There, you can select a user group and/or a single user as the target for the bundle. If you select a user group *and* a user, the system forwards the bundle to both the group and the individual user.

If **DestinationGroup** and/or **DestinationUser** properties are set in the script, an information message box opens and the system automatically forwards the process to the specified group and/or user.

Groups and Actions

About Groups and Actions

Groups and Actions restrict business user actions by way of predefined rules.

Groups

A group consists of two or more documents. The relationship type defines how many documents in the group the user can activate.

Actions

Actions define relationships between a primary document and one or more secondary documents.

The status of secondary documents depends on the status of the primary document.

Example

- The primary document has the status **Activated**.
- The secondary document has the status **Deactivated**.

If the user toggles the primary document to **Activated**, the status of the secondary document automatically changes to **Deactivated**.

Creating a Group

When you create a group, make sure that using multiple groups does not result in mutual exclusion. You risk mutual exclusion when using the same object in several groups. Also, do not apply a group rule that may compromise another rule.

To create a new group, complete the following steps.

1. Open the bundle.
2. In the **Editor** window, below the bottom-right pane, click the **Groups/Actions**tab.
3. In the upper-right pane of the **Editor** window, right-click and click **New Group**.
4. To define the relationship, right-click the group and select one of the following menu items.
    - **Minimum one document**. At least one document in the group must be activated.
    - **Exactly one document**. Only one document in the group can be activated.
    - **Maximum one document**. Only one or no document in the group can be activated.
5. From the **Editor** window, from the left pane, drag two or more documents to the group.

Creating an Action

When you create an action, make sure that the action does not compromise prevailing group logic when you combine it with other defined actions.

To create a new action, complete the following steps.

1. Open the bundle.
2. In the **Editor** window, below the bottom-right pane, click the **Groups/Actions** tab.
3. In the **Editor** window, from the left pane, drag the primary document to the bottom-right pane.
4. In the **Editor** window, from the left pane, drag the dependent documents onto the first document in the bottom-right pane.
5. In the bottom-right pane, right-click each document and select one of the following menu items.

- Activated
- Deactivated
- Required
- Disabled

Dynamic Bundles

About Dynamic Bundles

In a dynamic bundle, the documents to be inserted and their respective properties are defined during the actual bundle creation step.

Creating a Dynamic Bundle

To create a dynamic bundle, complete the following steps.

1. Create a new script.

2. In the **Object Inspector** pane, click the **Context** property, then click the down arrow next to the property and select **TextDocumentCollection**.
3. To specify the documents you want to insert into the bundle, use the statements `AddDocumentByName` or `AddDocumentByObjectId`.
4. Compile and save the script.
5. Open the bundle.
6. In the **Object Inspector** pane, in the section **Events**, click the **AfterDataRetrieval** property and then click the ellipsis button next to the property.
7. In the **Select an Object** dialog box, select the script that you just created and then click the **Select** button.

Testing a Bundle

Testing a Bundle

To test a bundle, complete the following steps.

1. Open the bundle.
2. On the **Editor Tools** tab, under **Test**, click the **Compile** button and then check the **Output**panefor success or failure messages.
3. On the **Editor Tools** tab, under **Test**, click the **Content Composer Client** button. The bundle opens in a new tab.
4. Optional. To toggle the status of a document, in the **Editor** window, right-click the document and click **Switch state**.
5. Optional. To toggle the status of a text block, complete the following substeps.
    1. In the left pane, click the document.
    2. In the right-pane, right-click one of the text blocks and click **Switch state**.
6. Optional. From the **Navigator**, drag additional documents to the left pane of the **Editor** window.
7. Optional. From the **Navigator**, drag additional text blocks to the right pane of the **Editor** window.
8. On the **Processes** tab, click the **Create Document** button.

Notes

- Generating the bundle creates all activated original documents.
- Copies are not created until the documents are printed or passed on to Odin.

9. Double-click a document to open it.
10. Optional. To save a generated document to a specific directory, right-click the document and click **Save Document As**.

Assigning an External Document

You can use this function to

- replace active text blocks without child objects with external documents.
- replace active static documents in the bundle with external documents.

To replace a text block with an external document, complete the following steps.

1. In the left pane, click the document.
2. In the right pane, right-click the text block and click **Assign external document**.
3. In the **Select External Document** dialog box, select a document and click **Open**.
   The text (**external document**) displays to the right of the text block.
4. Optional. To remove the external document, right-click the text block and select **Remove external document**.

To replace a static document with an external document, complete the following steps.

1. In the left pane, right-click the static document and click **Assign external document**.
2. In the **Select External Document** dialog box, select a document and click **Open**.
   The text (**external document**) displays to the right of the document.

Note Documents in a bundle can be replaced by external documents of the same type.

3. Optional. To remove the external document, right-click the document and click **Remove external document**.

Note Using the **LoadExternalDocument** function in a script overwrites the values set with this function for the corresponding object.

Identical Archive Attributes for all Documents in a Bundle

If all documents in a bundle are to be given the same archive attributes, you can use the bundle event InitializePrintAndArchive to do this in a single step.

The 1:n archive attributes defined there are passed to Odin for each of the jobs in the bundle.

Two alternative ways of defining the archive attributes for all documents are supported.

Alternative 1

The attributes to be used are defined as a string in a script assigned to the bundle event InitializePrintAndArchive, in the form of

Name=Value <CR><LF>Name=Value

which is then assigned as the archive attribute string of the variable ArchiveData(Content).

In doing so, each individual attribute is inserted into the archive dictionaries of all documents.

The string can be created, for example, by building a string list and converting it to a string.

Example

StringlistAdd(List,FormatStr(%s%s,

StringlistAdd(List,FormatStr(%s%s,MyArchiveVar2=,Value2)

ArchiveData=String(List)

Alternative 2

All scripts executed in the context TextDocumentCollection can call the **SetArchiveAttribute** function to insert attributes into the archive dictionaries of all documents.

The archive attributes are then passed to Odin for each of the jobs in the bundle.

The event BeforePrintingDocument is not suitable for this procedure, because, unlike other bundle events, this event is not just executed once but for each individual document in the bundle.

Example

SetArchiveAttribute("CreatorUser,ModusUser.UserId)

SetArchiveAttribute("MyArchiveVar2","Value2")

Transfer Properties

To transfer properties between documents, complete the following step.

- Press `CTRL` and drag the document whose properties you want to transfer onto the target object.

Reports

About Reports

Content Composer not only provides text block reports and document reports related to single objects. It also provides a number of global reports.

To create global reports, the user must have the **Reports (38)** right.

When you save a report as a CSV file, a special character separates the cells of the CSV file from each other. To determine this character, Content Composer uses the list separator specified in the Regional Settings section of the Windows Control Panel. We recommend that you do not change this setting.

Generating the Text Blocks Report

This report displays a tree structure representing existing documents that indicate the text blocks used in each of them.

To generate the Text Blocks report, complete the following steps.

1. On the **Reports** tab, click the **Text Blocks** button.
2. On the **Text blocks used within documents** tab, click the **Generate Report** button.
3. Optional. To collapse all nodes, in the **Editor** window, click the **Collapse all** link.
4. Optional. To expand all nodes, in the **Editor** window, click the **Expand all** link.
5. Optional. To expand or collapse the node of a specific document, in the **Editor** window, click the plus or minus sign to the left of the document.
6. Optional. To save the report, complete the following substeps.
    1. Click the **Export Report as CSV file** button.
    2. In the **Save Report** dialog box, type a name for the CSV file and click **Save**.
    3. In the **Report Generation** dialog box, decide whether to open the CSV file immediately.

CSV file structure

Column	Content
DocumentName	Document name
TextBlock	Text block used in document

Generating the Unused Objects Report

This report displays unused objects, grouped by the following object types.

- Text block
- Document
- Condition
- Script

The report displays all objects for which the following conditions are **not** true.

- The object is a child object of another object.
  Example
  A text block is used by another text block.
- The object is referenced by another object.
  Example
  A script is executed by a text block event.

To generate the Unused Objects report, complete the following steps.

1. On the **Reports** tab, click the **Unused Objects** button.
2. On the **Unused Objects** tab, click the **Generate Report** button.
3. In the **Generate Report on 'Unused Objects'** dialog box, specify the object types you want to display in the report and then click **Generate report**.

   Notes

   - Objects used in no other way than being called by a **MonaLisa** script at runtime display as unused.
   - The folder symbol indicates that a folder references the object.
   - The transfer job symbol indicates that a transfer job references the object.
4. Optional. To collapse all nodes, in the **Editor** window, click the **Collapse all** link.
5. Optional. To expand all nodes, in the **Editor** window, click the **Expand all** link.
6. Optional. To expand or collapse the node of a specific object, in the **Editor** window, click the plus or minus sign to the left of the object.
7. Optional. To save the report, complete the following substeps.

   1. Click the **Export Report as CSV file** button.
   2. In the **Save Report** dialog box, type a name for the CSV file and click **Save**.
   3. In the **Report Generation** dialog box, decide whether to open the CSV file immediately.

CSV file structure

Column	Content
ObjectType	Object type

ObjectName	Object name
VersionNumber	Version number
UsedByFolder	Value is 1 if the object is used by a folder
UsedByTransferjob	Value is 1 if the object is used by a transfer job
Context	Context (of scripts or conditions)
ObjectID	Object ID
LastModifiedAt	Date of last modification
LastModifiedBy	Last modified by
CreatedAt	Date of last generation
CreatedBy	Created by

Generating the Used Data Connections Report

This report displays the existing documents indicating the data links used in each of them.

To generate the Used Data Connections report, complete the following steps.

1. On the **Reports** tab, click the **Data Connections** button.
2. On the **Used data connections** tab, click the **Generate Report** button.
3. Optional. To collapse all nodes, in the **Editor** window, click the **Collapse all** link.
4. Optional. To expand all nodes, in the **Editor** window, click the **Expand all** link.
5. Optional. To expand or collapse the node of a specific document, in the **Editor** window, click the plus or minus sign to the left of the document.
6. Optional. To save the report, complete the following substeps.
    1. Click the **Export Report as CSV file** button.
    2. In the **Save Report** dialog box, type a name for the CSV file and click **Save**.
    3. In the **Report Generation** dialog box, decide whether to open the CSV file immediately.

CSV file structure

Column	Content
DocumentName	Document name
ObjectType	Object type
ObjectName	Object name
DataSource	Data source
DataLink	Data link
AttributeName	Attribute name

Scripts and Conditions

About Scripts and Conditions

The script language MonaLisa offers a wide range of possibilities to enhance the functional scope of Content Composer Studio.

Scripts

Scripts enable you to solve specific tasks and to implement requirements that the use of standard objects alone cannot fully cover. With scripts, you gain full access to all available business logic stored in the various variable pools.

The scripting language encompasses function calls to various contexts and offers control structures such as loops and branching.

For more information on script language syntax, refer to the MonaLisa Functions section in this guide.

Conditions

Conditions are scripts that, in connection with the process of document generation, check the validity of a statement.

A simple condition, for example, checks whether a field contains a specific value. The outcome passed to the variable **Result** is either **True** or **False**.

Conditions enable you to activate or deactivate variables, text blocks, or documents according to the value returned.

Creating Scripts and Conditions

Creating a Script

To create a script, complete the following steps.

1. In the bottom pane of the **Navigator**, right-click and click **New** > **Script**.

2. In the **Create New Script** dialog box, type a name for the script and click **OK**.
3. In the **Object Inspector** pane, click the **Context** property.
4. Click the down arrow next to the property and select the context for your script.
   Note The context determines which objects the script can be assigned to and which properties and functions are available.
5. In the **Editor** window, type the script.
6. On the **Script Note Editor Tools** tab, under **Control**, click the **Compile** button.
7. Observe success or failure messages in the **Output** pane.

Creating a Condition

To create a condition, complete the following steps.

1. In the bottom pane of the **Navigator**, right-click and click **New** > **Condition**.
2. In the **Create New Condition** dialog box, type a name for the condition and click **OK**.
3. In the **Object Inspector** pane, click the **Context** property.
4. Click the down arrow next to the property and select the context for your condition.
   Note The context determines which objects the condition can be assigned to and which properties and functions are available.
5. In the **Editor** window, type the condition.
6. Optional. To access variables of the data model, complete the following substeps.
   1. On the **Script Note Editor Tools** tab, under **Show**, click the **Selection Browser** button.
   2. In the upper-right pane of the **Editor** window, click the **Selection** box, and then click one of the displayed selections.
   3. Next to the **Selection** box, click the arrow button.
   4. In the upper-right pane of the **Editor** window, click a data object. The data object variables display in the bottom-right pane.
   5. Optional. From the bottom-right pane, drag variables to the left pane. This creates the command **SelVariable** in the **Editor** window.
7. Assign a boolean value to the variable **Result** as the result of the conditional statement.

   Example
   Result = true
8. On the **Script Note Editor Tools** tab, under **Control**, click the **Compile**

button.

9. Observe the **Output** pane for success or failure messages.

For more information, see also Operators.

Script and Condition Editor Tools

Icon	Name	Description
	Export Script	Allows you to save a script in a file.
	Debug Service	Enables you to add the opened script to the debug service, or remove it as the case may be.
	Compile	Scripts and conditions have to be compiled. The corresponding button is available in the toolbar.
	Execute	To run a script or condition, click the button **Execute**. Code is executed in the background.
	Debug	Scripts that only contain MonaLisa standard functions can be executed directly in debug mode, if the context **None** is assigned. For more information, refer to Test Scripts and Conditions.
	Selection Browser	Opens the **Selection Browser** in the **Editor** window.
	Encrypt Password	Opens a dialog window where you can encrypt a password and copy it to the clipboard.
	Activating SOAP Web Service Tool	Opens the **SOAP Web Service Tool** allowing inspection of a SOAP web service. To activate the tool, click the button. Clicking again closes the tool. For more information, refer to Inspect a SOAP Web Service.

	Edit	Provides a number of functions for editing script code.

Code Completion

For ease of use, short cut key combinations can be used while writing code.

Standard Functions

The key combination **CTRL + Space** opens a list of all MonaLisa standard functions.

This list also shows information about the function of the statement and its parameters.

Using ENTER or a double-click you can insert the command into the script editor.

Properties, functions and pools

The key combination **CTRL + ALT + Space** opens a list of properties, functions and pools available in the current context.

Control Structures

Control structures can be accessed by the key combination **CTRL + K + X** .

Alternatively you can press the Tab key for one of the listed keywords to display the corresponding command structure.

For more information, see chapter Control Structures.

Other key combinations

Shortcut	Effect
ALT + right arrow or CTRL + Space	Displays word completion.
CTRL + K + C	Marks the current code line as a comment.
CTRL + K + U	Remove the comment syntax from the current code line.
CTRL + K + F	Indents the code line selected like the surrounding code lines.
SHIFT + ALT + T	Moves a line with the cursor below the next line.

CTRL + U	Converts selected text to lowercase.
CTRL + SHIFT + U	Converts selected text to uppercase.
CTRL + SHIFT + Space	Shows information about the current parameter.
CTRL + DEL	Deletes the word to the right of the cursor.
CTRL + Backspace	Deletes the word to the left of the cursor.
CTRL + F	Displays the search dialog.
CTRL + H	Displays the search and replace dialog.
F3	Continues the search down the text.
SHIFT + F3	Continues the search up the text.
CTRL + A	Selects the whole content.
CTRL + W	Selects the word where the cursor is positioned or the word to the right of the cursor.

Setting a Bookmark

Use the **Bookmarks** feature to skip to specific lines in source code more quickly.

To set a bookmark, complete the following steps.

1. In the **Editor** window of the script or the condition, place the cursor in the line where you want to set a bookmark.
2. Press CTRL + K + a number between 1 and 9.

To delete a bookmark, complete the following step.

- Press CTRL + K + the number of the bookmark.

To skip through to a defined bookmark, complete the following step.

- Press CTRL + the number of the bookmark.

Displaying Context Information

The context information shows you the available variable pools and process components for the context you specified.

To view the context information, complete the following step.

- Below the **Editor** window, click the **Context Information** tab.

Creating a Script Parameter

You can specify script parameters and standard values when another script calls the script with the parameter passing function.

To create a script parameter, complete the following steps.

1. Below the **Editor** window, click the **Parameter** tab.
2. Click the **New Parameter** button.
3. In the **Parameter Name** box, type a name for the parameter.
4. Optional. In the **Standard Value** box, type a standard value.

For more information, refer to CallSystem.

Adding a comment

To add a comment for a script, complete the following steps.

1. Below the **Editor** window, click the **Comment** tab.
2. In the **Editor** window, type your comment.

Assigning a Script to an Object

Show All

You can assign scripts to events of variables, text blocks, documents, and bundles. You can also assign scripts to process designer blocks and components.

To assign a script to an event of a variable, a text block, a document, a bundle, or a process component, complete the following steps.

1. Open the object.
2. In the **Editor** window, click the object to which you want to assign the script.
3. In the **Object Inspector** pane, in the **Events** section, click the event property and click the ellipsis button next to the property.
4. In the **Select an object** dialog box, select the preferred script and click the **Select** button.

Notes

- In the **Select an object** dialog box, only scripts with the appropriate context are available for selection.
- In the **Editor** window, a line appears above the box next to the object after you assign a **BeforeInsert** script. A line appears below the box signals after you assign an **AfterInsert** script.

.

- In the **Object Inspector** pane, double-click an event to open the assigned script. If you double click an event with an empty event property, the **Create New Script** dialog box opens. The script automatically receives the context that fits.

Assigning a Condition to an Object

Show All

You can assign conditions to variable, text block and document events. You can also assign conditions to process designer blocks and components.

To assign a condition to a variable, a text block or a document, complete the following steps.

1. Open the object.
2. In the **Editor** window, click the object to which you want to assign the condition.
3. In the **Object Inspector** pane, in the **Condition** section, click the condition property and click the ellipsis button next to the property.
4. In the **Select an object** dialog box, select the preferred condition and click the **Select** button.

Notes

- In the **Select an object** dialog box, only conditions with the appropriate context are available for selection.
- In the **Editor** window, a question mark appears next to an object when you assign a condition.
- In the **Object Inspector** pane, double-click on the condition property to open the assigned condition. If you double-click a condition with an empty condition property, the **Create New Condition** dialog box opens. The condition automatically receives the context that fits.

Debugging

Debugging a Script or a Condition

Scripts and conditions that do not need a context for execution

In the debug mode, you can run scripts and conditions that do not use any specific context.

To debug a script or a condition, complete the following steps.

1. Open the script or condition.
2. In the **Object Inspector**, set the **Context** property to the value **None**.
3. In the **Script Note Editor Tools** tab, under **Control**, click the **Execute in Debug Mode** button.

The script opens in a new debug mode window.

4. Run the script using the buttons in the **Debugger Tools** tab.

Scripts and conditions that need a context for execution

In the debug mode, you cannot run scripts and conditions that use a specific context. Instead, run these scripts and conditions in the corresponding context.

To debug a script or a condition, complete the following steps.

1. In the **Navigator**, right-click the script or the condition and click **Add to Debug Service**.
2. Open the document or the process which uses the script and start the document creation or start the process.
   As soon as the script executes, the debugger window opens.
3. Run the script using the buttons in the **Debugger Tools** tab.

To disable execution in debug mode for a script or a condition, complete the following step.

- In the **Navigator**, right-click the script or the condition and click **Remove from Debug Service**.

Debugger Tools

Icon	Name	Description
	Run (CTRL + F9)	Executes the complete script and closes the Debugger. Does not takes into account any breakpoints.
	Run (F9)	Executes the complete script but does not close the Debugger. If breakpoints have been set, the script will stop at the corresponding code line and can be continued from this position.
	Set / Remove Breakpoint (F5)	Sets or removes a breakpoint at the current position.
	Step Into (F7)	Executes the current code line. If this statement is CallSystem the Debugger branches into the script being called.
	Procedure Step (F8)	Executes the current code line.
	Stop (CTRL + F2)	Ends the script and closes the Debugger.

	Quick Monitoring	Shows the current variable values.

SOAP Web Service

Inspecting a SOAP Web Service

The **SOAP Web Service Tool** allows you to explore the available API of a SOAP web service and to generate the appropriate MonaLisa Script code for calling a chosen function.

To inspect the web service and its available functions, complete the following steps.

1. Open the script.
2. In the **Script Note Editor Tools** tab, under **Show**, click the **Call Web Service** button**.**
3. **I**n the **Editor** window, in the right pane, in the **Namespace** box, check and modify the namespace if you work with the MLObjects class.
   Note The specified value transfers to the **Namespace** property of the MLWsClient class.
4. In the **WSDL-URI** box**,** type the URL of the web service.
5. To the right of the **WSDL-URI** box, click the **Load WSDL** button.
   The **Script Editor** displays the web service code.
6. Select a method from the **Method** drop-down menu.

To generate and process MonaLisa Script code, complete the following steps.

1. In the **Editor** window, in the right pane, click the **Generate Code** button.
   The appropriate MonaLisa script code displays in the output pane to the bottom-right of the**Editor**window.
   The code iscopied directly to the clipboard.
2. Optional. In the output pane to the bottom-right of the **Editor** window, edit the code and click the **Copy to Clipboard** link.

SOAP Web Service Pane

Box	Content
WSDL-URI	URL of the SOAP web service.

Method	List of functions provided by the web service.
Signature	Signature of the selected function, providing information about type, return type, and parameters of the function.
Parameter	List of function parameter names and types.

Trailing special character	Parameter type
&	out or reference
[]	array

Local variable	Name of the variable used in a Monalisa script for accessing the MLWsClient Extension object.

Script options check boxes	When selected
GetObject / FreeObject	Generate statements for creating and disposing a MLWsClient Extension object.
CallMethod	Generate code for setting the parameters and calling the selected web service function.
WebProxy Definition	Generate code statement to define a Webproxy used for accessing the web service.
WSDL-URI	Generate code for setting the Web Service URI in the Monalisa Script, using the URL of the web service provided in the UI.

MonaLisa Functions

Data Types

MonaLisa works with the data types Integer, BigInteger, Float and String.

Value ranges of the data types

DataType		Min	Max
I	Integer (int32)	-2147483648	2147483647
L	BigInteger (int64/long)	-9223372036854775808	9223372036854775807
F	Float (double)	-1.7976931348623157E+308	1.7976931348623157E+308
S	String		

Result data types

The following table lists the result data types according to data operations.

Operands		Result data type			
Operand 1	**Operand 2**	**Operator +**	**Operator -**	**Operator ***	**Operator /**
S	S	S	Error	Error	Error
S	I	S	Error	Error	Error
S	L	S	Error	Error	Error
S	F	S	Error	Error	Error
I	S	S	Error	Error	Error
I	I	I / L	I / L	I / L	F
I	L	L	L	L	F
I	F	F	F	F	F
L	S	S	Error	Error	Error
L	I	L	L	L	F
L	L	L	L	L	F
L	F	F	F	F	F

F	S	S	Error	Error	Error
F	I	F	F	F	F
F	L	F	F	F	F
F	F	F	F	F	F

Operators

The following operators are available.

Operator	Meaning
+	Adds two numbers.
-	Subtracts two numbers.
*	Multiplies two numbers.
/	Divides two numbers.
mod	Divides two integers and returns only the remainder. **Example** x = 3 mod 2 **Result** The variable x contains the remainder 1.
=	Returns `true` if the two values are identical.
<>	Returns `true` if the two values are not identical.
<	Returns `true` if the value on the left is less than the value on the right.
>	Returns `true` if the value on the left is greater than the value on the right
<=	Returns `true` if the value on the left is less than or equal to the value on the right.
>=	Returns `true` if the value on the left is greater than or equal to the value on the right

AND()	Logical AND
OR()	Logical OR
NOT()	Logical negation

Syntax

You can use operators to link a number of logical expressions together.

Enclose combined expressions in brackets to maintain their logical intent.

You can nest logical expressions.

Example

Result = ((SelVariable("PARTNER", "PAR_TITLECODE") = "1") OR (SelVariable("PARTNER", "PAR_TITLECODE") = "2"))

Control Structures

About Control Structures

Control structures enable you to control the flow of scripts and conditions. For example, control structures allow you to respond to different data contents.

CheckEvery

Checks a variable for its validity. If the first parameter passed is **EVERY**, more than one agreement is possible.

The variable to be checked is passed as the second parameter.

- Value checks the value passed for validity.
- Using **when** allows you to check further conditions.
- The statements after **when-any** are executed when one or more of the conditions are met.
- The statements after **when-all** are executed when all of the conditions are met.
- The statements after **when-none** are executed when none of the conditions are met.

Example

a = 11

erg = 0

```
c = false
;
Check("every", a)
  value(1)
    erg = erg + 1 ; if a is equal to 1
  when(c = true)
    erg= erg + 1  ; if c is TRUE
  When-Any
    message = "any"  ; if a is equal to 1 OR c is TRUE
  When-All
    message = "all"  ; if a is equal to 1 AND c is TRUE
  When-None
    message = "none" ; if a is not equal to 1 AND c is not TRUE
end-check
```

CheckFirst

Checks a variable for its validity. If the first parameter passed is **FIRST**, only the statements after the first agreement are executed.

The variable to be checked is passed as the second parameter.

- Value checks the value passed for validity.
- Using **when** allows you to check further conditions.
- The statements after **when-any** are executed when one or more of the conditions are met.
- The statements after **when-all** are executed when all of the conditions are met.
- The statements after **when-none** are executed when none of the conditions are met.

Example

```
a = 11
erg = 0
c = false
Check("FIRST", a)
  value(1)
    erg = 1   ; if a is equal to 1
```

```
  when(c = true)
    erg=2     ; if a is not equal to 1 AND c is TRUE
  When-Any
    Erg=3     ; if a is equal to 1 OR c is TRUE
  When-All
    Erg=4     ; if a is equal to 1 AND c is TRUE
  When-None
    Erg=5     ; if a is not equal to 1 AND c is not TRUE
end-check
```

If-Condition

Checks whether the condition in brackets is valid. If so, the following commands up to **End-if** are executed.

If the condition in brackets is not valid, the script is continued after the **End-if** command.

Each individual condition must be enclosed in brackets.

Multiple conditions, linked by **and** or **or** must be enclosed by additional brackets.

Examples

```
if (a < 10)
  b = 0
end-if
if ( (a = 10) and (b = 30) )
  c = 0
end-if
```

IfElse-Condition

Checks whether the condition in brackets is valid. If so, the following commands up to **Else** are executed.

The script then continues with the statements following the respective **End-If** command.

If the condition in brackets is not valid, the commands between **Else** and the respective **End-If** are executed.

Example

```
if (a < 10)
  b = 0
else
```

```
  b = 1
end-if
```

While-Loops

Executes the statements in a loop as long as the condition in parentheses is **TRUE**.

The condition is evaluated before the statements are executed.

Example

```
while (a < 100)
  a = a + 1
end-while
```

Region

Defines a section of a script that you can display or hide using the + or - symbols.

This helps make large scripts more manageable.

Example

```
#region InfoText
...
#endregion
```

Exception Handling

About Exception Handling

You can catch exceptions that occur during script execution by using the **Try - Finally** control.

TryFinally

If an error occurs while executing a statement, a script will typically terminate with an exception.

If the statement is inside a try-block, the statements in the finally-block are executed before the script is ended.

Example

```
try
  InitDatabase("ODIN")
finally
  Protocol("Inside Finally-block", 9)
end-try
```

TryOnError

If an error occurs while executing a statement in a try-block, use the command **Raise** to skip to the error label defined in **OnError**.

Only the first matching OnError block in a try-block will execute.

Choose the right order of OnError blocks so that specific OnError blocks are executed before more general ones.

Example

See TryOnErrorFinally

TryOnErrorFinally

If an error occurs while executing a statement in a try-block, use the command **Raise** to skip to the error label defined in **OnError**.

The code in the finally block is executed after the OnError block has been processed.

Example

```
;the special object "Error" contains information about the last error
;If no error occurred - all attributes contain an empty string
;
;Error.Category: the category identifies the sub-systems that threw the error (for example: "OdinBusinessFacade")
;Error.Token: the token is a one-word identifier for the error. Can be specified if you use the Raise function
;Error.Message: contains the error message
Protocol("before try: category '{0}'; error token: '{1}'; error message: '{2}'; ", 0, Error.Category, Error.Token, Error.Message)
;
;modify this value to play with the different OnError code-blocks
a = 10

try
  if(a < 10)
    ;This exception will be caught in the OnError("BUSINESS_ERR") block
    Raise("BUSINESS_ERR","a cannot be less than 10 sein")
  end-if
```

```
;
  ;This exception will be caught in the OnError() block
  ;because there is no OnError("Err") block
  Raise("Err", "Test-Raise")
;
;
OnError("BUSINESS_ERR")
  ;this code is executed if the exception token is "BUSINESS_ERR"
  ;the exception token is the first parameter of the Raise function
  Protocol("BUSINESS_ERR: {0}", 0, Error.Message)
;
  ;you can also rethrow the error
  ReRaise()
;
OnErrorCategory("OdinBusinessFacade")
  ;this code is executed if the category of the exception is "OdinBusinessFacade"
  Protocol("odin error caught", 0)
;
OnError()
  ;this code is executed if no matching OnError() was found up to this line
  ;information about the exception can be retrieved from the Error object
  Protocol("error category: '{0}'; error token: '{1}'; error message: '{2}'; ", 0, Error.Category, Error.Token,
Error.Message)
;
Finally
;
  ;code in the finally block is ALWAYS executed
  ;this is the right place to free resources (database connections, file handles)
  Protocol("Inside Finally-block", 9)
End-try
```

;

Protocol("after finally: error category: {0}; error token: '{1}'; error message: '{2}'; ", 0, Error.Category, Error.Token, Error.Message)

Standard Functions

Append

Inserts a dataset and fills fields with values.

Append		
SYNTAX	Append(datasetName,fields: String[])	
ARGUMENTS	datasetName	Name of the dataset.
	fields	Field values in the correct order.
EXAMPLE	`Append("Table","Name","3","26.10.2011")`	

Beep

Sends out the standard system signal tone.

Parameter	Definition
frequency	Frequency of the signal

BigInteger

Converts a value to a BigInteger value. The variable content must be a number format.

BigInteger	
SYNTAX	BigInteger(valueToConvert)

ARGUMENTS	valueToConvert	The variable to convert. Defining an empty string as parameter value results in an exception at run time.
RETURN	BigInteger value of the returned variable.	
EXAMPLE	Var = 100 Var = BigInteger(Var)	

Bof

Determines whether the cursor is at the start of the dataset.

Bof		
SYNTAX	Bof(datasetName)	
ARGUMENTS	datasetName	Name of the dataset.
RETURN	TRUE = The cursor is at the start of the dataset. FALSE = The cursor is not at the start of the dataset.	
EXAMPLE	Begin = Bof("Table")	

CallSystem

Calls a script from the same system.

Syntax

CallSystem(scriptName: String)

CallSystem(scriptName: String ,inputVariables: String)

CallSystem(scriptName: String ,inputVariables: String ,outputVariables: String)

Note about inVars/outvars:

Only those variables are passed that actually exist.

CallSystem

SYNTAX Name of the script to be executed

ARGUMENTS

scriptName	Name of the script you want to execute.
inputVariables	Comma-separated list with the names of the variables you want to pass (call by value).
outputVariables	Comma-separated list with the names of the variables you want to pass (call by value), these variables are also those returned.

RETURN Comma-separated list with the names of the variables to be passed (call by value)

EXAMPLE **Calling a script with no variable passing**

CallSystem("callTestScript")

Calling a script passing input variables

Amount1 = 1

Amount2 = 2

CallSystem("callTestScript", "Amount1,Amount2")

Here the script **callTestScript** is called from the system and the variables Amount1 and Amount2 are available in the called script.

Calling a script passing input variables and defining target variables

Amount1 = 1

Amount2 = 2

CallSystem("callTestScript", "Amount1=Number1,Amount2=Number2")

Here the script **callTestScript** is called from the system and the variables Amount1 and Amount2 are available as the variables Number1 and Number2 in the called script.

Calling a script passing input variables and output variables

Amount1 = 1

Amount2 = 2

CallSystem("callTestScript", "Amount1,Amount2", "Sum")

Protocol("Sum of Amount1 and Amount2: {0} ", 0, Sum)

Here the script **callTestScript** is called from the system and the variables Amount1 and Amount2 are available in the called script.

After the script is called, the variable Sum from the called script is passed back to the calling script and is available for further processing.

Close

Closes an open dataset.

Close		
SYNTAX	Close(datasetName)	
ARGUMENTS	datasetName	Name of the dataset.
EXAMPLE	Close("DSName")	

CloseDatabase

Closes the connection to a database.

CloseDatabase		
SYNTAX	CloseDatabase(aliasName: String)	
ARGUMENTS	aliasName	Name of the database alias.
EXAMPLE	CloseDatabase("DBAliasName")	

CloseSQL

Closes an open dataset.

CloseSQL

SYNTAX	CloseSQL(datasetName)
ARGUMENTS	datasetName — Name of the dataset.
EXAMPLE	CloseSQL("DSName")

Commit

Permanently stores all changes to data made in the current transaction. The current transaction closes.

The current transaction is the last transaction started by calling the StartTransaction function.

Commit

SYNTAX	Commit(datasetName)
ARGUMENTS	datasetName — Name of the dataset.
EXAMPLE	Commit("DSName")

Contains

Checks whether the source string contains the search string.

Contains

SYNTAX	Contains(sourceString, searchString)
ARGUMENTS	sourceString — The string you want to search through. searchString — The string you want to search for.
RETURN	TRUE = The source string contains the search string. FALSE = The source string does not contain the search string.
EXAMPLE	source_string = "Hyland Software"

	search_string = "Soft" ok=Contains(source_string ,search_string)

Contains_Exactly

Case sensitive check whether the source string contains the search string.

Contains_Exactly		
SYNTAX	Contains_Exactly(sourceString, searchString)	
ARGUMENTS	sourceString	The string you want to search through.
	searchString	The string you want to search for.
RETURN	TRUE = The source string contains the search string. FALSE = The source string does not contain the search string.	
EXAMPLE	source_string = "Hyland Software" search_string = "ware" ok=Contains_Exactly(source_string ,search_string)	

Count

Determines the number of records in a dataset.

Count		
SYNTAX	Count(datasetName)	
ARGUMENTS	datasetName	The name of the dataset.
RETURN	Number of datasets.	
EXAMPLE	num = Count("DSName")	

CreateGuid

This function creates a Globally Unique Identifier (GUID) and returns a string representation of it, according to the provided format specifier.
A GUID is a global 128 bit number and is an implementation of the Universally Unique Identifier-Standards (UUID).
A GUID is a typical represented in the format `XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX` in which each '`X`' is a placeholder for a hexadecimal character (0-9 or A-F).
For example, 936DA01F-9ABD-4D9D-80C7-02AF85C822A8.

CreateGuid	
SYNTAX	CreateGuid(format)
ARGUMENTS	format — A single-format specifier that indicates how to format the value of this GUID. The format parameter can contain the following format specifiers. All digits in the return value are hexadecimal. Each character '`x`' represents a hexadecimal digit; each hyphen ('-'), bracket ('{', '}'), and parenthesis ('(', ')') appears as shown. • `N`: 32 digits (`xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx`) • `D`: 32 digits separated by hyphens (`xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`) • `B`: 32 digits separated by hyphens, enclosed in brackets {`xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`} • `P`: 32 digits separated by hyphens, enclosed in parentheses (`xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`) If format is an empty string (""), "`D`" is used.
RETURN	The value of this GUID, represented as a series of lowercase hexadecimal digits in the specified format.
EXAMPLE	retString = CreateGuid("N")

CreatePool

Creates a new MonaLisa variable pool.

CreatePool		
SYNTAX	CreatePool(poolName)	
ARGUMENTS	poolName	Name of the pool.
EXAMPLE	CreatePool("MyPool")	

Decimal

Converts a value to floating point value (decimal).

Notes

- If the parameter for Decimal() is a string or a constant, the period is always the decimal separator.
- The thousands separator is always a COMMA.
- If, on the other hand, a MonaLisa variable is passed to Decimal(), the content is interpreted according to the regional Windows settings for DecimalSeparator and ThousandSeparator.

Decimal		
SYNTAX	Decimal(valueToConvert)	
ARGUMENTS	valueToConvert	The value you want to convert. **Note** An empty string as parameter value results in an exception at run time.
RETURN	Floating point value	
EXAMPLE	b = Decimal(a)	

Delete

Deletes the record with the specified record number.

Delete		
SYNTAX	Delete(datasetName, rowNum)	
ARGUMENTS	datasetName	Name of the dataset.
	rowNum	Row number of the record you want to delete.
EXAMPLE	Delete("DSName",1)	

DeletePool

Deletes the specified MonaLisa pool.

The following pools cannot be deleted.

- BAUSTEIN (not applicable to English version)
- BLOCKPOOL
- DOCUMENTPOOL
- DOKUMENT (not applicable to English version)
- MODUSUSER
- SYSTEM
- SYSTEMPOOL
- ZUSATZVARIABLEN (not applicable to English version)

DeletePool		
SYNTAX	DeletePool(poolName)	
ARGUMENTS	poolName	Name of the pool.
EXAMPLE	DeletePool("MyPool")	

EncryptString

The method returns a string containing the encrypted text.

EncryptString		
SYNTAX	EncryptString(ClearText)	
ARGUMENTS	ClearText	The string you want to encrypt.
RETURN	A string containing the encrypted text.	
EXAMPLE	retString = EncryptString("abc)	

EndsWith

Determines if the end of the input string matches the specified string.

EndsWith		
SYNTAX	EndsWith(inString, value, ignoreCase)	
ARGUMENTS	inString	The input string to check.
	value	The string value to compare with the end of the inString value.
	ignoreCase	TRUE = Ignore case in the search. FALSE = Consider case in the search.
RETURN	TRUE = A string match is found. FALSE = No string match is found.	
EXAMPLE	retPosition = EndsWith("	

Eof

Determines whether the cursor is at the end of the dataset.

Eof		
SYNTAX	Eof()	
ARGUMENTS	datasetName	Name of the dataset.
RETURN	TRUE = Eof reached. FALSE = Eof not reached.	
EXAMPLE	End = Eof("DSName")	

Execute

Executes a program.

Execute		
SYNTAX	Execute(commandLine,commandShow)	
ARGUMENTS	commandLine	Path
	commandShow	Win32 API parameter, defining how the window opens.
RETURN	ProgramHandle	
EXAMPLE	PrgHandle = Execute("C:\Tools\ProgramName.EXE",1)	

Note for 64 bit operating systems

Content Composer is a 32-bit program.

If you use the **ExecuteSynchron** command to start a program from the directory **%windir%\System32**, you must define the the directory **%windir%\Sysnative** instead.

If you use the **ExecuteSynchron** command to start a program from the directory **C:\windows\System32**, you must define the the directory **C:\windows\Sysnative** instead.

Refer to the Microsoft Developer Network website for more information on the File System Redirector.

Find

Searches the current index for a record that matches exactly the values specified.

Find		
SYNTAX	Find(datasetName,idxFields)	
ARGUMENTS	datasetName	Name of the dataset.
	idxFields	Field contents of the current index.
RETURN	TRUE = Dataset found. FALSE = Dataset not found.	
EXAMPLE	found = Find("odin", "3899")	

FindValue

Reads field values from a dataset according to a defined filter.

FindValue		
SYNTAX	FindValue(datasetName,filter,formattingRule,resultFields)	
ARGUMENTS	datasetName	Name of the dataset.
	filter	Filter criteria as comma separated string. Two alternatives are supported. **With definition of the string FILTER** If filter contains the string FILTER as first value, the rest of the value is interpreted as an expression and evaluated accordingly. An example of a possible expression is ACC_Par_Num = '0815'. **Without definition of the string FILTER**

	Specify filter values here in a comma separated filterString. The names of the corresponding column names derive from the dataset. Observe the order of the columns in the dataset when you define filter values.
formattingRule	Formatting rule for example "PartnerNr: %s, AccountType = %s, Balance = %s" If no formatting rule is used, the result values from the columns store without separators in the result string.
resultFields	Definition of the column names whose values the result string contains.

RETURN Values found.

EXAMPLE

```
InitDatabase("DataDbAlias")
InitDataSet("testSet","DataDbAlias","dbo.Accounts")
;Alternative1: with definition of FILTER --> independent of column order
StringListAdd("filterString", "FILTER")
StringListAdd("filterString", "KTO_Par_Num = '0815' ") ; Column name with search value for 1. column of the dataset
StringListAdd("filterString", "AND ")
StringListAdd("filterString", "KTO_Type = 'N'")       ; Column name with search value for 5. column
filter = String("filterString")
; no FormatRule
result = findValue("testSet", filter, "", "KTO_Par_Num", "KTO_Type", "KTO_Balance")
Protocol(result, 10, "") ; should be : 0815N24,34 --> Value from the first dataset found
; with FormatRule
formatRule = "PartnerNr: %s, AccountType: %s, Balance: %s"
result = FindValue("testSet", filter, formatRule, "KTO_Par_Num", "KTO_Type", "KTO_
Balance")
Protocol(result, 10, "") ; should be : PartnerNr: 0815, AccountType: N, Balance: 24,34 -->
Values from the first dataset found
```

```
;
;Alternative2: no definition of FILTER --> column order must be adhered to in filter
StringListClear("filterString")
StringListAdd("filterString", "0815")    ; Filter value for 1. column of the dataset
StringListAdd("filterString", "2")       ; Filter value for 2. column
StringListAdd("filterString", "55378924") ; Filter value for 3. column
filter = String("filterString")
; no FormatRule
result = findValue("testSet", filter, "", "KTO_Par_Num", "KTO_Type", "KTO_Balance")
Protocol(result, 10, "") ; should be : 0815N24,34 --> Value from the first dataset found
; with FormatRule
formatRule = "PartnerNr: %s, AccountType: %s, Balance: %s"
result = findValue("testSet", filter, formatRule, "KTO_Par_Num", "KTO_Type", "KTO_
Balance")
Protocol(result, 10, "") ; should be : PartnerNr: 0815, AccountType: N, Balance: 24,34 -->
Values from the first dataset found
```

First

Positions the dataset cursor on the first record in the dataset.

First		
SYNTAX	First(datasetName)	
ARGUMENTS	datasetName	Name of the dataset.
EXAMPLE	First("DSName")	

Float

Converts a value to a floating-point variable. The value used must be in a number format.

Notes

Float

SYNTAX Float(valueToConvert)

ARGUMENTS

valueToConvert	The variable to convert. **Notes** • If the parameter is a string or a constant, the period is always the decimal separator. The thousands separator is always a comma. • If, on the other hand, a MonaLisa variable is passed to Float(), the content is interpreted according to the Windows regional settings for DecimalSeparator and ThousandSeparator. • An empty string as parameter value results in an exception at run time.

RETURN Floating-point variable.

EXAMPLE

```
FloatVar = Float("9.15")        ;FloatVar = 9.15
FloatVar = Float("9,000.15")    ;FloatVar = 9000.15
FloatVar = Float("11,0.15")     ;FloatVar = 110.15
;
Test = "9.15"
FloatVar = Float(Test)          ;result depends on the windows regional settings, in Germany
FloatVar = 915
;
Test = "9,15"
FloatVar = Float(Test)          ;result depends on the windows regional settings, in Germany
FloatVar = 9,15
```

Format

Enables you to combine strings and at the same time extract a specific substring of a string. You can format the result to a specific length and fill it with the appropriate characters.

Format

SYNTAX Format(startPosition, numberCharacters, resultLength, formatRule, fillCharacter,formatParams)

ARGUMENTS

startPosition	Startposition beginning from 1.
numberCharacters	Number of characters from the start position to cut off.
resultLength	Maximum length of the result string. 0 means the result string has no length restriction.
formatRule	L = left aligned R = right aligned Z = centered
fillCharacter	Filling character (fill character) if the result string does not have the required length.
formatParams	Data strings to link together.

RETURN Formatted text.

EXAMPLE

```
Date = "1.7.2014"
Day = Format(3, 1, 2, "R", "0",Date) ;   Day   ==> "07"
Month = Format(1, 1, 2, "R", "0",Date) ; Month   ==> "01"
Year = Format(5, 4, 4, "R", "0",Date) ;  Year   ==> "2014"
;---------------------------------------------------------------------
Name = "John James Miller"
; subst without padding
Name1 = Format(6, 50, 0, "L", "_", Name); Name1 = "James Miller"
Name2 = Format( 1, 10, 0, "L", "_", Name); Name2 = "John James"
Name3 = Format(6, 5, 0, "L", "_", Name); Name3 = "James"
; subst with padding, text left aligned
Name4 = Format(6, 50, 20, "L", "_", Name); Name4 = "James Miller_________"
; subst with padding, text right aligned
```

	Name5 = Format(6, 50, 20, "R", "_", Name); Name5 = "_____James Miller" ; subst with padding, text centered Name6 = Format(6, 50, 20, "Z", "_", Name); Name6 = "____James Miller____"

FormatStr

Formats a string by concatenating the strings variable1 to variableN according to the defined format rule.

If the result should contain a '%' character, you must define it as '%%' in the formatRule parameter.

FormatStr		
SYNTAX	FormatStr(formatRule,formatParams: Variable1, Variable2, ..., VariableN)	
ARGUMENTS	formatRule	Format rule.
	formatParams	Strings to format.
RETURN	Formatted text.	
EXAMPLE	d = "01" m = "02" dm = FormatStr("Day: %s, Month: %s", d, m) ;Result: dm = "Day: 01, Month: 02" ; y ="2014" Date = FormatStr("%s/%s/%s", m, d, y) ;Result Date = "02.01.2014" ; Percent sign in the result string: Percentage = "5" Res = FormatStr("Percentage: %s %% p.a.", Percentage) ; Res = Percentage: 5 % p.a.	

FreeObject

Frees up a run time OLE object instance.

FreeObject		
SYNTAX	FreeObject(varName)	
ARGUMENTS	varName	Name of the instance.
EXAMPLE	FreeObject("Reg")	

GetBlobField

Reads the content of a BLOB field.

GetBlobField		
SYNTAX	GetBlobField(datasetName,fieldName)	
ARGUMENTS	datasetName	Name of the dataset
	fieldName	Field name.
RETURN	The content of the BLOB field.	
EXAMPLE	;Read BLOB field "Data" using the function GetBlobField() and save to file System.IO.File.WriteAllBytes("D:\Temp\SystemRequirements_modus_Version_5_DB2.rtf", GetBlobField("dataset", "Data"))	

GetConnection

Returns the database connection object currently in use.

GetConnection	
SYNTAX	GetConnection(varName,aliasName)

ARGUMENTS	varName	Name of the variable in which the database connection object is stored.
	aliasName	Name of the database alias.
EXAMPLE	InitDatabase("DataDbAlias") GetConnection("con", "DataDbAlias") con_s = con.ConnectionString	

GetField

Reads a field value.

GetDirectories

SYNTAX	GetField(datasetName, fieldName) GetField(datasetName, fieldName, destList, dataType)	
ARGUMENTS	datasetName	Name of the dataset.
	fieldName	Field name.
	destList	Name of the target string list for the read value if the data type is "L".
	dataType	Data type. S = String I = Integer F = Float L = String list
RETURN	Field value of the type String, Integer or Float, depending on the given data type.	
EXAMPLE	VN = GetField("DSName","Forename") VN = GetField("DSName","Forename", SListe, "L")	

GetLookupValue

Reads values in a lookup table.

GetLookupValue

SYNTAX GetLookupValue(objectName,key)

ARGUMENTS

objectName	Name of the lookup table.
key	Lookup key.

RETURN Lookup value.

EXAMPLE

```
VarContent="001"
res = GetLookupValue("LookupTab", VarContent)
```

Error handling in case lookup table or key does not exist

```
try
   TestVariable = GetLookupValue("TestLookUp", "LookUpKey")
OnError("LookupValueNotFound")
   ;Either the key within the lookup table or the lookup table itself was not found, therefore the variable is set to a default value
   TestVariable = "DefaultValue"
End-try
```

GetObject

Creates an instance of the specified object.

GetObject

SYNTAX GetObject(varName, className)

ARGUMENTS	varName	Name of the instance.
	className	Name of the DLL or class.
EXAMPLE	`GetObject("date","MLDate")`	
	`GetObject("Reg", "MSLReg_Ini_Lib.MSLRegistry")`	

GetVariable

Checks whether the specified variable exists and if so, returns its value. If the variable does not exist, no further action is taken and no error message results.

GetVariable		
SYNTAX	GetVariable(varName)	
ARGUMENTS	columnName	Name of the variable.
RETURN	Value of the variable.	
EXAMPLE	Forename = GetVariable("ModusUser.Forename")	

GetSystemOId

Returns the SystemOID of the system in which the script executes.

GetSystemOId	
SYNTAX	GetSystemOId()
RETURN	SystemOID
EXAMPLE	MyOID = GetSystemOID()

Index

Creates an index on the dataset.

Index		
SYNTAX	Index(datasetName,idx)	
ARGUMENTS	datasetName	Name of the dataset
	idx	Comma separated list of fields used in the index.
EXAMPLE	Index("DSName", "ACC_Account_Nr,ACC_BIC"	

InitDatabase

Creates a connection to a database.

InitDatabase		
SYNTAX	InitDatabase(aliasName)	
ARGUMENTS	aliasName	Name of the database alias.
RETURN	TRUE = Connection established. FALSE = Connection not established.	
EXAMPLE	OK = InitDatabase("odinTest")	

InitDataset

Creates a new dataset for the specified instance.

InitDataset	
SYNTAX	InitDataset(datasetName, aliasName)

	InitDataset(datasetName, aliasName, tableName)
ARGUMENTS	datasetName — Name of the dataset. aliasName — Name of the database alias. tableName — Name of the database table. If the parameter is set, the table is read using **Select *.**
EXAMPLE	InitDataSet("modusorder, ORDERDATA) InitDataSet("modusorder, ORDERDATA,

Insert

Inserts a dataset and fills fields with values.

Insert	
SYNTAX	InitDataset(datasetName, aliasName, tableName)
ARGUMENTS	datasetName — Name of the dataset. fields — Field values in the correct order.
EXAMPLE	Insert("DSName","Hyland Software","3","11.11.2011")

Integer

Converts a string to an integer variable. Variable content must be a number format.

Integer	
SYNTAX	Integer(valueToConvert)

ARGUMENTS	valueToConvert	The variable to convert. Defining an empty string as parameter value results in an exception at run time.
RETURN	Integer value of the variable.	
EXAMPLE	Var = "10" Var = Integer(Var)	

Last

Positions the dataset cursor at the last record of the dataset.

Last

SYNTAX	Last(datasetName)	
ARGUMENTS	datasetName	Name of the dataset.
EXAMPLE	Last("DSName")	

Max

Returns the highest value found in a string list or list of variables.

Max

SYNTAX	Max(stringList) Max(values)	
ARGUMENTS	stringList	Name of the string list.
	values	Values to check.
RETURN	Highest value.	

EXAMPLE	MaxVarL=Max(StrListe) v1=10 v2=30 v3=50 MaxVar=Max(v1,v2,v3)

Min

Returns the lowest value found in a string list or list of variables.

Min	
SYNTAX	Min(stringList) Min(values)
ARGUMENTS	stringList — Name of the string list. values — Values to check.
RETURN	Lowest value.
EXAMPLE	MinVarL=Min(StrListe) v1=10 v2=30 v3=50 MinVar=Min(v1,v2,v3)

Move

Moves the dataset cursor n steps on from the current position.

Move

SYNTAX	`Move(datasetName, relativeIndex)`
ARGUMENTS	datasetName — Name of the dataset. relativeIndex — Relative position from the current position.
RETURN	Lowest value.
EXAMPLE	Move("DSName",3) Move("DSName",-3)

Next

Positions the dataset cursor on the next record in the dataset.

`Next`

SYNTAX	Next(datasetName)
ARGUMENTS	datasetName — Name of the dataset.
EXAMPLE	Next("DSName")

OpenSQL

Executes the statement defined under SetSQL.

OpenSQL

SYNTAX	OpenSQL(datasetName)
ARGUMENTS	datasetName — Name of the dataset.

RETURN	TRUE = Successful FALSE = Not successful.
EXAMPLE	OpenSQL("DSName")

Parse

Allows you to process texts with special functions, for example, date, time, calendar days, or hex values.

Parse		
SYNTAX	Parse(formattedContent)	
ARGUMENTS	FormattedContent	The string to parse.
RETURN	Formatted text.	

Special function ^D[mm,dd,yy,yyyy] to insert the date

Arguments	Definition
mm	Current month
dd	Current day
yy	Current year (two figures)
yyyy	Current year (four figures)

Example

Test = Parse("^Ddd.mm.yyyy;")

;-->Test = "05.08.2011"

;

Test = Parse("Current date: ^Ddd.mm.yy;")

;-->Test = "Current date: 05.08.08 "

Special function ^U[hh,mm,ss] to insert the time

Arguments	Definition
HH	Current hour, 24 hour format
hh	Current hour, 12 hour format
mm	Current minute
ss	Current second

Example

Test = Parse("^Uhh:mm:ss;")

;-->Test = "10:23:55"

;

Test = Parse("Date: ^Ddd.mm.yy; Time: ^UHH:mm:ss;")

;--Test = "Date: 21.04.09 Time: 19:19:48"

Special function ^K to insert a calendar week number

Example

Test = Parse("^K;")

;-->Test = "32"

Special function ^H[Hexvalue] to insert hexadecimal values

Argument

Hexadecimal value

Example

Test = Parse("^H3F;")

;-->Test = "?"

PrepareSQL

Fills a query with parameters.

PrepareSQL	
SYNTA	PrepareSQL(datasetName, paramDescls)

X

ARGUMENTS

datasetName	Name of the dataset.
paramDescls	Name of the string list containing the description of the SQL parameters. The parameters used in the SQL statement are described in this string list. You need to specify the parameter descriptions in the same order as in the SQL statement. Each parameter must exist in the form ParameterName=ParameterType. A corresponding local MonaLisa variable with the value of the parameter must exist for each ParameterNamedefined. For the parameter type `b` (binary), the variable must contain path and file name of the data to insert. The parameter type consists of two characters. The first character determines the direction of the parameter. The following characters are supported.

Parameter	Definition
b	Bi-directional (input and output parameter)
i	Input parameter
o	Output parameter
r	Return value (stored procedure)

If an unsupported character is defined, the parameter `i` is used.

The second character determines the format of the parameter.

The following characters are supported.

Parameter	Definition
b	Binary (BLOB field)
d	Date
f	Float
i	Integer
s	String
t	Time
@	Date time

If an unsupported character is defined, the parameter `s` is used.

EXAMPLE

```
StringListClear ("ParamDescList)
StringListSetValue("ParamDescList, MyParam, is)
SetSQL(myDataSet, SQL)
PrepareSQL(myDataSet, "ParamDescList" )
```

Prev

Positions the dataset cursor on the previous record in the dataset.

Prev

SYNTAX `Prev(datasetName)`

ARGUMENTS

datasetName	Name of the dataset.

EXAMPLE `Prev("DSName")`

Protocol

Writes a message into the log file of the Content Composer application that executes this MonaLisa script.

Example

If Studio executes the MonaLisa script, the message is written into the log file **Composer.Studio_Log.txt**.

You can find this log file in the subdirectory **ContentComposer** in the user profile directory of the Windows user account that executes the Studio application.

Protocol		
SYNTAX	Protocol(traceMessage, traceLevel, formatParams)	
ARGUMENTS	traceMessage	Trace message.
	traceLevel	Trace level. **Possible values** 0 = Error 1-4 = Warning 5-9 = Information 10 = Verbose
	formatParams	Parameters set for each placeholder. Placeholders are {0},{1}, ...
EXAMPLE	ErrorNumber = "001" Protocol("Error message {0} ", 0, ErrorNumber)	

PutField

Inserts a value into a field.

PutField	
SYNTAX	PutField(datasetName, fieldName, fieldValue)

ARGUMENTS	datasetName	Dataset name.
	fieldName	Field name.
	fieldValue	New content of the field.
EXAMPLE	PutField("DSName","CompanyName,Hyland Software)	

Raise

Triggers an error and then skips to the error label defined in OnError.

Raise		
SYNTAX	Raise(errorTypeToken, errorMessage)	
ARGUMENTS	errorTypeToken	The value passed on to the LastError variable when the Raise function executes.
	errorMessage	Detailed error message.
EXAMPLE	Raise("ERR1","Error 1 thrown")	

Read

Reads a complete dataset and returns the result.

Read		
SYNTAX	Read(datasetName, fieldList, destList, delimiter, quote)	
ARGUMENTS	datasetName	Name of the dataset.
	fieldList	Name of the string list containing the column names to read.

	If the string list is empty, all fields are read. The string list must exist.
destList	Name of the string list in which the results store. If the specified string list does not exist, it is created. If an empty string is passed, only the result string returns.
delimiter	Contains the separator for field contents. A comma is used for an empty string.
quote	Defines the character that encloses the result value. If you define an empty string, none of the individual result values are enclosed.

RETURN Data fields read in a string.

EXAMPLE

```
InitDatabase(DataDbAlias)
InitDataSet("testSet","DataDbAlias","dbo.Accounts")
;
; fieldList Alternative 1: fieldList = Empty List --> all column values are in the result
StringListClear("fieldList") ; clears the string list or creates a new one if not found
;
dummy_s = Read("testSet", "fieldList", "", ",", "") ;no destList defined --> only result string
Protocol("fieldList Alternative 1: fieldList = Empty List; Set1",10,"")
Protocol(dummy_s, 10, "") ; Expected result:
0815,1,69924420,60050000,H,MainAccount,1,23
;
Next("testSet")
;
dummy_s = Read("testSet", "fieldList", "", ",", "") ;no destList defined --> only result string
Protocol("fieldList Alternative 1: fieldList = Empty List; Set2",10,"")
Protocol(dummy_s, 10, "") ; Expected result:
0815,2,55378924,62045000,H,SecondAccount,1,23
;
First("testSet") ;move to first record of dataset
```

```
;
;fieldList Alternative 2: fieldList contains the columns wanted in the result
StringListAdd("fieldList", "KTO_Par_Num")
StringListAdd("fieldList", "KTO_Type")
StringListAdd("fieldList", "KTO_Balance")
dummy_s = Read("testSet", "fieldList", "", ",", "") ;no destList defined --> only result string
Protocol("fieldList Alternative 2: fieldList contains the columns wanted in the result; Set1",20,"")
Protocol(dummy_s, 10, "") ; Expected result: 0815,H,1,23
;
Next("testSet")
;
dummy_s = Read("testSet", "fieldList", "", ",", "") ;no destList defined --> only result string
Protocol("fieldList Alternative 2: fieldList contains the columns wanted in the result; Set2",20,"")
Protocol(dummy_s, 10, "") ; Expected result: 0815,N,24,34
;
First("testSet") ;move to first record of dataset
;
;destList Alternative 1: only destList-Name defined --> List must be created automatically
dummy_s = Read("testSet", "fieldList", "destList", ",", "")
Protocol("destList Alternative 1: only destList-Name defined; Returnstring",10,"")
Protocol(dummy_s, 10, "") ; Expected result: 0815,H,1,23
Protocol("destList Alternative 1: only destList-Name defined; StringList 1.Zeile",10,"")
Protocol(StringListLine("destList", 0), 10, "")
;
Next("testSet")
;destList Alternative 2: destList already exists with entry --> List must be automatically appended
dummy_s = Read("testSet", "fieldList", "destList", ",", "")
destListString = String("destList")
```

```
Protocol("destList Alternative 2: extend existing destList; Returnstring",10,"")
Protocol(dummy_s, 10, "") ; Expected result: 0815,N,24,34
Protocol("destList Alternative 2: extend existing destList; StringList 1.Row",10,"")
Protocol(StringListLine("destList", 0), 10, "")
Protocol("destList Alternative 2: extend existing destList; StringList 2.Row",10,"")
Protocol(StringListLine("destList", 1), 10, "")
```

RegisterLib

Registers a function library (DLL) for the system.

RegisterLib		
SYNTAX	RegisterLib(dllFileName)	
ARGUMENTS	dllFileName	Name and path to the DLL.
EXAMPLE	RegisterLib("C:\Winnt\System\MSLDB_LIB.DLL")	

ReRaise

Use this function inside an OnError block to pass the exception back to the calling context.

ReRaise	
SYNTAX	ReRaise()

Rollback

Discards all modify, insert, and delete actions of the current transaction and closes the transaction.

Rollback

SYNTAX	Rollback(datasetName)
ARGUMENTS	datasetName Name of the dataset.
EXAMPLE	Rollback("DSName")

Round

This function rounds a floating point number to an integer value. If the floating number belongs to the number range of 2147483647 to -2147483648, you must type cast the return value to integer. If the floating number belongs to the number range of 2147483648 to 9223372036854775807 and -2147483649 to -9223372036854775808, you must type cast the return value to big integer.

Round	
SYNTAX	Round(double)
ARGUMENTS	double Floating point number.
RETURN	Rounded value.
EXAMPLE	v1=10.25 i=Integer(Round(v1)) ;--> i=10 y=922337203685477580.001 Round(y)=922337203685477580

Search

Looks for a dataset corresponding to the criteria defined in the parameter.

Search	
SYNTAX	Search(datasetName, expression)

ARGUMENTS	datasetName	Name of the dataset
	expression	Field contents of the current index
RETURN	TRUE = Dataset found FALSE = No dataset found	
EXAMPLE	found = Search("odin","ODJ_ID=3899")	

SelFirstRecord

Positions the cursor on the first record of a data object.

SelFirstRecord		
SYNTAX	SelFirstRecord(nodeName)	
ARGUMENTS	nodeName	Node in the data object **Example** "VSH.RI.ADR"
EXAMPLE	SelFirstRecord("VSH.RI.ADR")	

SelNextRecord

Positions the cursor one record further in a data object.

SelNextRecord		
SYNTAX	SelNextRecord(nodeName)	
ARGUMENTS	nodeName	Node in the data object **Example**

	"VSH.RI.ADR"
EXAMPLE	SelNextRecord("VSH.RI.ADR")

SelRecordCount

Returns the number of records in a data object.

SelRecordCount		
SYNTAX	SelRecordCount(nodeName)	
ARGUMENTS	nodeName	Node in the data object **Example** "VSH.RI.ADR"
RETURN	Number of datasets	
EXAMPLE	NumberOfRecords=SelRecordCount ("PARTNER.ACCOUNTS")	

SelRecordNumber

Returns the current record number of a data object.

SelRecordNumber		
SYNTAX	SelRecordNumber(nodeName)	
ARGUMENTS	nodeName	Node in the data object **Example** "VSH.RI.ADR"
RETURN	Record number of the data object	
EXAMPLE	RecordNumber=SelRecordNumber ("PARTNER.ACCOUNTS")	

SelVariable

Determines the variable value in a data object of the current selection.

SelVariable

SYNTAX

SelVariable(dataObject,dataAttribute)

SelVariable(dataObject,dataAttribute,defaultValue)

ARGUMENTS

dataObject	Complete path to the data object
dataAttribute	Attribute name
defaultValue	Default return value if the data object or attribute does not exist

RETURN

Variable value as a string

EXAMPLE

If the data object has a parent data object, as in the following example, the complete path must be specified.

STR = SelVariable("ADDRESS","STREET,"")

ExRate = SelVariable("PARTNER.CONTRACT.CURRENCY_CODE", "CUR_EXCHANGE_RATE", "")

SelVariableEx

Determines the variable value in a data object of the current selection.

SelVariableEx

SYNTAX

SelVariableEx(dataObject,dataAttribute)

SelVariableEx(dataObject,dataAttribute,defaultValue)

ARGUMENTS	dataObject — Complete path to the data object dataAttribute — Attribute name defaultValue — Default return value if the data object or attribute does not exist
RETURN	To determine the return data type, the following checks perform. The first check that applies determines the type. • If the value read is a valid integer, the result is a MonaLisa variable of the type Integer. • If the value read is a valid floating-point number, the result is a MonaLisa variable of the type Float. • Otherwise, the result is a MonaLisa variable of the type String.
EXAMPLE	If the data object has a parent data object, as in the following example, the complete path must be specified. STR = SelVariableEx("ADDRESS","STREET,"") ExRate = SelVariableEx("PARTNER.CONTRACT.CURRENCY_CODE", "CUR_EXCHANGE_RATE", "")

SelVariableXPath

This function selects an attribute value from the specified XML data object using the `XPath` expression and converts the returned value to the MonaLisa variable. When there is no parameter as `defaultValue` and the `XPath` expression does not select a node, the function returns an empty string.

SelVariableXPath	
SYNTAX	SelVariableXPath(xpath) SelVariableXPath(xpath, defaultValue)

ARGUMENTS	xpath	The xpath expression to execute
	defaultValue	The value to return if no data exists
RETURN	An instance of the MonaLisa variable	
EXAMPLE	`postbox = SelVariableXPath("concat('PostBox number is: ',/ORDER/CUSTOMER/@Zipcode)")` `desc = SelVariableXPath("/ORDER/ORDER_LINE[2]/@Description","Description not found.")`	

SelVariableXsDateTime

Returns a Long format or a date or time value of an XML attribute that has a date or time format of Coordinated Universal Time (UTC). You can further process or convert this value using the MonaLisa class MLDate.

SelVariableXsDateTime

SYNTAX

```
SelVariableXsDateTime(dataObject, attributeName, convertToLocalTime)
SelVariableXsDateTime(dataObject, attributeName, convertToLocalTime, defaultValue)
```

ARGUMENTS

dataObject	The path to the data object, such as the node name **Example** `"VSH.RI.ADR"`
attributeName	Attribute name **Example** `"CODE"`
convertToLocalTime	TRUE = Convert the date and time value to local time. FALSE = Do not convert the date and time value to local time.
defaultValue	The value to return if no data exists

RETURN The date and time expressed in Long format or local time format.

EXAMPLE `datetimeint = SelVariableXsDateTime("ORDER","DateTime",false,"0001-01-`

```
01T01:01:01Z")
GetObject("date","MLDate")
DateIntToStr = date.Int64ToDateString("MM.dd.yyyy", datetimeint)
days = date.DayOfWeek(datetimeint)
```

SelVariableXsFloat

Selects an attribute value from the specified XML data object and converts the returned value to MonaLisa Double variable.

SelVariableXsFloat

SYNTAX

```
SelVariableXsFloat(dataObject, attributeName)
SelVariableXsFloat(dataObject, attributeName, defaultValue)
```

ARGUMENTS

dataObject	The path to the data object, such as the node name **Example** `"VSH.RI.ADR"`
attributeName	Attribute name **Example** `"CODE"`
defaultValue	The value to return if no data exists

RETURN

The MonaLisa variable of type MDouble.

EXAMPLE

```
totalamount = SelVariableXsFloat("ORDER", "Total_amount")
```

Copyright<?rh-symbol_start name="Copyright" ?>©<?rh-symbol_end ?> 2023 Hyland Software, Inc. and its affiliates.

SetExecSQL

Show All

Executes an SQL command without a result.

If the SQL statement contains parameters (%s or :Param), it only executes if these parameters can be resolved or are recognized. This is realized by means of the paramsDescls list.

If the list is already passed with SetExecSQL, the function executes. If not, the parameters must be resolved or made available by calling PrepareSQL and passing the list. Following that, the ExecSQL function must be executed.

SetExecSQL

SYNTAX

SetExecSQL(datasetName, sqlList)

SetExecSQL(datasetName, sqlList, paramDescls)

ARGUMENTS

datasetName	Name of the dataset
sqlList	Name of the string list containing the SQL statement
paramDescls	Name of the string list containing the description of the SQL parameters The parameters used in the SQL statement are described in this string list You need to specify the parameter descriptions in the same order as in the SQL statement. Each parameter must exist in the form ParameterName=ParameterType. A corresponding local MonaLisa variable with the value of the parameter must exist for each ParameterNamedefined. For the parameter type `b` (binary), the variable must contain path and file name of the data to insert. The parameter type consists of two characters. The first character determines the direction of the parameter. The following characters are supported.

Parameter	Definition

b	Bi-directional (input and output parameter)
i	Input parameter
o	Output parameter
r	Return value (stored procedure)

If an unsupported character is defined, the parameter `i` is used.

The second character determines the format of the parameter.

The following characters are supported.

Parameter	Definition
b	Binary (BLOB field)
d	Date
f	Float
i	Integer
s	String
t	Time
@	Date time

If an unsupported character is defined, the parameter `s` is used.

EXAMPLE

Example

```
InitDatabase("DataDbAlias")
InitDataSet("testSet","DataDbAlias")
#region Testpreparation
;first delete any existing test datasets
StringListClear("SqlDefList")
```

```
StringListAdd("SqlDefList", "Delete from dbo.ACCOUNTS")
StringListAdd("SqlDefList", "where (ACC_Par_Num = '6666') ")
StringListAdd("SqlDefList", "OR (ACC_Par_Num = '9999') ")
SetExecSQL("testSet", "SqlDefList")
;
#region create Testddataset1
StringListClear("SqlDefList")
StringListAdd("SqlDefList", "INSERT INTO dbo.Accounts")
StringListAdd("SqlDefList", "(ACC_Par_Num, ")
StringListAdd("SqlDefList", "ACC_Num, ")
StringListAdd("SqlDefList", "ACC_Acc_Num, ")
StringListAdd("SqlDefList", "ACC_BIC, ")
StringListAdd("SqlDefList", "ACC_Type, ")
StringListAdd("SqlDefList", "ACC_Label, ")
StringListAdd("SqlDefList", "ACC_Balance)")
StringListAdd("SqlDefList", "VALUES")
StringListAdd("SqlDefList", "(6666, ")
StringListAdd("SqlDefList", "1, ")
StringListAdd("SqlDefList", "066066066, ")
StringListAdd("SqlDefList", "62093210, ")
StringListAdd("SqlDefList", "H, ")
StringListAdd("SqlDefList", "'MyAccount', ")
StringListAdd("SqlDefList", "66.66)")
SetExecSQL("testSet", "SqlDefList")
#endregion
;
#region create Testddataset2
StringListClear("SqlDefList")
StringListAdd("SqlDefList", "INSERT INTO dbo.Accounts")
StringListAdd("SqlDefList", "(ACC_Par_Num, ")
```

```
StringListAdd("SqlDefList", "ACC_Num, ")
StringListAdd("SqlDefList", "ACC_Acc_Num, ")
StringListAdd("SqlDefList", "ACC_BIC, ")
StringListAdd("SqlDefList", "ACC_Type, ")
StringListAdd("SqlDefList", "ACC_Label, ")
StringListAdd("SqlDefList", "ACC_Balance)")
StringListAdd("SqlDefList", "VALUES")
StringListAdd("SqlDefList", "(9999, ")
StringListAdd("SqlDefList", "1, ")
StringListAdd("SqlDefList", "099099099, ")
StringListAdd("SqlDefList", "62999999, ")
StringListAdd("SqlDefList", "H, ")
StringListAdd("SqlDefList", "'MyAccount2', ")
StringListAdd("SqlDefList", "99.99)")
SetExecSQL("testSet", "SqlDefList")
#endregion
#endregion
;
;---------------------------------


;---------------------------------
StringListClear("SqlDefList")
StringListAdd("SqlDefList", "Delete from dbo.Accounts")
StringListAdd("SqlDefList", "where (ACC_Par_Num = %s) ")
StringListAdd("SqlDefList", "AND (ACC_Type = :MyType) ")
;Store parameter descriptions in a string list
StringListClear("ParamDescList")
StringListSetValue("ParamDescList", "MyVar", "is") ; Alternative to StringListAdd
("ParamDescList", "MyVar=is")
StringListSetValue
("ParamDescList", "MyType", "") ; Empty string as Value: by default takes is
```

```
;
;-----------------------------------------------------------------------------
;Alternative1: Call SetExecSQL passing the paramsDescList: Delete test dataset1
;-----------------------------------------------------------------------------
MyVar = "6666"
MyTyp ="H"
SetExecSQL("testSet", "SqlDefList", "ParamDescList")
;Store SqlStatement in a string list
StringListClear("SqlDefList")
StringListAdd("SqlDefList", "Delete from dbo.Accounts")
StringListAdd("SqlDefList", "where (ACC_Type = :MyTyp) ")
StringListAdd("SqlDefList", "AND (ACC_Par_Num = %s) ")
;
;Store parameter descriptions in a string list
StringListClear("ParamDescList")
StringListSetValue
("ParamDescList", "MyType", "") ; Order of parameters must be identical to the order
StringListSetValue("ParamDescList", "MyVar", "is") ; in the SQL statement!
;-----------------------------------------------------------------------------------------------------------
;Alternative2: Alternative ExecSQL call
For this, SetExecSQL without passing the paramsList
(or SetSql) and PrepareSQL calls must already have been completed:
Delete test dataset2
;-----------------------------------------------------------------------------------------------------------
MyVar="9999"
SetExecSQL("testSet", "SqlDefList")
PrepareSQL("testSet", "ParamDescList")
ExecSQL("testSet")
Example: Insert RTF files in a BLOB field
InitDatabase("Odin")
InitDataSet("dataSet","Odin")
```

```
StringListClear("SqlDefList")
StringListAdd("SqlDefList", "Insert into dbo.Data (AutoIndex, Datatype, Data) Values
(:AutoIndex,:Datatype,:Data)")

StringListClear("ParamDescList")
StringListSetValue("ParamDescList", "AutoIndex", "ii") ; integer
StringListSetValue("ParamDescList", "Datatype", "ii") ; integer
StringListSetValue("ParamDescList", "Data", "ib") ; binary

AutoIndex = 4711
Datatype = 1
Data = "D:\Temp\SystemRequirements.rtf"
SetExecSQL("dataSet", "SqlDefList", "ParamDescList")
Example: Read BLOB field and save to file (string list)
InitDatabase("Odin")
InitDataSet("dataset","Odin")

StringListClear("SqlDefList")
StringListAdd("SqlDefList", "Select Data from Data where Autoindex = 4711") SetSQL
("dataset", "SqlDefList")

;read BLOB field "Data" and save to file
GetField("dataset", "Data", "D:\Temp\SystemRequirements.rtf", "")

; read blob-field "Data" into stringlist "test"
GetField("dataset", "Data", "test", "L")

;read BLOB field "Data" using the method GetBlobField() and save to file
System.IO.File.WriteAllBytes("D:\Temp\SystemRequirements.rtf", GetBlobField
("dataset", "Data"))
```

SetPoolVariable

Sets the value of a pool variable.

If the variable does not exist, it is created. If the variable already exists, it is overwritten.

SetPoolVariable

SYNTAX SetPoolVariable(poolName, varName, value)

ARGUMENTS

poolName	Pool name
varName	Variable name
value	The value to set

EXAMPLE

```
VarName = "Var_"
i = 10
while (i < 10)
  PoolVarName = VarName + i
  SetPoolVariable("System", PoolVarName, i)
  i = i + 1
end-while
```

SetSQL

Executes an SQL command.

SetSQL

SYNTAX SetSQL(datasetName,sqlList)

ARGUMENTS

datasetName	Dataset name
sqlList	Name of the string list containing the SQL definition

EXAMPLE StringlistAdd("SQL","Delete from dfd.db")

```
InitDataset("Tabelle","AllSQL")
SetSQL("Tabelle", "SQL")
```

Sleep

Pauses script execution for the specified period of time.

Sleep		
SYNTAX	Sleep(sleepTime)	
ARGUMENTS	sleepTime	Pause in milliseconds
EXAMPLE	Sleep(1000)	

SplitStr

Obtains a specific partial expression from a string.

SetSQL		
SYNTAX	SetValue(columnName, newValue)	
ARGUMENTS	data	String
	splitedStr	Number of the partial expression beginning from 1
	seperator	Delimiter for individual partial expressions
	errorText	Result string returned when a partial expression does not exist
RETURN	Partial expression of the string	
EXAMPLE	dbvar = "T06M07J08" s1 = SplitStr(dbvar, 1, "TMJX", "") ;s1 = "06"	

```
;
s2 = SplitStr(dbvar, 2, "TMJX", "")
;s2 = "07"
;
s3 = SplitStr(dbvar, 3, "TMJX", "")
;s3 = "08"
;
s4 = SplitStr(dbvar, 4, "TMJX", "-")
;s4 = "-"
```

StartTransaction

Starts a new transaction.

StartTransaction		
SYNTAX	StartTransaction(datasetName)	
ARGUMENTS	datasetName	Name of the dataset
EXAMPLE	StartTransaction("DSName")	

StartsWith

Determines if the beginning of the input string matches the specified string.

StartsWith		
SYNTAX	StartsWith(inString, value, ignoreCase)	
ARGUMENTS	inString	The input string to check

	value	The string value to compare with the beginning of the `inString` value
	ignoreCase	TRUE = Case is ignored during the search. FALSE = Case is included in the search.
RETURN	TRUE = A string match is found. FALSE = No string match found.	
EXAMPLE	retPosition = StartsWith("ABCxyz","ABC",false)	

StoredProcedure

Executes a stored procedure.

StoredProcedure

SYNTAX	StoredProcedure(aliasName, procName) StoredProcedure(aliasName, procName, paramDescls)	
ARGUMENTS	aliasName	Name of the database alias
	procName	Name of the stored procedure
	paramDescls	Parameter for the stored procedure in the right order. For more information, refer to SetExecSQL.
EXAMPLE	StoredProcedure("DSName","ProcedureName","John","Miller")	

StrCompare

This function compares two specified string objects, ignoring or honoring their case, and returns an integer that indicates their relative position in the sort order.

StrCompare

SYNTAX	StrCompare(strA, strB, ignorecase)
ARGUMENTS	strA — The first string to compare strB — The second string to compare ignoreCase — TRUE = Case is ignored during the search. FALSE = Case is included in the search.
RETURN	A 32-bit signed integer that indicates the lexical relationship between the two comparands. < 0 = strA is less than strB. = 0 = strA is equal to strB. > 0 = strA is greater than strB.

EXAMPLE	;Compare
	retvalue = StrCompare("String to compare.","String to compare.",true)
	if(retvalue <> 0)
	  Raise("Error", "StrCompare() The result should be 0.")
	end-if
	
	;Case sensitive test
	retvalue = StrCompare("Cat","cat",false)
	if(retvalue = 0)
	  Raise("Error", "StrCompare() The result should not be 0.")
	end-if

StrDelChar

Deletes a specific character from a string.

StrDelChar	
SYNTAX	StrDelChar(inString,delChar)

ARGUMENTS	inString	String or variable
	delChar	Character to delete
RETURN	Result string	
EXAMPLE	var1="a b c d" var2=StrDelChar(var1, " ") ;var2=abcd	

StrIndexOf

Reports the index, beginning from 1, of the first occurrence of a source string.

StrIndexOf		
SYNTAX	StrIndexOf(searchString, sourceString, startIndex, ignoreCase)	
ARGUMENTS	searchString	The string to search for
	sourceString	The source string beginning from 1
	startIndex	The search starting position
	ignoreCase	TRUE = Case is ignored during the search. FALSE = Case is included in the search.
RETURN	The index position, beginning from 1, of the value parameter if that string exists, or -1 if it does not exist.	
EXAMPLE	retvalue = StrIndexOf("m","animal",2,true) if(retvalue 4) Raise("Error", "StrCompare() The result should be 4 but it is " + retvalue) end-if	

StrLastIndexOf

This function reports the index, beginning from 1, of the last occurrence of a source string.

StrLastIndexOf		
SYNTAX	StrLastIndexOf(searchString, sourceString, ignoreCase)	
ARGUMENTS	searchString	The string to search for.
	sourceString	The source string.
	ignoreCase	TRUE = Case is ignored during the search. FALSE = Case is included in the search.
RETURN	The index position beginning from 1, of the value parameter if the string exists, or -1 if it does not exist.	
EXAMPLE	retvalue = StrLastIndexOf("N","Dot Net Perls",true) if(retvalue <> 5) Raise("Error", "StrLastIndexOf() The result should be 5 but is " + retvalue) end-if	

String

Converts a variable to the string format.

You can also use this function to revert a string list back to a string. The elements of the list are separated by [CRLF].

String		
SYNTAX	String(valueToConvert)	
ARGUMENTS	valueToConvert	The variable to convert (alphanumeric, numeric or a string list)
RETURN	Converted string	

EXAMPLE	i=5 s=String(i) ; StringlistAdd("StrList","Hyland Software","Konrad-Zuse-Str. 9","74172 Neckarsulm") Address=String("StrList")

StringListAdd

Adds strings to a string list.

StrDelChar

SYNTAX	StringListAdd(stringList, stringParameters)

ARGUMENTS

stringList	Name of the string list
stringParameters	Strings to add

EXAMPLE	Street = "Konrad-Zuse-Str. 9" Code = "74172" City = "Neckarsulm" ; StringlistAdd(SListe, Street, Code, City) Address = String(SListe) ;-->Address = "Konrad-Zuse-Str. 9,74172,Neckarsulm"

StringListClear

Removes all elements from a string list. If the specified string list does not exist, it is created.

StringListClear

SYNTAX	StringListClear(stringList)

ARGUMENTS	stringList	Name of the string list
EXAMPLE	StringListClear(SListe)	

StringListCount

Counts the rows in a string list.

StringListCount		
SYNTAX	StringListCount(stringList)	
ARGUMENTS	stringList	Name of the string list
RETURN	Number of rows	
EXAMPLE	i = StringListCount(StrList)	

StringListGetCommaText

Returns all rows of a string list in a comma separated string.

StringListGetCommaText		
SYNTAX	StringListGetCommaText(stringList)	
ARGUMENTS	stringList	Name of the string list
RETURN	Content of the string list	
EXAMPLE	Commatext = StringListGetCommatext(StrList)	

StringListGetText

Returns the text of a string list. Each row ends with CRLF.

StringListGetText		
SYNTAX	StringListGetText(stringList)	
ARGUMENTS	stringList	Name of the string list
RETURN	Content of the string list	
EXAMPLE	SLText = StringListGetText(StrList)	

StringListGetValue

Reads the value of a name within a string list in the form of "Name=Value".

StringListGetValue		
SYNTAX	StringListGetValue(stringList, aName)	
ARGUMENTS	stringList	Name of the string list
	aName	Identifier of the value to read
RETURN	The value assigned to the identifier	
EXAMPLE	SLValue = StringListGetValue(StrList,Name)	

StringListIndexOf

Returns the position of the first element in the line of the string list that contains the specified value.

StringListIndexOf	
SYNTAX	StringListIndexOf(stringList,aLine)

ARGUMENTS	stringList	Name of the string list
	aLine	Search value
RETURN	Position of the element	
EXAMPLE	StringlistAdd(Title, Mr.,Mrs.,Company,Family) Pos = StringListIndexOf ("Title","Company") ;-->Pos=2	

StringListItemValues

Reads the value after the specified index and writes the comma separated text in the target string list.

StringListItemValues

SYNTAX StringListItemValues(stringList, lineIndex,destList)

ARGUMENTS	stringList	Name of the string list
	lineIndex	Line index of the string list beginning with 0
	destList	Name of the target string list

RETURN Value name

EXAMPLE
```
StringlistAdd("test","Valuelist=1,2,3,4,5")
name = StringListItemValues("test",0,"Values")
;--> Name = "Valuelist"
;
value = StringListLine("Values",0)
;--> value = "1 "
;
value = StringListLine("Values",1)
;--> value = "2 "
```

StringListLine

Reads the content of a line in a string list.

StringListLine		
SYNTAX	StringListLine(stringList,lineIndex)	
ARGUMENTS	stringList	Name of the string list
	lineIndex	Number of the line to read beginning from 0
RETURN	Line content	
EXAMPLE	line = StringListLine("SListe",0)	

StringListLoad

Loads a list of strings from a file.

StringListItemValues		
SYNTAX	StringListLoad(stringList,fileName) StringListLoad(stringList,fileName,sectionName)	
ARGUMENTS	stringList	Name of the string list
	fileName	Name of the file from which the list loads
	sectionName	Name of the section from which the data loads. This parameter is optional.
RETURN	Value name.	
EXAMPLE	StringlistLoad("SListe", "c:\Temp\Test.lst") StringlistLoad("SListe", "c:\Temp\Test.lst","Section1")	

StringListParams

Configures the parameters of the string list.

StringListParams		
SYNTAX	StringListParams(stringList, sortList, duplicateState)	
ARGUMENTS	stringList	Name of the string list
	sortList	TRUE = The string list is sorted FALSE = The string list is not sorted
	duplicateState	0 = Ignore - ignore duplicates 1 = Accept - allow duplicates 2 = Error - error message when inserting duplicates. Setting parameters for duplicates has no effect on double strings already included in the list.
EXAMPLE	StringlistAdd("SListe", "Mr.","Mrs.","Company","Family") StringListParams("SListe",true,2) StringlistAdd("SListe", "Company")	

StringListSave

Show All

Saves a list of strings in a file.

StringListItemValues		
SYNTAX	StringListSave(stringList, fileName) StringListSave(stringList, fileName, encoding)	
ARGUMENTS	stringList	Name of the string list

fileName	Name of the file to which the list saves
encoding	You can use the following values as encoding parameters. • Empty string. In this case, the default encoding (ANSI code page of the operating system) is used. • Code page, for example, "1252" • Name, for example, "windows-1252" • Friendly name, for example, "Western European" (Windows)

Possible encoding parameters

Code page	name	Display name
37	IBM037	IBM EBCDIC (USA-Canada)
437	IBM437	OEM USA
500	IBM500	IBM EBCDIC (International)
708	ASMO-708	Arabic (ASMO 708)
720	DOS-720	Arabic (DOS)
737	ibm737	Greek (DOS)
775	ibm775	Baltic (DOS)
850	ibm850	Western European (DOS)
852	ibm852	Eastern European (DOS)
855	IBM855	OEM Cyrillic
857	ibm857	Turkish (DOS)
858	IBM00858	OEM Multilingual Latin I
860	IBM860	Portuguese (DOS)
861	ibm861	Icelandic (DOS)
862	DOS-862	Hebrew (DOS)
863	IBM863	French, Canada (DOS)
864	IBM864	Arabic (864)
865	IBM865	Nordic (DOS)
866	cp866	Cyrillic (DOS)
869	ibm869	Greek, modern (DOS)

870	IBM870	IBM EBCDIC (Multilingual Latin-2)
874	windows-874	Thai (Windows)
875	cp875	IBM EBCDIC (Greek, modern)
932	shift_jis	Japanese (Shift-JIS)
936	gb2312	Simplified Chinese (GB2312)
949	ks_c_5601-1987	Korean
950	big5	Chinese (Traditional) (Big5)
1026	IBM1026	IBM EBCDIC (Turkish, Latin-5)
1047	IBM01047	IBM Latin-1
1140	IBM01140	IBM EBCDIC (USA-Canada-European)
1141	IBM01141	IBM EBCDIC (Germany-European)
1142	IBM01142	IBM EBCDIC (Denmark-Norway-European)
1143	IBM01143	IBM EBCDIC (Finland-Sweden-European)
1144	IBM01144	IBM EBCDIC (Italian-European)
1145	IBM01145	IBM EBCDIC (Spain-European)
1146	IBM01146	IBM EBCDIC (Great Britain-European)
1147	IBM01147	IBM EBCDIC (France-European)
1148	IBM01148	IBM EBCDIC (International-European)
1149	IBM01149	IBM EBCDIC (Icelandic-European)
1200	utf-16	Unicode
1201	unicodeFFFE	Unicode (Big Endian)
1250	windows-1250	Central European (Windows)
1251	windows-1251	Cyrillic (Windows)
1252	Windows-1252	Western European (Windows)
1253	windows-1253	Greek (Windows)
1254	windows-1254	Turkish (Windows)
1255	windows-1255	Hebrew (Windows)
1256	windows-1256	Arabic (Windows)
1257	windows-1257	Baltic (Windows)

1258	windows-1258	Vietnamese (Windows)
1361	Johab	Korean (Johab)
10000	macintosh	Western European (Mac)
10001	x-mac-japanese	Japanese (Mac)
10002	x-mac-chinesetrad	Chinese traditional (Mac)
10003	x-mac-korean	Korean (Mac)
10004	x mac-arabic	Arabic (Mac)
10005	x-mac-hebrew	Hebrew (Mac)
10006	x-mac-greek	Greek (Mac)
10007	x-mac-cyrillic	Cyrillic (Mac)
10008	x-mac-chinesesimp	Chinese simplified (Mac)
10010	x-mac-romanian	Romanian (Mac)
10017	x-mac-ukrainian	Ukrainian (Mac)
10021	x-mac-thai	Thai (Mac)
10029	x-mac-ce	Central European (Mac)
10079	x-mac-icelandic	Icelandic (Mac)
10081	x-mac-turkish	Turkish (Mac)
10082	x-mac-croatian	Croatian (Mac)
12000	utf-32	Unicode (UTF-32)
12001	utf-32BE	Unicode (UTF-32-Big Endian)
20000	x-Chinese-CNS	Chinese traditional (CNS)
20001	x-cp20001	TCA Taiwan
20002	x-Chinese-Eten	Chinese traditional (Eten)
20003	x-cp20003	IBM5550 Taiwan
20004	x-cp20004	TeleText Taiwan
20005	x-cp20005	Wang Taiwan
20105	x-IA5	Western European (IA5)
20106	x-IA5-German	German (IA5)
20107	x-IA5-Swedish	Swedish (IA5)
20108	x-IA5-Norwegian	Norwegian (IA5)
20127	us-ascii	US-ASCII

20261	x-cp20261	T.61
20269	x-cp20269	ISO-6937
20273	IBM273	IBM EBCDIC (Germany)
20277	IBM277	IBM EBCDIC (Denmark-Norway)
20278	IBM278	IBM EBCDIC (Finland-Sweden)
20280	IBM280	IBM EBCDIC (Italy)
20284	IBM284	IBM EBCDIC (Spain)
20285	IBM285	IBM EBCDIC (Great Britain)
20290	IBM290	IBM EBCDIC (Japanese Katakana)
20297	IBM297	IBM EBCDIC (France)
20420	IBM420	IBM EBCDIC (Arabic)
20423	IBM423	IBM EBCDIC (Greek)
20424	IBM424	IBM EBCDIC (Hebrew)
20833	x-EBCDIC-KoreanExtended	IBM EBCDIC (Korean, extended)
20838	IBM-Thai	IBM EBCDIC (Thai)
20866	koi8-r	Cyrillic (KOI8-R)
20871	IBM871	IBM EBCDIC (Icelandic)
20880	IBM880	IBM EBCDIC (Cyrillic, Russian)
20905	IBM905	IBM EBCDIC (Turkish)
20924	IBM00924	IBM Latin-1
20932	EUC-JP	Japanese (JIS 0208-1990 and 0212-1990)
20936	x-cp20936	GB2312-80 Chinese (simple)
20949	x-cp20949	Korean Wansung
21025	cp1025	IBM EBCDIC (Cyrillic, Serbian-Bulgarian)
21866	koi8-u	Cyrillic (KOI8-U)
28591	iso-8859-1	Western European (ISO)
28592	iso-8859-2	Central European (ISO)
28593	iso-8859-3	Latin 3 (ISO)
28594	iso-8859-4	Baltic (ISO)

28595	iso-8859-5	Cyrillic (ISO)
28596	iso-8859-6	Arabic (ISO)
28597	iso-8859-7	Greek (ISO)
28598	iso-8859-8	Hebrew (ISO-Visual)
28599	iso-8859-9	Turkish (ISO)
28603	iso-8859-13	Estonian (ISO)
28605	iso-8859-15	Latin 9 (ISO)
29001	x-Europa	Europe
38598	iso-8859-8-i	Hebrew (ISO-Logical)
50220	iso-2022-jp	Japanese (JIS)
50221	csISO2022JP	Japanese (JIS, 1 Byte Kana allowed)
50222	iso-2022-jp	Japanese (JIS, 1 Byte Kana allowed - SO/SI)
50225	iso-2022-kr	Korean (ISO)
50227	x-cp50227	ISO-2022 Chinese (simplified)
51932	euc-jp	Japanese (EUC)
51936	EUC-CN	Chinese simplified (EUC)
51949	euc-kr	Korean (EUC)
52936	hz-gb-2312	Chinese simplified (HZ)
54936	GB18030	GB18030 Chines simplified
57002	x-iscii-de	ISCII Devanagari
57003	x-iscii-be	ISCII Bengali
57004	x-iscii-ta	ISCII Tamil
57005	x-iscii-te	ISCII Telugu
57006	x-iscii as	ISCII Assamese
57007	x-iscii-or	ISCII Oriya
57008	x-iscii-ka	ISCII Kannada
57009	x-iscii-ma	ISCII Malayalam
57010	x-iscii-gu	ISCII Gujarati
57011	x-iscii-pa	ISCII Punjabi
65000	utf-7	Unicode (UTF-7)
65001	utf-8	Unicode (UTF-8)

EXAMPLE	StringlistSave("SListe", "c:\Temp\Test.lst")

StringListSetCommaText

Writes a comma-separated value in a string list.

StringListSetCommaText

SYNTAX	StringListSetCommaText(stringList, commaText)

ARGUMENTS		
	stringList	Name of the string list
	commaText	Comma separated values to save in the string list, for example, "A, B, C"

EXAMPLE	StringListSetCommatext ("List1", "A,B,C")

StringListSetValue

Sets the value of a name within a string list in the form of: "Name=Value".

StringListSetValue

SYNTAX	StringListSetValue(stringList, aName, aValue)

ARGUMENTS		
	stringList	Name of the string list
	aName	Identifier of the value to set
	aValue	Value to assign to the identifier

EXAMPLE	StringListSetValue ("List1", "Lastname", "Miller")

StringListSort

Sorts the string list.

StringListSort		
SYNTAX	StringListSort(stringList)	
ARGUMENTS	stringList	Name of the string list
EXAMPLE	StringListSort("StrList")	

StrLeft

Left-aligned formatting of a string.

StrLeft		
SYNTAX	StrLeft(inString)	
ARGUMENTS	inString	String or variable
RETURN	Left aligned string	
EXAMPLE	Var = " 1234" VarL = StrLeft(Var) ;--> VarL = "1234 "	

StrLen

Calculates the length of a string.

StrLen	
SYNTAX	StrLen(inString)

ARGUMENTS	inString String or variable
RETURN	String length
EXAMPLE	Var = "1234" Len = StrLen(Var) ;-->Len = 4

StrLower

Converts a string to lower case.

StrLower	
SYNTAX	StrLower(inString)
ARGUMENTS	inString String or variable
RETURN	String in lower case
EXAMPLE	Var = "HYLAND SOFTWARE" VarLow = StrLower(Var) ; VarLow = "hyland software"

StrPos

Finds a string in a string.

StrPos	
SYNTAX	StrPos(searchStr, sourceStr, ignoreCase)

ARGUMENTS	searchStr	String to find
	sourceStr	String to search through
	ignoreCase	TRUE = Case sensitive FALSE = Not case sensitive
RETURN	Position of the word in the string or 0 if the search string does not exist	
EXAMPLE	SourceStr = "Monday Tuesday Wednesday" SuchStr = "Tuesday" res = StrPos(SuchStr, SourceStr, TRUE) ;-->res=8	

StrReplace

Replaces characters in a string with other characters.

StrReplace		
SYNTAX	StrReplace(inString, oldValue, newValue)	
ARGUMENTS	inString	String or variable
	oldValue	Character to replace
	newValue	Character to insert
RETURN	Altered string	
EXAMPLE	**Example - Replace Characters** var1 = "10/11/2011" Var2 = StrReplace(Var1, "/", ".") Var3 = StrReplace(Var1, "/", " - ") ;--> Var2 = "10.11.2011"	

;--> Var3 = "10 - 11 - 2011"

Example - Replace Quotes with Parse

Var1 = """John"","Peter"""

;--> Var1 = "John","Peter"

qoute = Parse("^H22;")

Var2 = StrReplace(Var1, quote, "")

;--> Var2=John,Peter

For more information, refer to **Parse**.

StrReverse

Reverses the string.

StrReverse	
SYNTAX	StrReverse(inString)
ARGUMENTS	inString String or variable
RETURN	Reversed string
EXAMPLE	Var = HYLAND SOFTWARE VarR = ;--> VarR = ERAWTFOS EVITPECREP

StrRight

Formats a string right-aligned.

StrRight	
SYNTAX	StrRight(inString)

ARGUMENTS	inString String or variable.\
RETURN	Right aligned string
EXAMPLE	Var = "1234 " VarR = StrRight(Var) ; VarR = " 1234"

StrTok

Splits a string after a specified delimiter and returns the extracted string in front of the delimiter as the result.

StrTok	
SYNTAX	StrTok(varName,delimiter)
ARGUMENTS	varName String or variable delimiter Separator character
RETURN	Extracted string
EXAMPLE	Variable = "1;2;3;4;" V = StrTok("Variable", ";") ;--> V="1" Variable="2;3;4;" V = StrTok("Variable", ";") ;--> V="2" Variable="3;4;" V = StrTok("Variable", ";") ;--> V="3" Variable="4;" V = StrTok("Variable", ";") ;--> V="4" Variable="" V = StrTok("Variable", ";") ;--> V="" Variable=""

StrToLen

Reduces a string to a specific length.

StrToLen

SYNTAX	StrToLen(inString, maxLen, fillChar, doRight)
ARGUMENTS	inString — String or variable maxLen — Result length If the value of this parameter is less than the length of the string, the string truncates at this position. Otherwise, the string fills to the required length with the defined filling character. fillChar — Filling character If no filling character is defined, the default character is a space. doRight — TRUE = Insert filling characters at the beginning. FALSE = Insert filling characters at the end.
RETURN	Altered string.
EXAMPLE	Var = "1234" Var10 = StrToLen(Var, 10, "0", TRUE) ;--> Var10="0000001234" Var10 = StrToLen(Var, 10, "0", FALSE) ;--> Var10="1234000000"

StrUpper

Converts a string to upper case.

StrUpper	
SYNTAX	StrUpper(inString)
ARGUMENTS	inString — String or variable
RETURN	String in uppercase

EXAMPLE	Var = "hyland software" VarUp = StrUpper(Var) ; VarUp = "HYLAND SOFTWARE"

StrWord

Extracts the part of a string that is separated by the delimiter.

StrWord	
SYNTAX	StrWord(inString, wordNumber, delimiter)
ARGUMENTS	inString — String or variable wordNumber — Word number beginning with 1 delimiter — Separator character
RETURN	Extracted substring.
EXAMPLE	Var = StrWord(W1.w2.W3,2,.) ;--> Var = w2

StrWordCount

Calculates the number of words according to the specified delimiter.

StrWordCount	
SYNTAX	StrWordCount(inString, delimiter)
ARGUMENTS	inString — String or variable delimiter — Separator character

RETURN	Number of words.
EXAMPLE	Var = StrWordCount("W1.w2.W3",".") ;--> Var = 3

SubString

This function retrieves a substring from the given input string. The substring starts at a specified position and has a specified length.

SubString

SYNTAX SubString(inString, startIndex, length)

ARGUMENTS

inString	The string from which a part is extracted
startIndex	The starting position beginning from 1
length	The number of characters in the substring. If you pass the value zero for this parameter, all characters starting from the start Index to the end of the given input string return.

RETURN A string of the given length that begins at the start index of the given input string

EXAMPLE retString = SubString("abc,2,1)

Trim

Removes all leading and trailing white space characters from the input string object.

Trim

SYNTAX Trim(inString)

ARGUMENTS

inString	The input string to trim

RETURN The string that remains after all white space characters are removed from the start and end

	of the given input string
EXAMPLE	retString = Trim(" abc XYZ ") ; Result: retString = "abc XYZ"

TrimEnd

Removes all trailing white space characters from the input string.

TrimEnd	
SYNTAX	TrimEnd(inString)
ARGUMENTS	inString The input string to trim
RETURN	The string that remains after all white space characters are removed from the end of the given input string
EXAMPLE	retString = TrimEnd(" abc XYZ ") ; Result: retString = " abc XYZ"

TrimStart

This function removes all leading white space characters from the input string.

TrimStart	
SYNTAX	TrimStart(inString)
ARGUMENTS	inString The input string to trim
RETURN	The string that remains after all white-space characters are removed from the start of the given input string
EXAMPLE	retString = TrimStart(" abc XYZ ") ; Result: retString = "abc XYZ "

Trunc

This function truncates the decimal places of a floating point number. If the floating number belongs to the number range of 2147483647 to -2147483648, you must type cast the return value to integer. If the floating number belongs to the number range of 2147483648 to 9223372036854775807 and -2147483649 to -9223372036854775808, you must type cast the return value to big integer.

Trunc	
SYNTAX	Trunc(value)
ARGUMENTS	value Floating point number.
RETURN	Integer
EXAMPLE	d = 1.9 i = Integer(Trunc(d)) ;--> i = 1 d = 9999999999.001 i = BigInteger(Trunc(d)) ;--> i = 9999999999

UnRegisterLib

Unregisters a function library (DLL).

UnRegisterLib	
SYNTAX	UnRegisterLib(dllFileName)
ARGUMENTS	dllFileName Name and path to the DLL
EXAMPLE	UnRegisterLib("C:\Winnt\System\MSLDB_LIB.DLL")

Update

Alters a dataset and fills fields with values.

Update		
SYNTAX	Update(datasetName,rowNum,fields)	
ARGUMENTS	datasetName	Name of the dataset
	rowNum	Row number after which data inserts
	fields	Field values in the correct order
EXAMPLE	Update("DSName",1,"Hyland Software","3","08/10/2013")	

VerifyPool

Checks if the specified pool exists and throws an exception if not.

VerifyPool		
SYNTAX	VerifyPool(poolName)	
ARGUMENTS	dllFileName	Name of the pool to check
EXAMPLE	VerifyPool("MyPool")	

Version

Determines the version of the MonaLisa interpreter.

Version	
SYNTAX	Version()
RETURN	Version number
EXAMPLE	MLVersion=Version()

Sample Script: Database Access

This script executes a select statement on a database and iterates through the results.

```
myDBALias = "OrderDataDbAlias"     ; Name of existing DB Alias to use for connection to the DB
myDataSet = "MyOrderDataDbDataSet" ; Name of local dataset to use

mySQlStatement = "Select * FROM xml_job WHERE state=:myState and SelParams=:mySelParam" ; The sql-statement using parameters
mySQLStringList = "SqlList" ; Name of the local list to use for the sql-statement

myParamDescList = "ParamDescList"  ; Name of the local parameter description list

InitDatabase(myDBAlias)          ; Initiate DB connection
InitDataset(myDataSet, myDBALias)  ; Create local dataset

StringlistAdd(mySQLStringList, mySQlStatement) ; Add sql-statement to the sql-stringlist

;Fill parameter description list
StringListClear(myParamDescList)
StringListSetValue(myParamDescList, "myState", "ii")
StringListSetValue(myParamDescList, "mySelParam", "ii")

myState = "6"       ;Local variable used as first parameter
mySelParam ="0815"  ;Local variable used as second parameter

SetSQL(myDataSet, mySQLStringList)     ; Setting sql-statement (with no parameters in the statement, sql will be executed immediately)
PrepareSQL(myDataSet, myParamDescList) ; Executing sql-statement with parameters

While (EOF(myDataSet) = false)                ; As long as not EndOfFile
```

```
  BundleName  = GetField(myDataSet,"BundleName") ; Get the preferred data
  Protocol("BundleName: {0}", 8, BundleName)     ; Do something with the data
  Next(myDataSet)                                ; Position dataset cursor to next set of data
End-while
```

Xdata Functions

About Xdata Functions

This library provides functions that you can use in reading scripts of data objects.

Transaction Control

About Transaction Control

The StartTransaction, Commit, and Rollback functions allow you to group several database operations (select, update, delete, insert) into one transaction.

Example

In the following script for an Oracle database, the section **Q1** executes a select statement on the database.

After any data processing in the script, the function **ExecuteQuery()** runs an update on the selected dataset.

SQL statements

Q1: select * from TEST where status = 0 and id = (select min(id) from test where status = 0) for update

Update_Status: update TEST set status = 1 where id = :ID

To make sure the data is locked for other database sessions between the execution of **Q1** and **Update_ Status**, the select statement must contain an update lock using the **for update** command.

Both SQL commands must be executed in the transaction to maintain the database lock until the update has been executed.

```
db.StartTransaction()
db.WorkWithQuery(Q1)
db.ApplyDataSet()
…
system.ID = db.GetField(ID)
db.ExecuteQuery(Update_Status)
db.Commit()
```

StartTransaction

Starts a transaction in the database.

StartTransaction	
SYNTAX	StartTransaction()

Commit

Ends a database transaction.

Commit	
SYNTAX	Commit()
EXAMPLE	Commit()

Rollback

Discards all modify, insert, and delete actions of the current transaction and closes the transaction.

RollBack	
SYNTAX	RollBack()

AddDataRecord

Adds a dataset to the current data object.

AddDataRecord		
SYNTAX	AddDataRecord(columnValues)	
ARGUMENTS	columnValues	Comma-separated values you want to insert as a dataset into the data object.
EXAMPLE	db.AddDataRecord(1234,"John","Miller")	

ApplyDataRecord

Applies the current dataset of the logical data model.

ApplyDataRecord	
SYNTAX	ApplyDataRecord()
EXAMPLE	db.ApplyDataRecord()

ApplyDataSet

Applies the data from the last read statement to the logical data model. All datasets are applied where 1:N relations are found.

ApplyDataSet	
SYNTAX	ApplyDataSet()
EXAMPLE	db.ApplyDataSet()

DefineContainer

Defines a container to extract values from a database column and store them in so-called container columns.

DefineContainer		
SYNTAX	DefineContainer(usingMode, fieldName, delimiter) DefineContainer(usingMode, fieldName, delimiterList)	
ARGUMENTS	usingMode	Currently not available, the term is reserved for future implementation.
	fieldName	Name of the container field in the database table.

	delimiter	ASCII-Code of the delimiter, for example 44 for a COMMA.
	delimiterList	Comma separated list of delimiters. A delimiter is defined by its ASCII-Code. For example "44,59" specifies a comma or semicolon as possible delimiters.
EXAMPLE	db.WorkWithQuery("Q1") db.DefineContainer(0, "ContainerCol", 59) db.ApplyDataSet()	

Eof

Determines whether the cursor is at the end of the dataset.

Eof	
SYNTAX	Eof()
RETURN	TRUE = Eof reached. FALSE = Eof not reached.
EXAMPLE	End_Of_File = db.Eof()

ExecuteQuery

Executes an SQL query.

ExecuteQuery		
SYNTAX	ExecuteQuery(queryName)	
ARGUMENTS	queryName	Name of the query to execute.
EXAMPLE	`db.ExecuteQuery("Q2")`	

First

Positions the dataset cursor at the top of the current data object.

First	
SYNTAX	First()
EXAMPLE	db.First()

GetField

Returns the field value as a string.

GetField		
SYNTAX	GetField(fieldName)	
ARGUMENTS	fieldName	Name of the column.
RETURN	Field value.	
EXAMPLE	db.WorkWithQuery("Q1") db.ApplyDataSet() System.Par_Name = db.GetField("PAR_NAME")	

Last

Positions the dataset cursor at the last dataset of the current data object.

Last	
SYNTAX	Last()
EXAMPLE	db.Last()

Message

If an error has occurred, the resulting message can be processed by this function.

Message	
SYNTAX	Message()
RETURN	Error message.
EXAMPLE	EM=db.Message()

Next

Positions the dataset cursor on the next set of the current data object.

Next	
SYNTAX	Next()
EXAMPLE	db.Next()

PrepareParameter

Replaces unnamed parameters (%s) in an SQL statement before it passes to the database.

To obtain the value to use, you now need to define the system pool variable in which the value is to be found.

Use the PrepareParameter function to do this. The function passes a comma-separated string containing the names of the corresponding MonaLisa variables in the order they are used in the SQL statement.

The unnamed parameters must be submitted in the same way the database expects them, for example, enclosed in single quotation marks.

PrepareParameter		
SYNTAX	PrepareParameter(paramValues)	
ARGUMENTS	paramValues	Comma-separated name of the variable.

EXAMPLE	**Reading Script** system.ParNum = "0815" system.Nr = "2" system.Type = "N" db.PrepareParameter("LfdNr, Type") db.WorkWithQuery("Q1") db.ApplyDataSet() **Query** Select * from "ACCOUNTS.DB" where (ACC_Num = %s) AND (ACC_Par_Num = :ParNum) AND (ACC_Type = %s)

Prev

Positions the dataset cursor on the previous dataset of the current data object.

Prev	
SYNTAX	Prev()
EXAMPLE	db.Prev()

RowCount

Returns the number of datasets read.

RowCount	
SYNTAX	RowCount()
RETURN	Number of datasets.
EXAMPLE	i=db.Rowcount()

StartProcedure

Executes a stored procedure.

StartProcedure		
SYNTAX	StartProcedure(procedureName) StartProcedure(procedureName, usingMode)	
ARGUMENTS	procedureName	Name of the stored procedure
	usingMode	0 = Binding based on a parameter array. Parameters are specified according to the order of stored procedure definitions. 1 = Binding based on a parameter name. Parameters are passed on through name comparison.
EXAMPLE	db.StartProcedure("DM_PROCEDURE", 1)	

WorkWithQuery

Opens a dataset with an SQL query.

WorkWithQuery		
SYNTAX	WorkWithQuery(queryName)	
ARGUMENTS	queryName	Name of the query
EXAMPLE	db.WorkWithQuery("Q1")	

Date Functions

About MLDate

The MonaLisa object MLDate enables you to create an object providing date functions. To load the object, use the MonaLisa function GetObject.

Example

GetObject("Date","MLDate")

Format Specifiers

The format specifiers required as a parameter by some functions must comply to the rules laid down by the .NET framework.

Examples

Format Specifiers	Result
dd	Day, numeric two digits.
ddd	Name of the weekday (abbreviated).
dddd	Name of the weekday (full).
MM	Month, numeric two digits.
MMM	Name of the month (abbreviated).
MMMM	Name of the month (full).
yy	Year, numeric two digits.
yyyy	Year, numeric four digits.
hh	Hours
mm	Minutes
ss	Seconds

All .NET Framework date and time formatting strings are supported. For more information, refer to "Custom date and time format strings" in docs.microsoft.com.

Functions of the Class MLDate

CurrentDate

Returns the current date as a serial number.

CurrentDate

SYNTAX	CurrentDate()
RETURN	Date as a serial number.
EXAMPLE	GetObject("date","MLDate") CurrDateAsInt = date.CurrentDate()

CurrentDateToString

Returns the current date as a date string.

CurrentDateToString

SYNTAX	CurrentDateToString(format)
ARGUMENTS	format — Format rules defining the how the date string shows.
RETURN	Current date as a formatted date string.
EXAMPLE	GetObject("date","MLDate") CurrDateAsStr = date.CurrentDateToString("MM.dd.yyyy")

DateStringToInt64

Converts a date string to a serial number date.

DateStringToInt64

SYNTAX	DateStringToInt64(format, date)
ARGUMENTS	format — Format rules that define the how the date string displays. date — Date string you want to convert.
RETURN	Date serial number.
EXAMPLE	GetObject("date","MLDate") DateStrToInt = date.DateStringToInt64("MM/dd/yyyy", "10/31/2013")

DayOfWeek

Returns the week day for a date specified as a serial number.

DayOfWeek	
SYNTAX	DayOfWeek(date)
ARGUMENTS	date Integer date.
RETURN	Week day number. 0 = Sunday to 6 = Saturday.
EXAMPLE	GetObject("date","MLDate") CurrDateAsInt = date.CurrentDate() DayOfWeek = date.DayOfWeek(CurrDateAsInt)

DaysInMonth

Calculates the number of days in the specified month for a specific year.

DaysInMonth	
SYNTAX	DaysInMonth(month, year)
ARGUMENTS	month Month as an integer. year Year as an integer.
RETURN	Number of days.
EXAMPLE	GetObject("date","MLDate") DaysInMonth = date.DaysInMonth(2, 2014)

FormatDateTimeString

Formats the specified date string according to the rule for a date string.

FormatDateTimeString		
SYNTAX	FormatDateTimeString(format, date)	
ARGUMENTS	format	Format rules defining how the date string displays.
	date	Date string.
RETURN	Formatted date string.	
EXAMPLE	GetObject("date","MLDate") FormatedDateTimeStr = date.FormatDateTimeString("MM/dd/yyyy", "31/10/2014")	

IncDate

Adds a specific number of days, months and years to a date. If a negative number is defined, then the number is subtracted from the date.

IncDate		
SYNTAX	IncDate(date, days, months, years)	
ARGUMENTS	date	Date serial number.
	days	Number of days.
	months	Number of months.
	years	Number of years.
RETURN	Date serial number	
EXAMPLE	GetObject("date","MLDate") CurrDateAsInt = date.CurrentDate() IncDateInt = date.IncDate(CurrDateAsInt, 10, 0, 0)	

Int64ToDateString

Converts the specified serial number date in a formatted date string.

Int64ToDateString	
SYNTAX	Int64ToDateString(format, date)
ARGUMENTS	format — Format rules defining how the date string displays. date — Date serial number.
RETURN	Date as a formatted date string.
EXAMPLE	GetObject("date","MLDate") CurrDateAsInt = date.CurrentDate() DateIntToStr = date.Int64ToDateString("MM.dd.yyyy", CurrDateAsInt)

IsLeapYear

Checks whether the year in question is a leap year.

IsLeapYear	
SYNTAX	IsLeapYear(year)
ARGUMENTS	year — Year as a number.
RETURN	TRUE = The year is a leap year. FALSE = The year is not a leap year.
EXAMPLE	GetObject("date","MLDate") IsLeapYear = date.IsLeapYear(2014)

WeekOfYear

Returns the week number from the specified date serial number in accordance with ISO 8601.

WeekOfYear		
SYNTAX	WeekOfYear(date)	
ARGUMENTS	date	Date serial number
RETURN	Week number	
EXAMPLE	GetObject("date","MLDate") CurrDateAsInt = date.CurrentDate() WeekOfYear = date.WeekOfYear(CurrDateAsInt)	

File Processing

About MLDos

The MonaLisa object MLDos enables you to access files and to perform general file-based operations. To load the object, use the MonaLisa function GetObject.

Example

GetObject("DOS", "MLDos")

Functions of the Class MLDos

ChangeExtension

Changes the extension of a path string.

ChangeExtension	
SYNTAX	ChangeExtension(path, extension)
ARGUMENTS	Path information you want to modify
RETURN	String containing the altered path information.

Combine

Combines two path strings.

Combine

SYNTAX Combine(path1, path2)

ARGUMENTS

path1	First path.
path2	Second path.

RETURN A string containing the combined path entries.

If one of the specified paths is a zero-length string, this function returns the other path. If `path2` contains an absolute path, this function returns `path2`.

CopyFile

Copies an existing file to a new file.

CopyFile

SYNTAX CopyFile(sourceFileName, destFileName)

CopyFile(sourceFileName, destFileName, overwrite)

ARGUMENTS Name of the target file.

The name cannot be a directory and if the parameter 'overwrite' is not set, cannot be an existing file.

overwrite True when the target file can be overwritten, otherwise False

CreateTextFile

Creates a text file to use to read, write or append to the file.

CreateTextFile

SYNTAX CreateTextFile(newLine)

ARGUMENTS	inputFileName	Defines the string for a line break. If the parameter is not specified, CRLF ("\r\n" - $0D $0A) is used as the default line break. **Possible values** DOS.NewLineLF = LineFeed ("\n" - $0A) DOS.NewLineCRLF = CarriageReturn/LineFeed ("\r\n" - $0D $0A)
RETURN	MonaLisaTextFile object to use to read, write or append to the file.	

CreateDirectory

Creates all directories and subdirectories according to a specified path.

CreateDirectory		
SYNTAX	CreateDirectory(path)	
ARGUMENTS	path	The directory path you want to create.

DeleteDirectory

Deletes the specified directory, and if so configured, all subdirectories thereof.

DeleteDirectory		
SYNTAX	DeleteDirectory(path) DeleteDirectory(path, recursive)	
ARGUMENTS	path	Name of the empty directory you want to delete. This directory must be writable or empty.
	recursive	TRUE = Delete the directories, subdirectories, and files in `path`. FALSE = Do not delete the directories, subdirectories, and files.

DeleteFile

Deletes the specified file.

Note No exception is thrown, if the file does not exist.

DeleteFile		
SYNTAX	DeleteFile(path)	
ARGUMENTS	path	Name of the file to delete.

DeleteFiles

Deletes all files in a directory and subdirectory that match a search pattern.

DeleteFiles		
SYNTAX	DeleteFiles(path, searchPattern, recursive)	
ARGUMENTS	path	Name of the directory.
	searchPattern	The search string to match against the names of files in `path`.
	recursive	TRUE = Search subdirectories. FALSE = Do not search subdirectories.

DirectoryExists

Determines whether the specified path points to a directory on a drive.

DirectoryExists		
SYNTAX	DirectoryExists(path)	
ARGUMENTS	path	Path you want to test.

RETURN	TRUE = Path exists. FALSE = Path does not exist.

ExecuteSynchron

Starts the specified program and returns an exit code when the program is closed.

ExecuteSynchron

SYNTAX ExecuteSynchron(file, parameter, hide)

ARGUMENTS

file	Program with path.
parameter	Program parameters.
hide	TRUE = Hides the window of the started program. FALSE = Displays the window of the started program .

RETURN Program exit code

Note for 64-bit operating systems

Content Composer is a 32-bit program.

If you use the **ExecuteSynchron** command to start a program from directory **%windir%\System32**, you must define the the directory **%windir%\Sysnative** instead.

If you use the **ExecuteSynchron** command to start a program from directory **C:\windows\System32**, you must define the the directory **C:\windows\Sysnative** instead.

Refer to the Microsoft Developer Network website for more information on the File System Redirector.

FileAppendText

Opens a file, appends the specified string to the file, and then closes the file.

If the file does not already exist, the function creates it, writes the string in the file, and closes it.

FileAppendText

SYNTAX FileAppendText(fileName, content)

FileAppendText(fileName, content, encoding)

ARGUMENTS

fileName	The file to which the string appends.
content	The string to append to the file.
encoding	You can use the following values as encoding parameters. • Empty string. In this case, the default encoding (ANSI code page of the operating system) is used. • Code page, for example, "1252" • Name, for example, "windows-1252" • Friendly name, for example, "Western European" (Windows) Possible encoding parameters

Code page	name	Display name
37	IBM037	IBM EBCDIC (USA-Canada)
437	IBM437	OEM USA
500	IBM500	IBM EBCDIC (International)
708	ASMO-708	Arabic (ASMO 708)
720	DOS-720	Arabic (DOS)
737	ibm737	Greek (DOS)
775	ibm775	Baltic (DOS)
850	ibm850	Western European (DOS)
852	ibm852	Eastern European (DOS)
855	IBM855	OEM Cyrillic
857	ibm857	Turkish (DOS)
858	IBM00858	OEM Multilingual Latin I
860	IBM860	Portuguese (DOS)
861	ibm861	Icelandic (DOS)
862	DOS-862	Hebrew (DOS)
863	IBM863	French, Canada (DOS)

864	IBM864	Arabic (864)
865	IBM865	Nordic (DOS)
866	cp866	Cyrillic (DOS)
869	ibm869	Greek, modern (DOS)
870	IBM870	IBM EBCDIC (Multilingual Latin-2)
874	windows-874	Thai (Windows)
875	cp875	IBM EBCDIC (Greek, modern)
932	shift_jis	Japanese (Shift-JIS)
936	gb2312	Simplified Chinese (GB2312)
949	ks_c_5601-1987	Korean
950	big5	Chinese (Traditional) (Big5)
1026	IBM1026	IBM EBCDIC (Turkish, Latin-5)
1047	IBM01047	IBM Latin-1
1140	IBM01140	IBM EBCDIC (USA-Canada-European)
1141	IBM01141	IBM EBCDIC (Germany-European)
1142	IBM01142	IBM EBCDIC (Denmark-Norway-European)
1143	IBM01143	IBM EBCDIC (Finland-Sweden-European)
1144	IBM01144	IBM EBCDIC (Italian-European)
1145	IBM01145	IBM EBCDIC (Spain-European)
1146	IBM01146	IBM EBCDIC (Great Britain-European)
1147	IBM01147	IBM EBCDIC (France-European)
1148	IBM01148	IBM EBCDIC (International-European)
1149	IBM01149	IBM EBCDIC (Icelandic-European)
1200	utf-16	Unicode
1201	unicodeFFFE	Unicode (Big Endian)
1250	windows-1250	Central European (Windows)
1251	windows-1251	Cyrillic (Windows)
1252	Windows-1252	Western European (Windows)
1253	windows-1253	Greek (Windows)

1254	windows-1254	Turkish (Windows)
1255	windows-1255	Hebrew (Windows)
1256	windows-1256	Arabic (Windows)
1257	windows-1257	Baltic (Windows)
1258	windows-1258	Vietnamese (Windows)
1361	Johab	Korean (Johab)
10000	macintosh	Western European (Mac)
10001	x-mac-japanese	Japanese (Mac)
10002	x-mac-chinesetrad	Chinese traditional (Mac)
10003	x-mac-korean	Korean (Mac)
10004	x mac-arabic	Arabic (Mac)
10005	x-mac-hebrew	Hebrew (Mac)
10006	x-mac-greek	Greek (Mac)
10007	x-mac-cyrillic	Cyrillic (Mac)
10008	x-mac-chinesesimp	Chinese simplified (Mac)
10010	x-mac-romanian	Romanian (Mac)
10017	x-mac-ukrainian	Ukrainian (Mac)
10021	x-mac-thai	Thai (Mac)
10029	x-mac-ce	Central European (Mac)
10079	x-mac-icelandic	Icelandic (Mac)
10081	x-mac-turkish	Turkish (Mac)
10082	x-mac-croatian	Croatian (Mac)
12000	utf-32	Unicode (UTF-32)
12001	utf-32BE	Unicode (UTF-32-Big Endian)
20000	x-Chinese-CNS	Chinese traditional (CNS)
20001	x-cp20001	TCA Taiwan
20002	x-Chinese-Eten	Chinese traditional (Eten)
20003	x-cp20003	IBM5550 Taiwan
20004	x-cp20004	TeleText Taiwan
20005	x-cp20005	Wang Taiwan

20105	x-IA5	Western European (IA5)
20106	x-IA5-German	German (IA5)
20107	x-IA5-Swedish	Swedish (IA5)
20108	x-IA5-Norwegian	Norwegian (IA5)
20127	us-ascii	US-ASCII
20261	x-cp20261	T.61
20269	x-cp20269	ISO-6937
20273	IBM273	IBM EBCDIC (Germany)
20277	IBM277	IBM EBCDIC (Denmark-Norway)
20278	IBM278	IBM EBCDIC (Finland-Sweden)
20280	IBM280	IBM EBCDIC (Italy)
20284	IBM284	IBM EBCDIC (Spain)
20285	IBM285	IBM EBCDIC (Great Britain)
20290	IBM290	IBM EBCDIC (Japanese Katakana)
20297	IBM297	IBM EBCDIC (France)
20420	IBM420	IBM EBCDIC (Arabic)
20423	IBM423	IBM EBCDIC (Greek)
20424	IBM424	IBM EBCDIC (Hebrew)
20833	x-EBCDIC-KoreanExtended	IBM EBCDIC (Korean, extended)
20838	IBM-Thai	IBM EBCDIC (Thai)
20866	koi8-r	Cyrillic (KOI8-R)
20871	IBM871	IBM EBCDIC (Icelandic)
20880	IBM880	IBM EBCDIC (Cyrillic, Russian)
20905	IBM905	IBM EBCDIC (Turkish)
20924	IBM00924	IBM Latin-1
20932	EUC-JP	Japanese (JIS 0208-1990 and 0212-1990)
20936	x-cp20936	GB2312-80 Chinese (simple)
20949	x-cp20949	Korean Wansung
21025	cp1025	IBM EBCDIC (Cyrillic, Serbian-Bulgarian)

21866	koi8-u	Cyrillic (KOI8-U)
28591	iso-8859-1	Western European (ISO)
28592	iso-8859-2	Central European (ISO)
28593	iso-8859-3	Latin 3 (ISO)
28594	iso-8859-4	Baltic (ISO)
28595	iso-8859-5	Cyrillic (ISO)
28596	iso-8859-6	Arabic (ISO)
28597	iso-8859-7	Greek (ISO)
28598	iso-8859-8	Hebrew (ISO-Visual)
28599	iso-8859-9	Turkish (ISO)
28603	iso-8859-13	Estonian (ISO)
28605	iso-8859-15	Latin 9 (ISO)
29001	x-Europa	Europe
38598	iso-8859-8-i	Hebrew (ISO-Logical)
50220	iso-2022-jp	Japanese (JIS)
50221	csISO2022JP	Japanese (JIS, 1 Byte Kana allowed)
50222	iso-2022-jp	Japanese (JIS, 1 Byte Kana allowed - SO/SI)
50225	iso-2022-kr	Korean (ISO)
50227	x-cp50227	ISO-2022 Chinese (simplified)
51932	euc-jp	Japanese (EUC)
51936	EUC-CN	Chinese simplified (EUC)
51949	euc-kr	Korean (EUC)
52936	hz-gb-2312	Chinese simplified (HZ)
54936	GB18030	GB18030 Chines simplified
57002	x-iscii-de	ISCII Devanagari
57003	x-iscii-be	ISCII Bengali
57004	x-iscii-ta	ISCII Tamil
57005	x-iscii-te	ISCII Telugu
57006	x-iscii as	ISCII Assamese
57007	x-iscii-or	ISCII Oriya

57008	x-iscii-ka	ISCII Kannada
57009	x-iscii-ma	ISCII Malayalam
57010	x-iscii-gu	ISCII Gujarati
57011	x-iscii-pa	ISCII Punjabi
65000	utf-7	Unicode (UTF-7)
65001	utf-8	Unicode (UTF-8)

FileExists

Checks whether the specified file exists.

FileExists

SYNTAX FileExists(path)

ARGUMENTS

path	The file to check.

RETURN TRUE = The caller has sufficient rights and **`path`** contains the name of an existing file.

FALSE = Invalid path or string length 0 set for **`path`**. An exception is thrown if a caller does not possess sufficient rights to read the specified file. The function returns FALSE irrespective of whether **`path`** exists.

FileReadAllBytes

Opens a file, reads the file content into a byte array, and then closes the file.

FileReadAllBytes

SYNTAX FileReadAllBytes(fileName)

ARGUMENTS

fileName	The file to open.

RETURN The complete file content as byte array.

FileReadAllText

Opens a text file, reads all lines of the file, and then closes the file.

FileReadAllText

SYNTAX

FileReadAllText(fileName)

FileReadAllText(fileName, encoding)

ARGUMENTS

fileName	The file to open.
encoding	You can use the following values as encoding parameters. • Empty string. In this case, the default encoding (ANSI code page of the operating system) is used. • Code page, for example, "1252" • Name, for example, "windows-1252" • Friendly name, for example, "Western European" (Windows) Possible encoding parameters

Code page	name	Display name
37	IBM037	IBM EBCDIC (USA-Canada)
437	IBM437	OEM USA
500	IBM500	IBM EBCDIC (International)
708	ASMO-708	Arabic (ASMO 708)
720	DOS-720	Arabic (DOS)
737	ibm737	Greek (DOS)
775	ibm775	Baltic (DOS)
850	ibm850	Western European (DOS)
852	ibm852	Eastern European (DOS)
855	IBM855	OEM Cyrillic
857	ibm857	Turkish (DOS)
858	IBM00858	OEM Multilingual Latin I

860	IBM860	Portuguese (DOS)
861	ibm861	Icelandic (DOS)
862	DOS-862	Hebrew (DOS)
863	IBM863	French, Canada (DOS)
864	IBM864	Arabic (864)
865	IBM865	Nordic (DOS)
866	cp866	Cyrillic (DOS)
869	ibm869	Greek, modern (DOS)
870	IBM870	IBM EBCDIC (Multilingual Latin-2)
874	windows-874	Thai (Windows)
875	cp875	IBM EBCDIC (Greek, modern)
932	shift_jis	Japanese (Shift-JIS)
936	gb2312	Simplified Chinese (GB2312)
949	ks_c_5601-1987	Korean
950	big5	Chinese (Traditional) (Big5)
1026	IBM1026	IBM EBCDIC (Turkish, Latin-5)
1047	IBM01047	IBM Latin-1
1140	IBM01140	IBM EBCDIC (USA-Canada-European)
1141	IBM01141	IBM EBCDIC (Germany-European)
1142	IBM01142	IBM EBCDIC (Denmark-Norway-European)
1143	IBM01143	IBM EBCDIC (Finland-Sweden-European)
1144	IBM01144	IBM EBCDIC (Italian-European)
1145	IBM01145	IBM EBCDIC (Spain-European)
1146	IBM01146	IBM EBCDIC (Great Britain-European)
1147	IBM01147	IBM EBCDIC (France-European)
1148	IBM01148	IBM EBCDIC (International-European)
1149	IBM01149	IBM EBCDIC (Icelandic-European)
1200	utf-16	Unicode
1201	unicodeFFFE	Unicode (Big Endian)

1250	windows-1250	Central European (Windows)
1251	windows-1251	Cyrillic (Windows)
1252	Windows-1252	Western European (Windows)
1253	windows-1253	Greek (Windows)
1254	windows-1254	Turkish (Windows)
1255	windows-1255	Hebrew (Windows)
1256	windows-1256	Arabic (Windows)
1257	windows-1257	Baltic (Windows)
1258	windows-1258	Vietnamese (Windows)
1361	Johab	Korean (Johab)
10000	macintosh	Western European (Mac)
10001	x-mac-japanese	Japanese (Mac)
10002	x-mac-chinesetrad	Chinese traditional (Mac)
10003	x-mac-korean	Korean (Mac)
10004	x mac-arabic	Arabic (Mac)
10005	x-mac-hebrew	Hebrew (Mac)
10006	x-mac-greek	Greek (Mac)
10007	x-mac-cyrillic	Cyrillic (Mac)
10008	x-mac-chinesesimp	Chinese simplified (Mac)
10010	x-mac-romanian	Romanian (Mac)
10017	x-mac-ukrainian	Ukrainian (Mac)
10021	x-mac-thai	Thai (Mac)
10029	x-mac-ce	Central European (Mac)
10079	x-mac-icelandic	Icelandic (Mac)
10081	x-mac-turkish	Turkish (Mac)
10082	x-mac-croatian	Croatian (Mac)
12000	utf-32	Unicode (UTF-32)
12001	utf-32BE	Unicode (UTF-32-Big Endian)
20000	x-Chinese-CNS	Chinese traditional (CNS)
20001	x-cp20001	TCA Taiwan

20002	x-Chinese-Eten	Chinese traditional (Eten)
20003	x-cp20003	IBM5550 Taiwan
20004	x-cp20004	TeleText Taiwan
20005	x-cp20005	Wang Taiwan
20105	x-IA5	Western European (IA5)
20106	x-IA5-German	German (IA5)
20107	x-IA5-Swedish	Swedish (IA5)
20108	x-IA5-Norwegian	Norwegian (IA5)
20127	us-ascii	US-ASCII
20261	x-cp20261	T.61
20269	x-cp20269	ISO-6937
20273	IBM273	IBM EBCDIC (Germany)
20277	IBM277	IBM EBCDIC (Denmark-Norway)
20278	IBM278	IBM EBCDIC (Finland-Sweden)
20280	IBM280	IBM EBCDIC (Italy)
20284	IBM284	IBM EBCDIC (Spain)
20285	IBM285	IBM EBCDIC (Great Britain)
20290	IBM290	IBM EBCDIC (Japanese Katakana)
20297	IBM297	IBM EBCDIC (France)
20420	IBM420	IBM EBCDIC (Arabic)
20423	IBM423	IBM EBCDIC (Greek)
20424	IBM424	IBM EBCDIC (Hebrew)
20833	x-EBCDIC-KoreanExtended	IBM EBCDIC (Korean, extended)
20838	IBM-Thai	IBM EBCDIC (Thai)
20866	koi8-r	Cyrillic (KOI8-R)
20871	IBM871	IBM EBCDIC (Icelandic)
20880	IBM880	IBM EBCDIC (Cyrillic, Russian)
20905	IBM905	IBM EBCDIC (Turkish)
20924	IBM00924	IBM Latin-1
20932	EUC-JP	Japanese (JIS 0208-1990 and 0212-1990)

20936	x-cp20936	GB2312-80 Chinese (simple)
20949	x-cp20949	Korean Wansung
21025	cp1025	IBM EBCDIC (Cyrillic, Serbian-Bulgarian)
21866	koi8-u	Cyrillic (KOI8-U)
28591	iso-8859-1	Western European (ISO)
28592	iso-8859-2	Central European (ISO)
28593	iso-8859-3	Latin 3 (ISO)
28594	iso-8859-4	Baltic (ISO)
28595	iso-8859-5	Cyrillic (ISO)
28596	iso-8859-6	Arabic (ISO)
28597	iso-8859-7	Greek (ISO)
28598	iso-8859-8	Hebrew (ISO-Visual)
28599	iso-8859-9	Turkish (ISO)
28603	iso-8859-13	Estonian (ISO)
28605	iso-8859-15	Latin 9 (ISO)
29001	x-Europa	Europe
38598	iso-8859-8-i	Hebrew (ISO-Logical)
50220	iso-2022-jp	Japanese (JIS)
50221	csISO2022JP	Japanese (JIS, 1 Byte Kana allowed)
50222	iso-2022-jp	Japanese (JIS, 1 Byte Kana allowed - SO/SI)
50225	iso-2022-kr	Korean (ISO)
50227	x-cp50227	ISO-2022 Chinese (simplified)
51932	euc-jp	Japanese (EUC)
51936	EUC-CN	Chinese simplified (EUC)
51949	euc-kr	Korean (EUC)
52936	hz-gb-2312	Chinese simplified (HZ)
54936	GB18030	GB18030 Chines simplified
57002	x-iscii-de	ISCII Devanagari
57003	x-iscii-be	ISCII Bengali

57004	x-iscii-ta	ISCII Tamil
57005	x-iscii-te	ISCII Telugu
57006	x-iscii as	ISCII Assamese
57007	x-iscii-or	ISCII Oriya
57008	x-iscii-ka	ISCII Kannada
57009	x-iscii-ma	ISCII Malayalam
57010	x-iscii-gu	ISCII Gujarati
57011	x-iscii-pa	ISCII Punjabi
65000	utf-7	Unicode (UTF-7)
65001	utf-8	Unicode (UTF-8)

RETURN A string containing all lines of the file.

FileSize

Returns the size of a file.

FileSize

SYNTAX FileSize(fileName)

ARGUMENTS

fileName	Name of the file.

RETURN Size of the file.

GetCommandLine

Returns the command line for the process.

GetCommandLine

SYNTAX GetCommandLine()

RETURN String containing all command line arguments.

GetCommandLineArgByName

Returns the specified command line argument.

GetCommandLineArgByName		
SYNTAX	GetCommandLineArgByName(name)	
ARGUMENTS	name	Name of the command line argument.
RETURN	Value of the command line argument, or an empty string if the parameter does not exist.	

GetCommandLineArgs

Returns a comma separated string of the command line arguments of the current process.

GetCommandLineArgs	
SYNTAX	GetCommandLineArgs()
RETURN	Comma separated string of the command line arguments, or an empty string.

GetCurrentDirectory

Gets the current working directory of the program.

GetCurrentDirectory	
SYNTAX	GetCurrentDirectory()
RETURN	String containing the path of the current working directory.

GetDirectories

Returns the names of subdirectories in a specified directory as a comma separated string.

GetDirectories

SYNTAX	GetDirectories(path) GetDirectories(path, searchPattern) GetDirectories(path, searchPattern, recursive)
ARGUMENTS	path — Path where the names of the subdirectories are obtained. searchPattern — The search string to match against the names of files in `path`. recursive — TRUE = Search subdirectories. FALSE = Do not search subdirectories.
RETURN	Comma separated string containing the names of all subdirectories.

GetDirectoryName

Returns the directory information for a specified path string.

GetDirectoryName

SYNTAX	GetDirectoryName(path)
ARGUMENTS	path — Path of a file or directory.
RETURN	String containing directory information for path or an empty string if `path` has no directory information.

GetDirectoryRoot

Returns information regarding the volume, root or both for the specified path.

GetDirectoryRoot

SYNTAX	GetDirectoryRoot(path)
ARGUMENTS	path — Path of a file or directory.
RETURN	String containing information regarding the volume, root or both.

GetEnvironmentVariable

Returns the value of an environment variable.

GetEnvironmentVariable		
SYNTAX	GetEnvironmentVariable(variable)	
ARGUMENTS	variable	Name of the environment variable.
RETURN	The value of the environment variable specified as `variable`, or an empty string if the environment variable does not exist.	

GetExtension

Returns the file extension of the specified path string.

GetExtension		
SYNTAX	GetExtension(path)	
ARGUMENTS	path	Path string from which the extension is read.
RETURN	String containing the extension of the specified path (including ".").	

GetFileName

Returns the file name and extension of the specified path.

GetFileName		
SYNTAX	GetFileName(path)	
ARGUMENTS	path	Path where the file name and extension is obtained.
RETURN	String containing the characters after the last directory character in path. If the last character of path is a directory or volume separator character, this function returns an empty string.	

GetFileNameWithoutExtension

Returns the name of the file from the specified path string without the extension.

GetFileNameWithoutExtension		
SYNTAX	GetFileNameWithoutExtension(path)	
ARGUMENTS	path	Path of the file.
RETURN	String containing the result string from GetFileName without the last dot (.) and all following characters.	

GetFiles

Returns the names of files in a specified directory as a comma separated string.

GetFiles		
SYNTAX	GetFiles(path) GetFiles(path, searchPattern) GetFiles(path, searchPattern, recursive)	
ARGUMENTS	path	The directory from which the files are obtained.
	searchPattern	The search string to match against the names of files in path. The searchPattern parameter cannot end in two periods ("..") or contain two periods ("..")
	recursive	TRUE = Search in the current directory and all subdirectories. **Note** If the directory structure contains a link that creates a loop back within the directory structure, the search results in an infinite loop. FALSE = Search only in the current directory.
RETURN	A comma separated string of file names in the specified directory. Defining the searchPattern parameter returns the names of files that match the specified search pattern. File names contain the full path.	

GetFileSystemEntries

Returns the names of all files and subdirectories in a specified directory in a comma separated string.

GetFileSystemEntries		
SYNTAX	GetFileSystemEntries(path) GetFileSystemEntries(path, searchPattern)	
ARGUMENTS	path	The directory the files are obtained from.
	searchPattern	The search string to match against the names of files in path. The searchPattern parameter cannot end in two periods ("..") or contain two periods ("..")
RETURN	A comma separated string containing the names of the file system entries found in the specified directory and matching the search criteria specified in `searchPattern`.	

GetFullPath

Returns the absolute path for a specified path string.

GetFullPath		
SYNTAX	GetFullPath(path)	
ARGUMENTS	path	File or directory for which the absolute path information is retrieved.
RETURN	String containing the absolute path of the variable path, for example "C:\MyFile.txt".	

GetLogicalDrives

Returns the names of the logical drives of this computer in the format "<DriveLetter>:\".

GetLogicalDrives	
SYNTAX	GetLogicalDrives()
RETURN	The logical drives of this computer.

GetParentDirectory

Retrieves the parent directory of the specified path, including both absolute and relative paths.

GetParentDirectory		
SYNTAX	GetParentDirectory(path)	
ARGUMENTS	path	Path from which the parent directory retrieves.
RETURN	The parent directory or an empty string when `path` is the root directory, including the root of a UNC server or share name.	

GetPathRoot

Returns the root directory information of the specified path.

GetPathRoot		
SYNTAX	GetPathRoot(path)	
ARGUMENTS	path	Path from which the root directory information retrieves.
RETURN	A string containing the root directory from `path`, for example "C:\" or an empty string when `path` contains no information on the root directory.	

GetRandomFileName

Returns a random file or directory name.

GetRandomFileName	
SYNTAX	GetRandomFileName()
RETURN	Random file or directory name.

Notes

- This function returns a cryptographically strong, random string that you can use as either a directory name or a file name.
- Unlike GetTempFileName, the GetRandomFileName function does not create a file.
- When the security of your file system is paramount, use this function instead of GetTempFileName.

GetTempFileName

Creates a uniquely named temporary file, with the **.tmp** extension, of zero bytes on a data storage device and returns the full path to this file.

GetTempFileName	
SYNTAX	GetTempFileName()
RETURN	String containing the full path of the temporary file.

GetTempPath

Returns the path to the temporary directory of the current system.

GetTempPath	
SYNTAX	GetTempPath()
RETURN	String containing the path to a temporary directory.

HasExtension

Determines whether a path contains a file name extension.

HasExtension		
SYNTAX	HasExtension(path)	
ARGUMENTS	path	Path in which to search for an extension.
RETURN	TRUE = The characters that follow the last directory separator (\\ or /) or volume separator character (:) contain a period (.) and one or more trailing characters. FALSE = The path does not contain a file name extension.	

IsPathRooted

Returns a value denoting whether the specified path string contains a relative or absolute path.

IsPathRooted		
SYNTAX	IsPathRooted(path)	
ARGUMENTS	path	Path to test.
RETURN	TRUE = The string contains an absolute path. FALSE = The string contains a relative path.	

MoveDirectory

Moves a file or directory and its content to a new storage location.

MoveDirectory		
SYNTAX	`MoveDirectory(sourceDirName, destDirName)`	
ARGUMENTS	sourceDirName	Path of the file or directory to move.
	destDirName	Path to the new storage location for `sourceDirName`. If `sourceDirName` is a file, then `destDirName` must also be a file name.

MoveFile

Moves the specified file to a new storage location and also allows you to modify the name of the file.

MoveFile	
SYNTAX	`MoveFile(sourceFileName, destFileName)`

ARGUMENTS	sourceFileName	Name of the file to move.
	destFileName	New path to the file.

ReplaceFile

Replaces the content of a specified file with the content of another file, deletes the original file, and creates a back up copy of the replaced file.

ReplaceFile		
SYNTAX	ReplaceFile(sourceFileName, destinationFileName, destinationBackupFileName)	
ARGUMENTS	sourceFileName	The name of the file that the file defined as `destinationFileName` replaces.
	destinationFileName	The name of the file to replace.
	destinationBackupFileName	The name of the backup file of the file that is replaced or an empty string if no backup file is required.

SetCurrentDirectory

Sets the current working directory of the program to the specified directory.

SetCurrentDirectory		
SYNTAX	SetCurrentDirectory(path)	
ARGUMENTS	path	Path of the current working directory

Properties of the Class MLDos

Property	Description
NewLine	Gets the string for a line break defined for a specific environment.
NewLineLF	Returns LineFeed as a string ("\n" - $0A).
NewLineCRLF	Returns CarriageReturn / LineFeed (CRLF) as a string ("\r\n" - $0D $0A).

Functions of the Class MonaLisaTextFile

Append

Opens a file to append text.

Append

SYNTAX

Append(string fileName)

Append(string fileName, string encoding)

ARGUMENTS

fileName	The path to the file you want to append.
encoding	You can use the following values as encoding parameters. • Empty string. In this case, the default encoding (ANSI code page of the operating system) is used. • Code page, for example, "1252" • Name, for example, "windows-1252" • Friendly name, for example, "Western European" (Windows)

Possible encoding parameters

Code page	name	Display name
37	IBM037	IBM EBCDIC (USA-Canada)
437	IBM437	OEM USA

500	IBM500	IBM EBCDIC (International)
708	ASMO-708	Arabic (ASMO 708)
720	DOS-720	Arabic (DOS)
737	ibm737	Greek (DOS)
775	ibm775	Baltic (DOS)
850	ibm850	Western European (DOS)
852	ibm852	Eastern European (DOS)
855	IBM855	OEM Cyrillic
857	ibm857	Turkish (DOS)
858	IBM00858	OEM Multilingual Latin I
860	IBM860	Portuguese (DOS)
861	ibm861	Icelandic (DOS)
862	DOS-862	Hebrew (DOS)
863	IBM863	French, Canada (DOS)
864	IBM864	Arabic (864)
865	IBM865	Nordic (DOS)
866	cp866	Cyrillic (DOS)
869	ibm869	Greek, modern (DOS)
870	IBM870	IBM EBCDIC (Multilingual Latin-2)
874	windows-874	Thai (Windows)
875	cp875	IBM EBCDIC (Greek, modern)
932	shift_jis	Japanese (Shift-JIS)
936	gb2312	Simplified Chinese (GB2312)
949	ks_c_5601-1987	Korean
950	big5	Chinese (Traditional) (Big5)
1026	IBM1026	IBM EBCDIC (Turkish, Latin-5)
1047	IBM01047	IBM Latin-1
1140	IBM01140	IBM EBCDIC (USA-Canada-European)
1141	IBM01141	IBM EBCDIC (Germany-European)
1142	IBM01142	IBM EBCDIC (Denmark-Norway-European)

1143	IBM01143	IBM EBCDIC (Finland-Sweden-European)
1144	IBM01144	IBM EBCDIC (Italian-European)
1145	IBM01145	IBM EBCDIC (Spain-European)
1146	IBM01146	IBM EBCDIC (Great Britain-European)
1147	IBM01147	IBM EBCDIC (France-European)
1148	IBM01148	IBM EBCDIC (International-European)
1149	IBM01149	IBM EBCDIC (Icelandic-European)
1200	utf-16	Unicode
1201	unicodeFFFE	Unicode (Big Endian)
1250	windows-1250	Central European (Windows)
1251	windows-1251	Cyrillic (Windows)
1252	Windows-1252	Western European (Windows)
1253	windows-1253	Greek (Windows)
1254	windows-1254	Turkish (Windows)
1255	windows-1255	Hebrew (Windows)
1256	windows-1256	Arabic (Windows)
1257	windows-1257	Baltic (Windows)
1258	windows-1258	Vietnamese (Windows)
1361	Johab	Korean (Johab)
10000	macintosh	Western European (Mac)
10001	x-mac-japanese	Japanese (Mac)
10002	x-mac-chinesetrad	Chinese traditional (Mac)
10003	x-mac-korean	Korean (Mac)
10004	x mac-arabic	Arabic (Mac)
10005	x-mac-hebrew	Hebrew (Mac)
10006	x-mac-greek	Greek (Mac)
10007	x-mac-cyrillic	Cyrillic (Mac)
10008	x-mac-chinesesimp	Chinese simplified (Mac)
10010	x-mac-romanian	Romanian (Mac)
10017	x-mac-ukrainian	Ukrainian (Mac)

10021	x-mac-thai	Thai (Mac)
10029	x-mac-ce	Central European (Mac)
10079	x-mac-icelandic	Icelandic (Mac)
10081	x-mac-turkish	Turkish (Mac)
10082	x-mac-croatian	Croatian (Mac)
12000	utf-32	Unicode (UTF-32)
12001	utf-32BE	Unicode (UTF-32-Big Endian)
20000	x-Chinese-CNS	Chinese traditional (CNS)
20001	x-cp20001	TCA Taiwan
20002	x-Chinese-Eten	Chinese traditional (Eten)
20003	x-cp20003	IBM5550 Taiwan
20004	x-cp20004	TeleText Taiwan
20005	x-cp20005	Wang Taiwan
20105	x-IA5	Western European (IA5)
20106	x-IA5-German	German (IA5)
20107	x-IA5-Swedish	Swedish (IA5)
20108	x-IA5-Norwegian	Norwegian (IA5)
20127	us-ascii	US-ASCII
20261	x-cp20261	T.61
20269	x-cp20269	ISO-6937
20273	IBM273	IBM EBCDIC (Germany)
20277	IBM277	IBM EBCDIC (Denmark-Norway)
20278	IBM278	IBM EBCDIC (Finland-Sweden)
20280	IBM280	IBM EBCDIC (Italy)
20284	IBM284	IBM EBCDIC (Spain)
20285	IBM285	IBM EBCDIC (Great Britain)
20290	IBM290	IBM EBCDIC (Japanese Katakana)
20297	IBM297	IBM EBCDIC (France)
20420	IBM420	IBM EBCDIC (Arabic)
20423	IBM423	IBM EBCDIC (Greek)
20424	IBM424	IBM EBCDIC (Hebrew)

20833	x-EBCDIC-KoreanExtended	IBM EBCDIC (Korean, extended)
20838	IBM-Thai	IBM EBCDIC (Thai)
20866	koi8-r	Cyrillic (KOI8-R)
20871	IBM871	IBM EBCDIC (Icelandic)
20880	IBM880	IBM EBCDIC (Cyrillic, Russian)
20905	IBM905	IBM EBCDIC (Turkish)
20924	IBM00924	IBM Latin-1
20932	EUC-JP	Japanese (JIS 0208-1990 and 0212-1990)
20936	x-cp20936	GB2312-80 Chinese (simple)
20949	x-cp20949	Korean Wansung
21025	cp1025	IBM EBCDIC (Cyrillic, Serbian-Bulgarian)
21866	koi8-u	Cyrillic (KOI8-U)
28591	iso-8859-1	Western European (ISO)
28592	iso-8859-2	Central European (ISO)
28593	iso-8859-3	Latin 3 (ISO)
28594	iso-8859-4	Baltic (ISO)
28595	iso-8859-5	Cyrillic (ISO)
28596	iso-8859-6	Arabic (ISO)
28597	iso-8859-7	Greek (ISO)
28598	iso-8859-8	Hebrew (ISO-Visual)
28599	iso-8859-9	Turkish (ISO)
28603	iso-8859-13	Estonian (ISO)
28605	iso-8859-15	Latin 9 (ISO)
29001	x-Europa	Europe
38598	iso-8859-8-i	Hebrew (ISO-Logical)
50220	iso-2022-jp	Japanese (JIS)
50221	csISO2022JP	Japanese (JIS, 1 Byte Kana allowed)
50222	iso-2022-jp	Japanese (JIS, 1 Byte Kana allowed - SO/SI)

50225	iso-2022-kr	Korean (ISO)
50227	x-cp50227	ISO-2022 Chinese (simplified)
51932	euc-jp	Japanese (EUC)
51936	EUC-CN	Chinese simplified (EUC)
51949	euc-kr	Korean (EUC)
52936	hz-gb-2312	Chinese simplified (HZ)
54936	GB18030	GB18030 Chines simplified
57002	x-iscii-de	ISCII Devanagari
57003	x-iscii-be	ISCII Bengali
57004	x-iscii-ta	ISCII Tamil
57005	x-iscii-te	ISCII Telugu
57006	x-iscii as	ISCII Assamese
57007	x-iscii-or	ISCII Oriya
57008	x-iscii-ka	ISCII Kannada
57009	x-iscii-ma	ISCII Malayalam
57010	x-iscii-gu	ISCII Gujarati
57011	x-iscii-pa	ISCII Punjabi
65000	utf-7	Unicode (UTF-7)
65001	utf-8	Unicode (UTF-8)

EXAMPLE

```
File.Append("c:\temp\test.txt")
```

Close

Closes an open file.

Close

SYNTAX Close()

EXAMPLE Close()

Create

Creates or opens a file in which to write text.

Create

SYNTAX

Create(fileName)

Create(fileName, encoding)

ARGUMENTS

fileName — The file you want to create or open.

encoding — You can use the following values as encoding parameters.

- Empty string. In this case, the default encoding (ANSI code page of the operating system) is used.
- Code page, for example, "1252"
- Name, for example, "windows-1252"
- Friendly name, for example, "Western European" (Windows)

Possible encoding parameters

Code page	name	Display name
37	IBM037	IBM EBCDIC (USA-Canada)
437	IBM437	OEM USA
500	IBM500	IBM EBCDIC (International)
708	ASMO-708	Arabic (ASMO 708)
720	DOS-720	Arabic (DOS)
737	ibm737	Greek (DOS)
775	ibm775	Baltic (DOS)
850	ibm850	Western European (DOS)
852	ibm852	Eastern European (DOS)
855	IBM855	OEM Cyrillic
857	ibm857	Turkish (DOS)
858	IBM00858	OEM Multilingual Latin I

860	IBM860	Portuguese (DOS)
861	ibm861	Icelandic (DOS)
862	DOS-862	Hebrew (DOS)
863	IBM863	French, Canada (DOS)
864	IBM864	Arabic (864)
865	IBM865	Nordic (DOS)
866	cp866	Cyrillic (DOS)
869	ibm869	Greek, modern (DOS)
870	IBM870	IBM EBCDIC (Multilingual Latin-2)
874	windows-874	Thai (Windows)
875	cp875	IBM EBCDIC (Greek, modern)
932	shift_jis	Japanese (Shift-JIS)
936	gb2312	Simplified Chinese (GB2312)
949	ks_c_5601-1987	Korean
950	big5	Chinese (Traditional) (Big5)
1026	IBM1026	IBM EBCDIC (Turkish, Latin-5)
1047	IBM01047	IBM Latin-1
1140	IBM01140	IBM EBCDIC (USA-Canada-European)
1141	IBM01141	IBM EBCDIC (Germany-European)
1142	IBM01142	IBM EBCDIC (Denmark-Norway-European)
1143	IBM01143	IBM EBCDIC (Finland-Sweden-European)
1144	IBM01144	IBM EBCDIC (Italian-European)
1145	IBM01145	IBM EBCDIC (Spain-European)
1146	IBM01146	IBM EBCDIC (Great Britain-European)
1147	IBM01147	IBM EBCDIC (France-European)
1148	IBM01148	IBM EBCDIC (International-European)
1149	IBM01149	IBM EBCDIC (Icelandic-European)
1200	utf-16	Unicode
1201	unicodeFFFE	Unicode (Big Endian)

1250	windows-1250	Central European (Windows)
1251	windows-1251	Cyrillic (Windows)
1252	Windows-1252	Western European (Windows)
1253	windows-1253	Greek (Windows)
1254	windows-1254	Turkish (Windows)
1255	windows-1255	Hebrew (Windows)
1256	windows-1256	Arabic (Windows)
1257	windows-1257	Baltic (Windows)
1258	windows-1258	Vietnamese (Windows)
1361	Johab	Korean (Johab)
10000	macintosh	Western European (Mac)
10001	x-mac-japanese	Japanese (Mac)
10002	x-mac-chinesetrad	Chinese traditional (Mac)
10003	x-mac-korean	Korean (Mac)
10004	x mac-arabic	Arabic (Mac)
10005	x-mac-hebrew	Hebrew (Mac)
10006	x-mac-greek	Greek (Mac)
10007	x-mac-cyrillic	Cyrillic (Mac)
10008	x-mac-chinesesimp	Chinese simplified (Mac)
10010	x-mac-romanian	Romanian (Mac)
10017	x-mac-ukrainian	Ukrainian (Mac)
10021	x-mac-thai	Thai (Mac)
10029	x-mac-ce	Central European (Mac)
10079	x-mac-icelandic	Icelandic (Mac)
10081	x-mac-turkish	Turkish (Mac)
10082	x-mac-croatian	Croatian (Mac)
12000	utf-32	Unicode (UTF-32)
12001	utf-32BE	Unicode (UTF-32-Big Endian)
20000	x-Chinese-CNS	Chinese traditional (CNS)
20001	x-cp20001	TCA Taiwan

20002	x-Chinese-Eten	Chinese traditional (Eten)
20003	x-cp20003	IBM5550 Taiwan
20004	x-cp20004	TeleText Taiwan
20005	x-cp20005	Wang Taiwan
20105	x-IA5	Western European (IA5)
20106	x-IA5-German	German (IA5)
20107	x-IA5-Swedish	Swedish (IA5)
20108	x-IA5-Norwegian	Norwegian (IA5)
20127	us-ascii	US-ASCII
20261	x-cp20261	T.61
20269	x-cp20269	ISO-6937
20273	IBM273	IBM EBCDIC (Germany)
20277	IBM277	IBM EBCDIC (Denmark-Norway)
20278	IBM278	IBM EBCDIC (Finland-Sweden)
20280	IBM280	IBM EBCDIC (Italy)
20284	IBM284	IBM EBCDIC (Spain)
20285	IBM285	IBM EBCDIC (Great Britain)
20290	IBM290	IBM EBCDIC (Japanese Katakana)
20297	IBM297	IBM EBCDIC (France)
20420	IBM420	IBM EBCDIC (Arabic)
20423	IBM423	IBM EBCDIC (Greek)
20424	IBM424	IBM EBCDIC (Hebrew)
20833	x-EBCDIC-KoreanExtended	IBM EBCDIC (Korean, extended)
20838	IBM-Thai	IBM EBCDIC (Thai)
20866	koi8-r	Cyrillic (KOI8-R)
20871	IBM871	IBM EBCDIC (Icelandic)
20880	IBM880	IBM EBCDIC (Cyrillic, Russian)
20905	IBM905	IBM EBCDIC (Turkish)
20924	IBM00924	IBM Latin-1
20932	EUC-JP	Japanese (JIS 0208-1990 and 0212-1990)

20936	x-cp20936	GB2312-80 Chinese (simple)
20949	x-cp20949	Korean Wansung
21025	cp1025	IBM EBCDIC (Cyrillic, Serbian-Bulgarian)
21866	koi8-u	Cyrillic (KOI8-U)
28591	iso-8859-1	Western European (ISO)
28592	iso-8859-2	Central European (ISO)
28593	iso-8859-3	Latin 3 (ISO)
28594	iso-8859-4	Baltic (ISO)
28595	iso-8859-5	Cyrillic (ISO)
28596	iso-8859-6	Arabic (ISO)
28597	iso-8859-7	Greek (ISO)
28598	iso-8859-8	Hebrew (ISO-Visual)
28599	iso-8859-9	Turkish (ISO)
28603	iso-8859-13	Estonian (ISO)
28605	iso-8859-15	Latin 9 (ISO)
29001	x-Europa	Europe
38598	iso-8859-8-i	Hebrew (ISO-Logical)
50220	iso-2022-jp	Japanese (JIS)
50221	csISO2022JP	Japanese (JIS, 1 Byte Kana allowed)
50222	iso-2022-jp	Japanese (JIS, 1 Byte Kana allowed - SO/SI)
50225	iso-2022-kr	Korean (ISO)
50227	x-cp50227	ISO-2022 Chinese (simplified)
51932	euc-jp	Japanese (EUC)
51936	EUC-CN	Chinese simplified (EUC)
51949	euc-kr	Korean (EUC)
52936	hz-gb-2312	Chinese simplified (HZ)
54936	GB18030	GB18030 Chines simplified
57002	x-iscii-de	ISCII Devanagari
57003	x-iscii-be	ISCII Bengali

57004	x-iscii-ta	ISCII Tamil
57005	x-iscii-te	ISCII Telugu
57006	x-iscii as	ISCII Assamese
57007	x-iscii-or	ISCII Oriya
57008	x-iscii-ka	ISCII Kannada
57009	x-iscii-ma	ISCII Malayalam
57010	x-iscii-gu	ISCII Gujarati
57011	x-iscii-pa	ISCII Punjabi
65000	utf-7	Unicode (UTF-7)
65001	utf-8	Unicode (UTF-8)

Eof

Determines whether the current position is the end of the file.

Eof	
SYNTAX	Eof()
RETURN	TRUE = Eof reached. FALSE = Eof not reached.

Flush

Clears all buffers and writes them to the file.

Flush	
SYNTAX	Flush()

Open

Opens a file to read a text.

Open

SYNTAX

Open(fileName)

Open(fileName, encoding)

ARGUMENTS

fileName — The file to open.

encoding — You can use the following values as encoding parameters.

- Empty string. In this case, the default encoding (ANSI codepage of the operating system) is used.
- Code page, for example, "1252"
- Name, for example, "windows-1252"
- Friendly name, for example, "Western European" (Windows)

Possible encoding parameters

Code page	name	Display name
37	IBM037	IBM EBCDIC (USA-Canada)
437	IBM437	OEM USA
500	IBM500	IBM EBCDIC (International)
708	ASMO-708	Arabic (ASMO 708)
720	DOS-720	Arabic (DOS)
737	ibm737	Greek (DOS)
775	ibm775	Baltic (DOS)
850	ibm850	Western European (DOS)
852	ibm852	Eastern European (DOS)
855	IBM855	OEM Cyrillic
857	ibm857	Turkish (DOS)
858	IBM00858	OEM Multilingual Latin I
860	IBM860	Portuguese (DOS)
861	ibm861	Icelandic (DOS)
862	DOS-862	Hebrew (DOS)

863	IBM863	French, Canada (DOS)
864	IBM864	Arabic (864)
865	IBM865	Nordic (DOS)
866	cp866	Cyrillic (DOS)
869	ibm869	Greek, modern (DOS)
870	IBM870	IBM EBCDIC (Multilingual Latin-2)
874	windows-874	Thai (Windows)
875	cp875	IBM EBCDIC (Greek, modern)
932	shift_jis	Japanese (Shift-JIS)
936	gb2312	Simplified Chinese (GB2312)
949	ks_c_5601-1987	Korean
950	big5	Chinese (Traditional) (Big5)
1026	IBM1026	IBM EBCDIC (Turkish, Latin-5)
1047	IBM01047	IBM Latin-1
1140	IBM01140	IBM EBCDIC (USA-Canada-European)
1141	IBM01141	IBM EBCDIC (Germany-European)
1142	IBM01142	IBM EBCDIC (Denmark-Norway-European)
1143	IBM01143	IBM EBCDIC (Finland-Sweden-European)
1144	IBM01144	IBM EBCDIC (Italian-European)
1145	IBM01145	IBM EBCDIC (Spain-European)
1146	IBM01146	IBM EBCDIC (Great Britain-European)
1147	IBM01147	IBM EBCDIC (France-European)
1148	IBM01148	IBM EBCDIC (International-European)
1149	IBM01149	IBM EBCDIC (Icelandic-European)
1200	utf-16	Unicode
1201	unicodeFFFE	Unicode (Big Endian)
1250	windows-1250	Central European (Windows)
1251	windows-1251	Cyrillic (Windows)
1252	Windows-1252	Western European (Windows)

1253	windows-1253	Greek (Windows)
1254	windows-1254	Turkish (Windows)
1255	windows-1255	Hebrew (Windows)
1256	windows-1256	Arabic (Windows)
1257	windows-1257	Baltic (Windows)
1258	windows-1258	Vietnamese (Windows)
1361	Johab	Korean (Johab)
10000	macintosh	Western European (Mac)
10001	x-mac-japanese	Japanese (Mac)
10002	x-mac-chinesetrad	Chinese traditional (Mac)
10003	x-mac-korean	Korean (Mac)
10004	x mac-arabic	Arabic (Mac)
10005	x-mac-hebrew	Hebrew (Mac)
10006	x-mac-greek	Greek (Mac)
10007	x-mac-cyrillic	Cyrillic (Mac)
10008	x-mac-chinesesimp	Chinese simplified (Mac)
10010	x-mac-romanian	Romanian (Mac)
10017	x-mac-ukrainian	Ukrainian (Mac)
10021	x-mac-thai	Thai (Mac)
10029	x-mac-ce	Central European (Mac)
10079	x-mac-icelandic	Icelandic (Mac)
10081	x-mac-turkish	Turkish (Mac)
10082	x-mac-croatian	Croatian (Mac)
12000	utf-32	Unicode (UTF-32)
12001	utf-32BE	Unicode (UTF-32-Big Endian)
20000	x-Chinese-CNS	Chinese traditional (CNS)
20001	x-cp20001	TCA Taiwan
20002	x-Chinese-Eten	Chinese traditional (Eten)
20003	x-cp20003	IBM5550 Taiwan
20004	x-cp20004	TeleText Taiwan

20005	x-cp20005	Wang Taiwan
20105	x-IA5	Western European (IA5)
20106	x-IA5-German	German (IA5)
20107	x-IA5-Swedish	Swedish (IA5)
20108	x-IA5-Norwegian	Norwegian (IA5)
20127	us-ascii	US-ASCII
20261	x-cp20261	T.61
20269	x-cp20269	ISO-6937
20273	IBM273	IBM EBCDIC (Germany)
20277	IBM277	IBM EBCDIC (Denmark-Norway)
20278	IBM278	IBM EBCDIC (Finland-Sweden)
20280	IBM280	IBM EBCDIC (Italy)
20284	IBM284	IBM EBCDIC (Spain)
20285	IBM285	IBM EBCDIC (Great Britain)
20290	IBM290	IBM EBCDIC (Japanese Katakana)
20297	IBM297	IBM EBCDIC (France)
20420	IBM420	IBM EBCDIC (Arabic)
20423	IBM423	IBM EBCDIC (Greek)
20424	IBM424	IBM EBCDIC (Hebrew)
20833	x-EBCDIC-KoreanExtended	IBM EBCDIC (Korean, extended)
20838	IBM-Thai	IBM EBCDIC (Thai)
20866	koi8-r	Cyrillic (KOI8-R)
20871	IBM871	IBM EBCDIC (Icelandic)
20880	IBM880	IBM EBCDIC (Cyrillic, Russian)
20905	IBM905	IBM EBCDIC (Turkish)
20924	IBM00924	IBM Latin-1
20932	EUC-JP	Japanese (JIS 0208-1990 and 0212-1990)
20936	x-cp20936	GB2312-80 Chinese (simple)
20949	x-cp20949	Korean Wansung
21025	cp1025	IBM EBCDIC (Cyrillic, Serbian-

		Bulgarian)
21866	koi8-u	Cyrillic (KOI8-U)
28591	iso-8859-1	Western European (ISO)
28592	iso-8859-2	Central European (ISO)
28593	iso-8859-3	Latin 3 (ISO)
28594	iso-8859-4	Baltic (ISO)
28595	iso-8859-5	Cyrillic (ISO)
28596	iso-8859-6	Arabic (ISO)
28597	iso-8859-7	Greek (ISO)
28598	iso-8859-8	Hebrew (ISO-Visual)
28599	iso-8859-9	Turkish (ISO)
28603	iso-8859-13	Estonian (ISO)
28605	iso-8859-15	Latin 9 (ISO)
29001	x-Europa	Europe
38598	iso-8859-8-i	Hebrew (ISO-Logical)
50220	iso-2022-jp	Japanese (JIS)
50221	csISO2022JP	Japanese (JIS, 1 Byte Kana allowed)
50222	iso-2022-jp	Japanese (JIS, 1 Byte Kana allowed - SO/SI)
50225	iso-2022-kr	Korean (ISO)
50227	x-cp50227	ISO-2022 Chinese (simplified)
51932	euc-jp	Japanese (EUC)
51936	EUC-CN	Chinese simplified (EUC)
51949	euc-kr	Korean (EUC)
52936	hz-gb-2312	Chinese simplified (HZ)
54936	GB18030	GB18030 Chines simplified
57002	x-iscii-de	ISCII Devanagari
57003	x-iscii-be	ISCII Bengali
57004	x-iscii-ta	ISCII Tamil
57005	x-iscii-te	ISCII Telugu
57006	x-iscii as	ISCII Assamese

57007	x-iscii-or	ISCII Oriya
57008	x-iscii-ka	ISCII Kannada
57009	x-iscii-ma	ISCII Malayalam
57010	x-iscii-gu	ISCII Gujarati
57011	x-iscii-pa	ISCII Punjabi
65000	utf-7	Unicode (UTF-7)
65001	utf-8	Unicode (UTF-8)

ReadLine

Reads a line of characters from the current file and returns the data as a string.

ReadLine	
SYNTAX	ReadLine()
RETURN	The next line of the file or an empty string if the end of the file is reached.

ReadToEnd

Reads the file from the current position to the end of the file.

ReadToEnd	
SYNTAX	ReadToEnd()
RETURN	The rest of the file from the current position to the end as a string. If the current position is the end of the file, an empty string ("") returns.

Write

Writes a string in a file.

Write		
SYNTAX	Write(text)	
ARGUMENTS	text	The string to write

WriteFormat

Writes a formatted string using the same semantics as FormatStr.

WriteFormat		
SYNTAX	WriteFormat(formatRule, formatParams)	
ARGUMENTS	formatRule	Formatting rule
	formatParams	Object(s) to write to the formatted string

WriteLine

Writes a string followed by a character for line termination in a file.

WriteLine		
SYNTAX	WriteLine(text)	
ARGUMENTS	text	The string to write

WriteLineFormat

Writes a formatted string and a new line using the same semantics as FormatStr.

WriteLineFormat

SYNTAX	WriteLineFormat(formatRule, formatParams)
ARGUMENTS	formatRule — Formatting rule formatParams — Object(s) to write to the formatted string

Properties of the Class MonaLisaTextFile

Property	Description
FileName	Returns the file name of the open file or an empty string if no file is open.

Examples

```
; Create DOS object
GetObject("DOS", "MLDos")
;
;-------------------------------------
; Create file and open to write
;-------------------------------------
try
File = DOS.CreateTextFile()
File.Create("c:\temp\test.txt")
File.WriteLine("Line 1")
File.Write("Line ")
File.Write("2")
File.WriteLine("")
File.WriteLine("Line 3")
lineNo = 4
File.WriteLineFormat("Line {0}", lineNo)
```

```
File.WriteFormat("Line {0}", lineNo + 1)
File.WriteLine("")
Finally
File.Close()
end-try
;
;---------------------------------------------------------------------
; Create file and open to write, line break is LineFeed ($0A)
;---------------------------------------------------------------------
try
File = DOS.CreateTextFile(DOS.NewLineLF)
File.Create("c:\temp\test.txt")
File.WriteLine("Line 1")
File.Write("Line ")
File.Write("2")
File.WriteLine("")
File.WriteLine("Line 3")
; change to a new line break CRLF ($0D $0A)
File.NewLine = DOS.NewLineCRLF
lineNo = 4
File.WriteLineFormat("Line {0}", lineNo)
File.WriteFormat("Line {0}", lineNo + 1)
File.WriteLine("")
Finally
File.Close()
end-try

;
;----------------------
; open file to append
```

```
;----------------------
try
File = DOS.CreateTextFile()
File.Append("c:\temp\test.txt")
File.WriteLine("Line 6")
Finally
File.Close()
end-try
;
;-------------------------------------------------------------
; Open file to append, line break by default CRLF ($0D $0A)
;-------------------------------------------------------------
try
File = DOS.CreateTextFile()
; Set line break to LineFeed ($0A)
File.NewLine = DOS.NewLineLF
File.Append("c:\temp\test.txt")
File.WriteLine("Line 6")
Finally
File.Close()
end-try

;--------------------
; Open file to read
;--------------------
try
File = DOS.CreateTextFile()
File.Open("c:\temp\test.txt")
;
while (not File.Eof())
```

```
StringListAdd("file", File.ReadLine())
end-while
Finally
File.Close()
end-try
text = StringListGetCommaText("file")
;
;-----------------------------
; Read complete text of a file
;-----------------------------
text2 = DOS.FileReadAllText("c:\temp\test.txt")
;
;--------------------------
; Append text to a file
;--------------------------
DOS.FileAppendText("c:\temp\test.txt", "last line")
```

XML Processing

About MLXmlDocument

The MonaLisa object MLXmlDocument enables you to access the class XmlDocument from the .NET Framework. This class enables you to read and write in XML files.

Refer to the Microsoft Developer Network website for more information on the XmlDocumentClass.

To load the object, use the MonaLisa function GetObject.

Example

```
GetObject("XML", "MLXmlDocument")
```

Additional Functions of the Class MLXmlDocument

GetAttribute

Reads the value of an attribute.

GetAttribute

SYNTAX	GetAttribute(xpath, attributeName)
ARGUMENTS	xpath — XPath expression to select the node. obj — Name of the attribute belonging to the node select with the xpath parameter.
RETURN	Value of the attribute or an empty string, if the attribute does not exist.

GetAttributeOfNode

Reads the attribute value of a specific XML element.

GetAttributeOfNode

SYNTAX	GetAttributeOfNode(xmlNode, attributeName)
ARGUMENTS	xmlNode — Instance of an XML element from which to read the attribute. attributeName — Attribute name.
RETURN	Value of the attribute or an empty string, if the attribute does not exist.

GetNamespaceManager

Returns a namespace manager specific to this XML document.

GetNamespaceManager

SYNTAX	GetNamespaceManager()
RETURN	XML namespace manager object for this MLXmlDocument instance.

GetXpathValue

Reads the value of an attribute or element.

GetXPathValue		
SYNTAX	GetXPathValue(xmlNode, xpath)	
ARGUMENTS	xmlNode	XML node.
	xpath	XPath
RETURN	Value of the attribute or element or an empty string if the attribute or element does not exist.	

Load

Loads XML data from a URL.

Load		
SYNTAX	Without namespaces preservation option: Load(string filename) With namespaces preservation option: Load(string filename, bool stripOutNamespaces)	
ARGUMENTS	filename	URL of the file you want to load, which contains a XML document. The URL may be a local file path or a HTTP URL (web address).
	stripOutNamespaces	TRUE = Removes namespace prefixes and definitions. FALSE = Preserves namespace prefixes and definitions.

LoadXml

Loads an XML document from a string.

LoadXml	
SYNTAX	Without namespaces preservation option: LoadXML(string xml) With namespaces preservation option: LoadXML(string xml, bool stripOutNamespaces)

ARGUMENTS	xml	String which contains the XML document.
	stripOutNamespaces	TRUE = Removes namespace prefixes and definitions. FALSE = Preserves namespace prefixes and definitions.

SetAttribute

Set the attribute value of a specific XML element specified in an XPath expression.

SetAttribute		
SYNTAX	SetAttribute(xpath, attributeName, value) SetAttribute(xpath, attributeName, value, create)	
ARGUMENTS	xpath	XPATH expression to select the node
	attributeName	Name of the attribute belonging to the node select with the parameter XPATH
	value	New attribute value
	create	TRUE = The attribute is created if it does not already exist. FALSE = Attribute is not created.
RETURN	TRUE = Attribute value is set. FALSE = Attribute value could not be set.	

SetAttributeOfNode

Set the attribute value of a specific XML element.

SetAttribute	
SYNTAX	SetAttributeOfNode(xmlNode, attributeName, value) SetAttributeOfNode(xmlNode, attributeName, value, create)

ARGUMENTS	xmlNode	Instance of an XML element whose attribute you want to set
	attributeName	Name of the attribute belonging to the node select with the parameter XPATH
	value	New attribute value
	create	TRUE = The attribute is created if it does not already exist. FALSE = Attribute is not created.
RETURN	TRUE = Attribute value is set. FALSE = Attribute value could not be set.	

Sample Scripts

Sample Script: Reading and Modifying Attributes

```
; create XmlDocument Object
GetObject("xml", "MLXmlDocument")
;
; load XML from file
xml.Load("C:\Temp\bookstore.xml")
;
; or read XML from a string
;xml.LoadXml("<bookstore><book><title lang=""en"">Italienisch kochen</title></book></bookstore>")
;
; Get root element of the XML document
root = xml.DocumentElement
;
; Using XPath select all title nodes in an XmlNodeList
nodeList = root.SelectNodes("/bookstore/book/title")
;
; get an node enumerator.
```

```
nodeListEnumerator = nodeList.GetEnumerator()
;
; Iterate over all items...
readNext = nodeListEnumerator.MoveNext()
while (readNext)
  ;
  ; get current item node
  node = nodeListEnumerator.Current
  ;
  ;Language attribute
  BookLang = xml.GetAttributeOfNode(node, "lang")
  ; and read value of the title node
  BookTitle = node.InnerText
  ;
  Protocol("Title: {0}, language: {1}", 5,BookTitle, BookLang)
  ;
  ; set Language attribute to German
  xml.SetAttributeOfNode(node, "lang", "de")
  ;
  ; next item node
  readNext = nodeListEnumerator.MoveNext()
  ;
end-while
;
; set language attribute of the 2. Book node to English with XPath
xml.SetAttribute("/bookstore/book[2]/title", "lang", "en")
;
; save as a test
xml.Save("C:\Temp\bookstore_test.xml")
;
```

```
;assign to a string variable
xmlString = xml.OuterXml
;
;Example for GetXPathValue()
FirstTitle = xml.GetXPathValue(root, "/bookstore/book/title[1]")
CoverTextOfFirstTitle = xml.GetXPathValue(root, "/bookstore/book/covertext[1]")
CommentOfFirstTitle = xml.GetXPathValue(root, "/bookstore/book[1]/comment()")
Protocol("GetXPathValue() 1. Title: {0}, cover-text: {1}, comment: {2}", 5, FirstTitle, CoverTextOfFirstTitle,
CommentOfFirstTitle)
;
```

The XML used in the example:

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book>
    <title lang="en">Italian cooking</title>
    <covertext>

    </covertext>
    <author>Giada De Florenz</author>
    <year>2001</year>
    <price>30.00</price>
    <!-- comment for 1st book -->
  </book>
  <book>
    <title lang="de">Harry Schlotter</title>
    <covertext>

    </covertext>
    <author>J R. Bowling</author>
    <year>2005</year>
```

```
    <price>29.99</price>
    <!-- comment for 2nd book -->
  </book>
</bookstore>
```

Sample Script: Create an XML File

```
; create XmlDocument Object
GetObject("xml", "MLXmlDocument")
;
; create XML Declaration
decl = xml.CreateXmlDeclaration("1.0", "UTF-8", "")
xml.AppendChild(decl)
;
; create and insert new Fields-Element
fields = xml.CreateElement("FIELDS")
xml.AppendChild(fields)
;
; create new Field-Element
field = xml.CreateElement("FIELD")
;
; Attribute name=test (only if the attribute already exists!)
;xml.SetAttributeOfNode(field, "name", "test")
;
;Example: ; create and insert CDATA
cdata = xml.CreateCDataSection("test value")
field.AppendChild(cdata)
;
; get Attributes-Collection of the Field-Node
attributes = field.Attributes
;
; create Name-Attribute
```

```
name = xml.CreateAttribute("name")
; set value of the Attribute
name.Value = "test"
; and insert the Attribute of the Attributes-Collection
attributes.Append(name)
;
; the new Field-Element can now be inserted into the Fields-Element
fields.AppendChild(field)
;
; and save
xml.Save("c:\temp\fields.xml")
;
;assign to a string variable
xmlString = xml.OuterXml
;
Result
<?xml version="1.0" encoding="UTF-8"?>
<FIELDS>
  <FIELD name="test"></FIELD>
</FIELDS>
```

Ini-File Processing

About MLIniFile

The MonaLisa object MLIniFile enables you to perform read and write operations on **.ini** files. To load the object, use the MonaLisa function GetObject.

Example

```
GetObject("IniFile","MLIniFile")
```

Functions of the Class MLInifile

Close

Closes an open .ini file.

Close	
SYNTAX	Close()
EXAMPLE	GetObject("IniFile","MLIniFile") ... IniFile.Close()

CopySection

Copies the specified section and identifiers in the specified .ini file. If either the identifier or the section do not exist, they will be created.

CopySection		
SYNTAX	CopySection(section, destFile)	
ARGUMENTS	section	The name of the section.
	destFile	Name of the '.ini' file in which the section copies in the following form. `[Drive\Path\]File name`
EXAMPLE	`GetObject("IniFile","MLIniFile")` `IniFile.CopySection("Section_Test", "d:\test\MLTestIniFile_COPY.ini")`	

DeleteKey

Deletes the specified identifier from the respective section.

DeleteKey	
SYNTAX	DeleteKey(section, key)

ARGUMENTS	section	Name of the section.
	key	Identifier you want to delete.
RETURN	TRUE = Successful FALSE = Unsuccessful	
EXAMPLE	GetObject("IniFile","MLIniFile") ok = IniFile.DeleteKey("Delete-Section","Delete")	

EraseSection

Deletes the specified section from an .ini file.

EraseSection		
SYNTAX	EraseSection(section)	
ARGUMENTS	section	Name of the section to delete.
RETURN	TRUE = Successful FALSE = Unsuccessful	
EXAMPLE	GetObject("IniFile","MLIniFile") ok = IniFile.EraseSection("Delete-Section")	

GetSectionNames

Returns a comma separated string containing all identifiers of a defined section.

GetSectionNames		
SYNTAX	GetSectionNames(section)	
ARGUMENTS	section	Name of the section.

RETURN	Comma separated string of the identifiers.
EXAMPLE	GetObject("IniFile","MLIniFile") sectionNames = IniFile.GetSectionNames("Section_Test")

Open

Opens the specified .**ini** file for read or write operations.

Open

SYNTAX	Open(fileName)
ARGUMENTS	fileName — Name of the .ini file in the form of `[Drive\Path\]File name`
RETURN	TRUE = Successful FALSE = Not successful.
EXAMPLE	GetObject("IniFile","MLIniFile") ... ok = IniFile.Open("d:\test\MLTestIniFile.ini")

ReadBool

Reads the Boolean operator of the specified identifier from the respective section.

ReadBool

SYNTAX	ReadBool(section, key)
ARGUMENTS	section — Name of the section. key — Identifier

RETURN	Value read.
EXAMPLE	GetObject("IniFile","MLIniFile") ... Flag = IniFile.ReadBool("Section_Test","IsNewCustomer")

ReadInteger

Reads the integer value of the specified identifier from the respective section.

ReadInteger

SYNTAX	ReadInteger(section, key)
ARGUMENTS	section — Name of the section. key — Identifier
RETURN	Value read.
EXAMPLE	GetObject("IniFile","MLIniFile") ... Year = IniFile.ReadInteger("Section_Test","Year")

ReadSectionToCommatext

Reads all identifier=value pairs from the specified section and returns them in a comma-separated string.

ReadSectionToCommatext

SYNTAX	ReadSectionToCommatext(section)
ARGUMENTS	section — Name of the section.
RETURN	Comma separated string containing all identifier=value pairs in the following form. `"identifier=value1,identifier2=value2"`

EXAMPLE	GetObject("IniFile","MLIniFile") ... section = IniFile.ReadSectionToCommatext(

ReadString

Reads the string of the specified identifier from the respective section.

ReadString	
SYNTAX	ReadString(section, key)
ARGUMENTS	section — Name of the section. key — Identifier
RETURN	Value read.
EXAMPLE	GetObject("IniFile","MLIniFile") ... Name = IniFile.ReadString("Section_Test","Name")

WriteBool

Writes the Boolean value as the value of the specified identifier in the respective section.

If the section does not exist, it is created.

If the identifier does not exist, it is created.

WriteBool	
SYNTAX	WriteBool(section, key, value)
ARGUMENTS	section — Name of the section

	key	Identifier
	value	The Boolean value to write

EXAMPLE

```
GetObject("IniFile","MLIniFile")
...
IniFile.WriteBool("Section_Test ","IsNewCustomer",TRUE)
```

WriteInteger

Writes the integer value as the value of the specified identifier in the respective section.

If the section does not exist, it is created.

If the identifier does not exist, it is created.

WriteInteger

SYNTAX

```
WriteInteger(section, key, value)
```

ARGUMENTS

	section	Name of the section
	key	Identifier
	value	Integer value to write

EXAMPLE

```
GetObject("IniFile","MLIniFile")
...
IniFile.WriteInteger("Section_Test ","Year",1998)
```

WriteSectionFromCommatext

Writes all identifier=value pairs to the specified section.

If either the identifier or the section do not exist, they are created.

WriteSectionFromCommatext

SYNTAX	WriteSectionFromCommatext(section, commaText)
ARGUMENTS	section — Name of the section commaText — Comma-separated string in the form. `"identifier=value1,identifier2=value2"`
EXAMPLE	GetObject("IniFile","MLIniFile") ... IniFile.WriteSectionFromCommatext("Section_Test","NAME=MyName,Company=PA")

WriteString

Writes the string as a value of the specified identifier in the respective section.

If the section does not exist, it is created.

If the identifier does not exist, it is created.

WriteString

SYNTAX	WriteString(section, key, value)
ARGUMENTS	section — Name of the section key — Identifier value — Integer value to write
EXAMPLE	GetObject("IniFile","MLIniFile") ... IniFile.WriteString("Section_Test ","Company","MyCompanyName")

MWS Functions

About MLMwsClient

The MonaLisa object MLMwsClient enables you to use MWS functions. The object provides all functions necessary for communication with the MWSProcessService, general helper function, and various properties required. For a description of the structure and meaning of the various options XMLs, XML results, and result codes, refer to the *Content Composer Web Services Developer's Guide*.

To load the object, use the MonaLisa function GetObject.

Example

GetObject("MwsClient","MLMwsClient")

Functions of the Class MLMwsClient

General Information

A description of the structure and meaning of the various options XMLs and XML results or result codes can be found in the documentation of the web service module WS Process (PDC.WebServices.chm).

The property LastXmlResult can be used to read the XML result written by the methods directly after a method has been called.

It is not part of the method signature, even though for individual methods it is described as such in the comments.

The same applies to the property LastBase64DocumentString of the method Doc_GetFile_Mime with regard to the Parameters doc

ClearCache

Clears the cache of the RepositoryRuntimeService.

ClearCache	
SYNTAX	ClearCache()
RETURN	0 = Successful <>0 = Unsuccessful
EXAMPLE	ClearCache()

Doc_GetFile_Mime

Returns a file that manages the business object.

Doc_GetFile_Mime

SYNTAX Doc_GetFile_Mime(processId, fileName)

ARGUMENTS

processId	ID of the process whose business object you want to manage.
fileName	File name.
doc	Base64 coded string, representing the content of the file.
xmlResult	Empty when OK.

RETURN 0 = Successful

<>0 = Unsuccessful

Doc_SetFile_Mime

Writes a file that manages the business object.

Doc_SetFile_Mime

SYNTAX Doc_SetFile_Mime(processId, fileName, doc)

ARGUMENTS

processId	ID of the process whose business object you want to manage.
fileName	File name.
doc	Base64 coded string that represents the content of the file.
xmlResult	Empty when OK.

RETURN 0 = Successful

<>0 = Unsuccessful

GetList

Returns a list of all currently existing processes.

GetList

SYNTAX GetList(startAt, max, options)

ARGUMENTS

startAt	Start index.
max	Maximum number of processes to return.
options	Use this parameter to specify which processes you want to include in the retrieved process list.
xmlResult	An MwsProcessList XML when successful, otherwise an MwsError.

RETURN 0 = Successful.

<> = Not successful.

Obj_GetStructure

Returns the structure of the referenced object.

Obj_GetStructure

SYNTAX Obj_GetStructure(processId, rootRef, maxLevel)

ARGUMENTS

processId	ID of the process the object references.
rootRef	Object reference within the process.
maxLevel	-1 = All 0 = Just the object. >0 = The structure including all child elements at the n-th level.
xmlResult	Object XML depending on the component.

RETURN 0 = Successful

<>0 = Not successful.

Obj_SetStructure

Sets the structure of the business object.

Obj_SetStructure		
SYNTAX	Obj_SetStructure(processId, content, rootRef, maxLevel)	
ARGUMENTS	processId	ID of the process the object references.
	content	The business object information to set.
	rootRef	Object reference whose structure returns.
	maxLevel	Determines how many levels return. -1 = All 0 = Just the object. >0 = The structure including all child elements at the n-th level.
	xmlResult	Object structure of the business object, if OK.
RETURN	0 = Successful <>0 = Not successful.	

Obj_Toggle

Toggles the status of an object.

Obj_Toggle		
SYNTAX	Obj_Toggle(processId, objRef, rootRef, maxLevel)	
ARGUMENTS	processId	ID of the process the object references.
	content	Reference of the object to toggle.
	rootRef	Object reference whose structure returns.

	maxLevel	Determines how many levels return. -1 = All 0 = Just the object. >0 = The structure including all child elements at the n-th level.
	xmlResult	Object structure of the business object, if OK.
RETURN	0 = Successful <>0 = Not successful.	

Ping

Returns the string "MWS Process Service".

Ping	
SYNTAX	Ping()
RETURN	"MWS Process Service"

Process_Close

Closes a process with the passed in status.

Process_Close		
SYNTAX	Process_Close(processId, status, options)	
ARGUMENTS	processId	Process ID.
	status	Free text status definition.
	options	Close options XML with title and description.
	xmlResult	Empty when OK.

RETURN	0 = Successful. <> 0 = Not successful.

Process_Create

Creates a new MWS process.

Process_Create

SYNTAX Process_Create(processId, mSystem, mType, mTitle, objectIndex, objectName, objectType, objectData, options)

ARGUMENTS

processId	If the process ID is not passed in, a new one is created.
mSystem	System OID of the system the object to load resides in.
mType	Free text to define the process type.
mTitle	Free text to define a title for the process.
objectIndex	Index of the object to load. If this parameter is defined, the objectName parameter is ignored.
objectName	Name of the loaded object. This parameter is only taken into account if objectindex is not specified.
objectType	Type of object to load. Currently only "P" for bundles is possible.
objectData	Optional XML data for a bundle. The data is assigned to the first passive selection in the OnNewStart script.
options	Options XML for the MWS process.
xmlResult	If the process is created, either the bundle structure or process information returns. The bundle structure returns if a start command passes to the options.

RETURN 0 = Successful.

<> 0 = Not successful.

Process_Delete

Deletes an open process and frees its server resources.

Process_Delete		
SYNTAX	Process_Delete(processId)	
ARGUMENTS	processId	ID of the process to delete.
	xmlResult	Empty when OK, otherwise MwsError.
RETURN	0 = Successful. <> 0 = Not successful.	

Process_Forward

Forwards a process.

Process_Forward		
SYNTAX	Process_Forward(processID, destUser, destUsergroup, options)	
ARGUMENTS	processID	Process ID.
	destUser	Name of the target user the process forwards to.
	destUsergroup	Name of the target group the process forwards to.
	options	Close options XML with title and description.
	xmlResult	Empty when OK.
RETURN	0 = Successful. <> 0 = Not successful.	

Process_GetInfo

Returns information about the process.

Process_GetInfo

SYNTAX Process_GetInfo(processId, infoType, options)

ARGUMENTS

processId	ID of the process for which the information returns.
infoType	Comma separated string containing the information to retrieve. This information is the predefined types PROCESSINFO and POOLVARS. The MWS process is dependent and is loaded and evaluated by the OnGetInfo script.
options	Currently not supported.
xmlResult	ProcessInfo XML when OK.

RETURN 0 = Successful.

<> 0 = Not successful.

Process_GetLastError

Returns the last error of a process.

Process_GetLastError

SYNTAX Process_GetLastError(processID)

ARGUMENTS

processId	ID of the process for which the last error retrieves.
xmlResult	Last error XML when successful.

RETURN 0 = Successful.

<> 0 = Not successful.

Process_Open

Resumes processing of a process and sets the user name to that of the current user.

Process_Open		
SYNTAX	Process_Open(processId, options)	
ARGUMENTS	processId	ID of the process to open.
	xmlResult	Currently not supported.
RETURN	0 = Successful. <> 0 = Not successful.	

Process_SetData

Sets XML data for the business object.

Process_SetData		
SYNTAX	Process_SetData(processId, dataType, data)	
ARGUMENTS	processId	ID of the process.
	dataType	Name to assign to the data. Currently refers to the name of the data reference.
	data	The XML data to set.
	xmlResult	Empty when OK.
RETURN	0 = Successful. <> 0 = Not successful.	

Process_Start

Starts a process action.

Process_Start

SYNTAX	Process_Start(processId, command, options)
ARGUMENTS	processId — Process ID of the process where the action starts. command — Contains the command to pass in. The names of the commands derive from the InvokeActivities script of the MWS process. options — You can use this parameter to set various options of the process, to set process activities, and to pass in variables. xmlResult — The structure of the bundle when successful.
RETURN	0 = Successful. <> 0 = Not successful.

Srv_GetInfo

Returns information about the MWS server.

Srv_GetInfo

SYNTAX	Srv_GetInfo(options)
ARGUMENTS	options — Currently not used xmlResult — MWS server info XML if OK, otherwise MWSError
RETURN	0 = OK <> 0 = Error
EXAMPLE	GetObject("MwsClient","MLMwsClient") … … options = "" resultCode = MwsClient.Srv_GetInfo(options) xmlResult = MwsClient.LastXmlResult

	Protocol(resultCode, 8) Protocol(xmlResult, 8)

Class Helper Functions

LoadBase64StringFromBinFile

Loads the specified file and converts the contents in a Base64 string.

LoadBase64StringFromBinFile	
SYNTAX	LoadBase64StringFromBinFile(binFileName)
ARGUMENTS	binFileName File name.
RETURN	Base64 coded string of the file. If no file exists, an empty string returns.
EXAMPLE	newDocString = MwsClient.LoadBase64StringFromBinFile("c:\temp\MyDoc_changed.docx")

SaveBase64StringToBinFile

Writes a Base64 string converted to a byte array to a binary file.

SaveBase64StringToBinFile	
SYNTAX	SaveBase64StringToBinFile(base64Content, binFileName)
ARGUMENTS	base64Content Base64Content binFileName File name
RETURN	TRUE = File written FALSE = File not written
EXAMPLE	if (myDocString <> "") ok = MwsClient.SaveBase64StringToBinFile(myDocString, "c:\temp\MyDoc_org.docx")

```
end-if
```

Properties of the Class MLMwsClient

Property	Description

LastBase64DocumentString	Allows you to read the last Base64 string retrieved.
	Read-only property.
	The MWSService function Doc_GetFile_Mime returns the Base64 coded string of the specified document in addition to an integer result code and the XML result string.
	Example
	`myXmlResult = ""`
	`resCode = MwsClient.Doc_GetFile_Mime(myProcessId, filename)`
	`myDocString = MwsClient.LastBase64DocumentString`
	`myXmlResult = MwsClient.LastXmlResult`

LastProcessId	Allows you to read the last process ID retrieved.
	Read-only property.
	The MWSService function Process_Create enables you to define a process ID or, if the corresponding parameter is passed as an empty string, enables you to automatically generate a process ID.
	This is irrespective of whether the process ID was predefined or created automatically.
	The property is set with the corresponding process ID when the Process_Create function call was successful.
	Example
	`myXmlResult = ""`
	`myProcessId = ""`
	`options = ""`
	`resCode = MwsClient.Process_Create(myProcessId, "dm1", "MyType", "MyTitle", "", "MyBundleName",`

	"P", "", options) myProcessId = MwsClient.LastProcessId myXmlResult = MwsClient.LastXmlResult
LastXmlResult	Allows you to read the last XmlResult string retrieved. Read-only property. Most MWSService functions return both an integer result code and an XmlResult as string. **Example** myXmlResult = "" resCode = MwsClient.Process_Open(myProcessId, "") myXmlResult = MwsClient.LastXmlResult
MwsUrl	Allows you to define the URL of the MWS server. The property must be set with the address of the MWS service before the first call of the MWS service is made. Internally, when the property is being set, a previously opened service channel is closed so that the following call to the MWS service can create a new service channel. If this property is not defined, the first call to the MWS service tries to read the required URL for the MWS service from the key appSettings (key="mws"). An exception is thrown if no corresponding setting can be found here either. **Example** MwsClient.MwsUrl = "http://localhost:8011/mws/mwsprocess"

Sample Script MLMwsClient

```
;********************* MLMwsClient ****************************************
;
GetObject("MwsClient","MLMwsClient")
;
```

```
MwsClient.MwsUrl = "http://localhost:8011/mws/mwsprocess"
;
dummy = MwsClient.Ping()
Protocol(dummy, 8)
;
; On setting the URL, a previously created ServiceChannel is closed
;MwsClient.MwsUrl = "http://localhost:8011/mws/mwsprocess"
;dummy = MwsClient.Ping()
;Protocol(dummy, 8)
;MwsClient.MwsUrl = "http://localhost:8011/mws/mwsprocess"
;dummy = MwsClient.Ping()
;Protocol(dummy, 8)
;
myXmlResult = ""
myProcessId = ""
options = "<mws><mwsaction name=""DATASELECTION"" configured = ""1"" ><selparams><selparam
name=""Partnernumber"" >0815</selparam><selparam
name=""test""></selparam></selparams></mwsaction></mws>"
res = MwsClient.Process_Create(myProcessId, "dm1", "MyType", "MyTitle", "", "Application2", "P", "",
options)
myProcessId = MwsClient.LastProcessId
myXmlResult = MwsClient.LastXmlResult
Protocol(result, 8)
Protocol(res, 8)
;
if (res = 0)
  myXmlResult = ""
  res = MwsClient.Process_Start(myProcessId, "CREATE_DOCUMENTS", "")
  myXmlResult = MwsClient.LastXmlResult
  Protocol(result, 8)
  Protocol(res, 8)
```

```
end-if
;
if (myXmlResult <> "")
  filename = ""
  GetObject("xml", "MLXmlDocument")
  xml.LoadXml(myXmlResult)
  filename = xml.GetAttribute("/mws/process/object/object[1]", "filename")
  FreeObject("xml")
end-if
;
if (filename <> "")
  res = MwsClient.Doc_GetFile_Mime(myProcessId, filename)
  myDocString = MwsClient.LastBase64DocumentString
  Protocol(myDocString, 8)
  Protocol(res, 8)
  if (myDocString <> "")
    ok = MwsClient.SaveBase64StringToBinFile(myDocString, "c:\temp\MyDoc_org.docx")
    Protocol(ok, 8)
  end-if
end-if
;
; now alter document, upload it and then download it again for manual check
if (filename <> "")
  myNewDocString = MwsClient.LoadBase64StringFromBinFile("c:\temp\MyDoc_changed.docx")
  Protocol(myNewDocString, 8)
  if (myNewDocString <> "")
    res = MwsClient.Doc_SetFile_Mime(myProcessId, filename, myNewDocString)
    Protocol(res, 8)
    if (res = 0)
      res = MwsClient.Doc_GetFile_Mime(myProcessId, filename)
```

```
      myDocString = MwsClient.LastBase64DocumentString
      Protocol(myDocString, 8)
      Protocol(res, 8)
      if (myDocString <> "")
        ok = MwsClient.SaveBase64StringToBinFile(myDocString, "c:\temp\MyDoc_downChanged.docx")
        Protocol(ok, 8)
      end-if
    end-if
  end-if
end-if
;
; upload any file and download it again
filename= "c:\temp\MMDMO_33.B.migrate.xml"
myFileString = MwsClient.LoadBase64StringFromBinFile(filename)
Protocol(myFileString, 8)
res = MwsClient.Doc_SetFile_Mime(myProcessId, filename, myFileString)
Protocol(res, 8)
if (res = 0)
  res = MwsClient.Doc_GetFile_Mime(myProcessId, filename)
  myFileString = MwsClient.LastBase64DocumentString
  Protocol(myFileString, 8)
  Protocol(res, 8)
  if (myFileString <> "")
    ok = MwsClient.SaveBase64StringToBinFile(myFileString, "c:\temp\MMDMO_DOWNLOAD_
33.B.migrate.xml")
    Protocol(ok, 8)
  end-if
end-if
;
;all remaining methods possible
```

```
if (myProcessId <> "" )
  myXmlResult = ""
  res = MwsClient.Obj_GetStructure(myProcessId, "", 1)
  myXmlResult = MwsClient.LastXmlResult
  Protocol(result, 8)
  Protocol(res, 8)
;
  myXmlResult = ""
  res = MwsClient.Obj_Toggle(myProcessId, "3C331B86-8552-4AD6-A242-29106BFBCA21", "", 1)
  myXmlResult = MwsClient.LastXmlResult
  Protocol(result, 8)
  Protocol(res, 8)
;
  myXmlResult = ""
  res = MwsClient.Process_SetData(myProcessId, "My_nonExisting_BundleDataId",
"<data>MyDataString</data>")
  myXmlResult = MwsClient.LastXmlResult
  Protocol(result, 8)
  Protocol(res, 8)
;
  myXmlResult = ""
  res = MwsClient.Process_GetLastError(myProcessId)
  myXmlResult = MwsClient.LastXmlResult
  Protocol(result, 8)
  Protocol(res, 8)
;
  myXmlResult = ""
  res = MwsClient.ClearCache()
  myXmlResult = MwsClient.LastXmlResult
  Protocol(result, 8)
```

```
  Protocol(res, 8)
;
  myXmlResult = ""
  res = MwsClient.Process_GetInfo(myProcessId, "PROCESSINFO", "")
  myXmlResult = MwsClient.LastXmlResult
  Protocol(result, 8)
  Protocol(res, 8)
;
  content = "<mws type=""OBJECT"" version=""1""><process><object action=""includedoc""
name=""TestDocument"" target_reference="""" position=""after"" /></process></mws>"
  myXmlResult = ""
  res = MwsClient.Obj_SetStructure(myProcessId, content, "", 1)
  myXmlResult = MwsClient.LastXmlResult
  Protocol(result, 8)
  Protocol(res, 8)
;
  myXmlResult = ""
  res = MwsClient.Process_GetInfo(myProcessId, "PROCESSINFO", "")
  myXmlResult = MwsClient.LastXmlResult
  Protocol(result, 8)
  Protocol(res, 8)
;
  options = "<mws version=""2""><title>Forwarded to Mike</title><description>Description of the
task</description></mws>"
  myXmlResult = ""
  res = MwsClient.Process_Forward(myProcessId, "Mike", "", options)
  myXmlResult = MwsClient.LastXmlResult
  Protocol(result, 8)
  Protocol(res, 8)
;
```

```
  options = "<mws><processlist><item name=""listuser"" include=""Mike"" /><item
name=""shortprocdesc"" include=""30"" /></processlist></mws>"
  myXmlResult = ""
  res = MwsClient.Process_GetList(0, 100, options)
  myXmlResult = MwsClient.LastXmlResult
  Protocol(result, 8)
  Protocol(res, 8)
;
  ;IntegerDate = date.DateStringToInt64("ddMMyyyyHHmmss", "21012010174853")
  ;DateStr = date.Int64ToDateString("MM-dd-yyyy HH:mm:ss", IntegerDate)
;
  options = "<mws><processlist><item name=""listuser"" include=""Mike"" /><item
name=""shortprocdesc"" include=""30"" /><item name=""forwarded"" include=""1"" /><item
name=""nonforwarded"" include=""0"" /></processlist></mws>"
  myXmlResult = ""
  res = MwsClient.Process_GetList(0, 100, options)
  myXmlResult = MwsClient.LastXmlResult
  Protocol(result, 8)
  Protocol(res, 8)
;
  myXmlResult = ""
  res = MwsClient.Process_Open(myProcessId, "")
  myXmlResult = MwsClient.LastXmlResult
  Protocol(result, 8)
  Protocol(res, 8)
;
  myXmlResult = ""
  res = MwsClient.Process_GetTransDocsList(myProcessId, "")
  myXmlResult = MwsClient.LastXmlResult
  Protocol(result, 8)
  Protocol(res, 8)
```

```
;
  myXmlResult = ""
  res = MwsClient.Process_Close(myProcessId, 0, "")
  myXmlResult = MwsClient.LastXmlResult
  Protocol(result, 8)
  Protocol(res, 8)
;
  myXmlResult = ""
  res = MwsClient.Process_Delete(myProcessId)
  myXmlResult = MwsClient.LastXmlResult
  Protocol(result, 8)
  Protocol(res, 8)
;
end-if
;
myXmlResult = ""
res = MwsClient.Process_GetList(0, 100, "")
myXmlResult = MwsClient.LastXmlResult
Protocol(result, 8)
Protocol(res, 8)
;
FreeObject("MwsClient")]]></scriptcode>
;
```

Sample Script: Create a Process

```
GetObject("MwsClient","MLMwsClient")
MwsClient.MwsUrl = "http://localhost:8011/mws/mwsprocess"
result = MwsClient.Ping()
Protocol(result, 8)
;
```

```
myXmlResult = ""

myProcessId = ""

options = ""

res = MwsClient.Process_Create(myProcessId, "MySystemOId", "MyType", "MyTitle", "",
"MyBundleName", "P", "", options)

myProcessId = MwsClient.LastProcessId

myXmlResult = MwsClient.LastXmlResult

Protocol(result, 8)

Protocol(res, 8)

;

FreeObject("MwsClient")
```

SOAP Webservices

About SOAP Web Services

The MonaLisa object MLWsClient enables you to call SOAP WebServices. The object provides all functions necessary for communication with the web service, together with the required properties.

To load the object, use the MonaLisa function GetObject.

Example

```
GetObject("WS","MLWsClient")
```

Requirements

Powershell 3.0

Windows Powershell 3.0 must be installed to be able to use the class.

Powershell 3.0 might already be installed on newer Windows systems.

Network Configuration

Depending on the network configuration it is necessary to configure Composition Studio to use a specific proxy server.

Complete the following steps to do this:

1. Close **Content Composer Studio**.
2. Open the file **Composer.Studio.exe.config**.
3. Insert the following section as a child of the node **<configuration>**.

```
<system.net>
```

```
    <defaultProxy enabled="true">
      <proxy proxyaddress="http://10.12.9.100:8080" bypassonlocal="true" />
    </defaultProxy>
  </system.net>
```

4. Replace the address **http://10.12.9.100** with the address of valid proxy server.
5. Save the modified file.
6. Start **Content Composer Studio**.

For more information, refer to http://msdn.microsoft.com/en-us/library/kd3cf2ex.aspx.

Properties of the Class MLWsClient

Property	Description
Namespace	The namespace to be used while runtime-generating types of the web service proxy. Set this property before setting the WsdlUri property.

PropertiesCount	Contains the number of properties of the function called.
	Example
	`GetObject("WsClient", "MLWsClient")`
	`PropertiesCount = WsClient.PropertiesCount`

RefParamsCount	Contains the number of reference parameters of the function called.
	Example
	`GetObject("WsClient", "MLWsClient")`
	`RefParamsCount = WsClient.RefParamsCount`

ResultObject	The result of the web service function call as an object.
	You can directly access properties and functions of this object.
	Example
	`GetObject("WsClient", "MLWsClient")`
	`Result = WsClient.CallMethod("EchoObject")`
	`ResObj = WsClient.ResultObject`

WebserviceProxyModule	The module that contains all the runtime-generated types of the web service proxy.

WsdlUri	Contains the URI of the required web service.
	Example
	`GetObject("WsClient", "MLWsClient")`
	`WsClient.WsdlUri = "http://localhost:8888/WebserviceHost"`

Functions of the Class MLWsClient

CallMethod

Calls the web service function.

CallMethod	
SYNTAX	CallMethod(methodName, args)
ARGUMENTS	methodName — Name of the web service function to call args — Arguments for the web service function call
RETURN	Result of the web service function call as a string. If result is a complex type, it is a serialized XML string. This can be processed using the MLXmlDocument functions.
EXAMPLE	GetObject("WsClient", "MLWsClient") Result = WsClient.CallMethod("Ping")

DefineWebProxy

Sets the web proxy.

DefinePrinterByOId	
SYNTAX	DefineWebProxy(uri, bypassList) DefineWebProxy(uri, bypassOnLocal, bypassList)

DefineWebProxy(uri, domain, proxyUser, password, code, bypassOnLocal, bypassList)

ARGUMENTS

uri	URI of the web proxy you want to use.
domain	Domain of the proxy user.
proxyUser	Name of the proxy user.
password	Password of the proxy user.
code	Encoded password of the proxy user.
bypassOnLocal	TRUE = The proxy is bypassed on local addresses. FALSE = The proxy is not bypassed on local addresses.
bypassList	Addresses that bypass the proxy.

EXAMPLE

```
GetObject("WsClient", "MLWsClient")
WsClient.DefineWebProxy("http://10.12.9.101:8080")
```

GetArrayLength

Returns the array size (length) of a one-dimensional array transferred object or the object stored in ResultObject.

GetArrayLength

SYNTAX

```
GetArrayLength()
GetArrayLength(obj)
```

ARGUMENTS

obj	The array object to check.

RETURN

The size (lenght) of the array.

EXAMPLE

```
GetObject("WsClient", "MLWsClient")
WsClient.GetArrayLength()
```

GetEnumValue

Returns the enumeration value of a specific type by a specific untyped value.

GetEnumValue		
SYNTAX	GetEnumValue(type, value)	
ARGUMENTS	type	Enumeration type.
	value	Untyped value of the enumeration value.
RETURN	Typed enumeration value.	

GetObjectFromArray

Returns a single object from a one-dimensional array object transferred or stored in ResultObject specified by the index.

GetObjectFromArray		
SYNTAX	GetObjectFromArray(arrayIndex) GetObjectFromArray(obj, arrayIndex)	
ARGUMENTS	obj	The object to check.
	arrayIndex	The index of the object to return from the array.
RETURN	The object from the array.	
EXAMPLE	GetObject("WsClient", "MLWsClient") WsClient.GetObjectFromArray(XmlDoc,0)	

GetObjectStringFromArray

Returns a string serialization of an array item of the transferred object or the object stored in ResultObject.

GetObjectStringFromArray

SYNTAX	GetObjectStringFromArray(arrayIndex) GetObjectStringFromArray(obj, arrayIndex)
ARGUMENTS	obj — The object to check. arrayIndex — The index of the object to return from the array.
RETURN	XML serialized string of the object.
EXAMPLE	GetObject("WsClient", "MLWsClient") test = WsClient.GetObjectStringFromArray(ResObj,0)

GetPropertyNameByIndex

Determines the name of the property by the index.

GetPropertyNameByIndex

SYNTAX	GetPropertyNameByIndex(idx)
ARGUMENTS	idx — Index of the property.
RETURN	Name of the property.

EXAMPLE	GetObject("WsClient", "MLWsClient")
	a = 0
	while (a < PropertiesCount)
	  pName = WsClient.GetPropertyNameByIndex(a)
	  pValue = WsClient.GetPropertyValueByIndex(a)
	  refObjValue = WsClient.GetPropertyValueByIndexAsObject(a)
	  pValueByName = WsClient.GetPropertyValueByName(pName)
	  refObjValueByName = WsClient.GetPropertyValueByNameAsObject(pName)
	  Protocol("Property[{0}] {1}: {2}", 8, a, pName, pValue)
	  a = a + 1
	end-while

GetPropertyValueByIndex

Determines the value of the property by the index.

GetPropertyValueByIndex

SYNTAX GetPropertyValueByIndex(idx)

ARGUMENTS

idx	Index of the property.

RETURN Value of the property.

EXAMPLE

```
GetObject("WsClient", "MLWsClient")
a = 0
while (a < PropertiesCount)
  pName = WsClient.GetPropertyNameByIndex(a)
  pValue = WsClient.GetPropertyValueByIndex(a)
  refObjValue = WsClient.GetPropertyValueByIndexAsObject(a)
  pValueByName = WsClient.GetPropertyValueByName(pName)
  refObjValueByName = WsClient.GetPropertyValueByNameAsObject(pName)
  Protocol("Property[{0}] {1}: {2}", 8, a, pName, pValue)
  a = a + 1
end-while
```

GetPropertyValueByIndexAsObject

Determines the value of the property by the index.

GetPropertyValueByIndexAsObject

SYNTAX GetPropertyValueByIndexAsObject(idx)

ARGUMENTS

idx	Index of the property.

RETURN	Value of the property as object.
EXAMPLE	GetObject("WsClient", "MLWsClient") a = 0 while (a < PropertiesCount) pName = WsClient.GetPropertyNameByIndex(a) pValue = WsClient.GetPropertyValueByIndex(a) refObjValue = WsClient.GetPropertyValueByIndexAsObject(a) pValueByName = WsClient.GetPropertyValueByName(pName) refObjValueByName = WsClient.GetPropertyValueByNameAsObject(pName) Protocol("Property[{0}] {1}: {2}", 8, a, pName, pValue) a = a + 1 end-while

GetPropertyValueByName

Determines the value of the property by its name.

GetPropertyValueByName	
SYNTAX	GetPropertyValueByName(propName)
ARGUMENTS	propName — Name of the property.
RETURN	Value of the property.
EXAMPLE	GetObject("WsClient", "MLWsClient") a = 0 while (a < PropertiesCount) pName = WsClient.GetPropertyNameByIndex(a) pValue = WsClient.GetPropertyValueByIndex(a) refObjValue = WsClient.GetPropertyValueByIndexAsObject(a) pValueByName = WsClient.GetPropertyValueByName(pName)

```
    refObjValueByName = WsClient.GetPropertyValueByNameAsObject(pName)
    Protocol("Property[{0}] {1}: {2}", 8, a, pName, pValue)
    a = a + 1
  end-while
```

GetPropertyValueByNameAsObject

Determines the value of the property by its name.

GetPropertyValueByNameAsObject

SYNTAX GetPropertyValueByNameAsObject(propName)

ARGUMENTS

propName	Name of the property.

RETURN Value of the property as object.

EXAMPLE

```
GetObject("WsClient", "MLWsClient")
a = 0
while (a < PropertiesCount)
  pName = WsClient.GetPropertyNameByIndex(a)
  pValue = WsClient.GetPropertyValueByIndex(a)
  refObjValue = WsClient.GetPropertyValueByIndexAsObject(a)
  pValueByName = WsClient.GetPropertyValueByName(pName)
  refObjValueByName = WsClient.GetPropertyValueByNameAsObject(pName)
  Protocol("Property[{0}] {1}: {2}", 8, a, pName, pValue)
  a = a + 1
end-while
```

GetRefParamNameByIndex

Determines the name of the reference parameter by its index.

GetRefParamNameByIndex

SYNTAX GetRefParamNameByIndex(idx)

ARGUMENTS

idx	Index of the reference parameter.

RETURN Name of the reference parameter.

EXAMPLE

```
GetObject("WsClient", "MLWsClient")
a = 0
while (a < RefParamsCount)
  pName = wscObj.GetRefParamNameByIndex(a)
  pValue = wscObj.GetRefParamValueByIndex(a)
  Protocol("(RefParam) {0}: {1}", 8,pName, pValue)
  a = a + 1
end-while
```

GetRefParamValueByIndex

Determines the value of the reference parameter by its index.

GetRefParamValueByIndex

SYNTAX GetRefParamValueByIndex(idx)

ARGUMENTS

idx	Index of the reference parameter.

RETURN Value of the reference parameter.

EXAMPLE

```
GetObject("WsClient", "MLWsClient")
a = 0
while (a < RefParamsCount)
  pName = wscObj.GetRefParamNameByIndex(a)
  pValue = wscObj.GetRefParamValueByIndex(a)
```

```
    Protocol("(RefParam) {0}: {1}", 8,pName, pValue)
    a = a + 1
  end-while
```

GetRefParamValueByIndexAsObject

Determines the value of the reference parameter by its index.

GetRefParamValueByIndexAsObject	
SYNTAX	GetRefParamValueByIndexAsObject(idx)
ARGUMENTS	idx Index of the reference parameter.
RETURN	Value of the reference parameter as object.
EXAMPLE	GetObject("WsClient", "MLWsClient") a = 0 while (a < RefParamsCount) pName = WsClient.GetRefParamNameByIndex(a) pValue = WsClient.GetRefParamValueByIndex(a) refObjValue = WsClient.GetRefParamValueByIndexAsObject(a) pValueByName = WsClient.GetRefParamValueByName(pName) refObjValueByName = WsClient.GetRefParamValueByNameAsObject(pName) Protocol("(RefParam) {0}: {1}", 8,pName, pValue) a = a + 1 end-while

GetRefParamValueByName

Determines the value of the reference parameter by its name.

GetRefParamValueByName

SYNTAX	GetRefParamValueByName(refParamName)
ARGUMENTS	refParamName Name of the reference parameter.
RETURN	Value of the reference parameter.
EXAMPLE	GetObject("WsClient", "MLWsClient") a = 0 while (a < RefParamsCount) pName = WsClient.GetRefParamNameByIndex(a) pValue = WsClient.GetRefParamValueByIndex(a) refObjValue = WsClient.GetRefParamValueByIndexAsObject(a) pValueByName = WsClient.GetRefParamValueByName(pName) refObjValueByName = WsClient.GetRefParamValueByNameAsObject(pName) Protocol("(RefParam) {0}: {1}", 8,pName, pValue) a = a + 1 end-while

GetRefParamValueByNameAsObject

Determines the value of the reference parameter by its name.

GetRefParamValueByNameAsObject	
SYNTAX	GetRefParamValueByNameAsObject(refParamName)
ARGUMENTS	refParamName Name of the reference parameter.
RETURN	Value of the reference parameter as object.
EXAMPLE	GetObject("WsClient", "MLWsClient") a = 0 while (a < RefParamsCount) pName = WsClient.GetRefParamNameByIndex(a)

```
    pValue = WsClient.GetRefParamValueByIndex(a)
    refObjValue = WsClient.GetRefParamValueByIndexAsObject(a)
    pValueByName = WsClient.GetRefParamValueByName(pName)
    refObjValueByName = WsClient.GetRefParamValueByNameAsObject(pName)
    Protocol("(RefParam) {0}: {1}", 8,pName, pValue)
    a = a + 1
  end-while
```

GetWebserviceProxyTypeFromString

Returns a type instance of a web service proxy module specific typestring.

GetWebserviceProxyTypeFromString

SYNTAX	GetWebserviceProxyTypeFromString(type)
ARGUMENTS	type — The typestring that returns a type instance.
RETURN	The web service proxy module specific type.

IsArrayObject

Checks whether the specified object or the object in ResultObject is an array.

IsArrayObject

SYNTAX	IsArrayObject() IsArrayObject(obj)
ARGUMENTS	obj — The object to check.
RETURN	TRUE = The object is an array. FALSE = The object is not an array.

EXAMPLE	GetObject("WsClient", "MLWsClient") bTestResult = WsClient.IsArrayObject() bTestResult = WsClient.IsArrayObject(XmlDoc)

IsObjectNull

Checks whether the specified object or the object in ResultObject is null.

IsObjectNull	
SYNTAX	IsObjectNull() IsObjectNull(obj)
ARGUMENTS	obj The object to check.
RETURN	TRUE = The object is null. FALSE = The object is not null.
EXAMPLE	GetObject("WsClient", "MLWsClient") if(WsClient.IsObjectNull()<>true) Raise("NullCheckMethodFail","ResultObject should be null at this point. IsObjectNull function failed.") end-if

SetParameter

Sets the parameters for the web service functions.

SetParameter	
SYNTAX	SetParameter(paramName, value)
ARGUMENTS	paramName Parameter name value Parameter value

EXAMPLE	streamDoc.SetInputBin(streamdoc.PageCount - 1, "testBin")

Examples

Example Login / Logout

```
GetObject("WsClient", "MLWsClient")
; Wsdl-Uri Mws
WsClient.WsdlUri = "http://localhost:8011/mwsbasic?WSDL"

; CallMethod 'Ping'
Result = WsClient.CallMethod("Ping")

; Evaluation
if (Result = "")
  Raise("Mws Ping", "Result not as expected")
end-if

Protocol("PingResult: {0}", 8, Result)
System.Console.WriteLine("PingResult: {0}", Result)

; CallMethod 'Login'
WsClient.SetParameter("userName", "Administrator")
WsClient.SetParameter("password", "")
WsClient.SetParameter("code", "YYICUj9uSKJqDDQV8Vj9PUH45pv0dhTUpNh4xXn5giA=")
WsClient.SetParameter("options", "")
WsClient.SetParameter("LoginResult", "")
WsClient.SetParameter("LoginResultSpecified", "")
WsClient.SetParameter("sessionID", "")
WsClient.SetParameter("xmlResult", "")
Result = WsClient.CallMethod("Login")
```

```
RefParamsCount = WsClient.RefParamsCount
PropertiesCount = WsClient.PropertiesCount

; Evaluation
sessionID = WsClient.GetRefParamValueByName("sessionID")
if (sessionID = "")
  Raise("Mws Login", "No sessionID received")
end-if

a = 0
while (a < RefParamsCount)
  pName = WsClient.GetRefParamNameByIndex(a)
  pValue = WsClient.GetRefParamValueByIndex(a)
  Protocol("(RefParam) {0}: {1}", 8,pName, pValue)
  System.Console.WriteLine("(RefParam) {0}: {1}", pName, pValue)
  a = a + 1
end-while

Protocol("sessionID: {0}", 8, sessionID)
System.Console.WriteLine("sessionID: {0}", sessionID)
loginResult = WsClient.GetRefParamValueByName("LoginResult")
Protocol("LoginResult: {0}", 8, loginResult)
System.Console.WriteLine("LoginResult: {0}", loginResult)

; CallMethod 'Logout'
WsClient.SetParameter("sessionID", sessionId)
WsClient.SetParameter("LogoutResult", "")
WsClient.SetParameter("LogoutResultSpecified", "")
WsClient.SetParameter("xmlResult", "")
result = WsClient.CallMethod("Logout")
```

```
; Evaluation
a = 0
while (a < RefParamsCount)
  pName = WsClient.GetRefParamNameByIndex(a)
  pValue = WsClient.GetRefParamValueByIndex(a)
  Protocol("(RefParam) {0}: {1}", 8,pName, pValue)
  System.Console.WriteLine("(RefParam) {0}: {1}", pName, pValue)
  a = a + 1
end-while

logoutResult = WsClient.GetRefParamValueByName("LogoutResult")
Protocol("LogoutResult: {0}", 8, logoutResult)
System.Console.WriteLine("LogoutResult: {0}", logoutResult)

; Wsdl-Uri MwsRws
WsClient.WsdlUri = "http://localhost:8010/mwsbasic?WSDL"

; CallMethod 'Ping'
Result = WsClient.CallMethod("Ping")
RefParamsCount = WsClient.RefParamsCount
PropertiesCount = WsClient.PropertiesCount

; Evaluation
if (Result = "")
  Raise("MwsRws Ping", "Result not as expected")
end-if
Protocol("PingResult: {0}", 8, Result)
System.Console.WriteLine("PingResult: {0}", Result)
```

```
; Wsdl-Uri Mur
WsClient.WsdlUri = "http://localhost:8010/murbasic?WSDL"

; CallMethod 'Ping'
Result = WsClient.CallMethod("Ping")
RefParamsCount = WsClient.RefParamsCount
PropertiesCount = WsClient.PropertiesCount

; Evaluation
if (Result = "")
  Raise("Mur Ping", "Result not as expected")
end-if
Protocol("PingResult: {0}", 8, Result)
System.Console.WriteLine("PingResult: {0}", Result)

; CallMethod 'GetUsersOfRole'
WsClient.SetParameter("roleName", "AdminRole")
WsClient.SetParameter("GetUsersOfRoleResult", "0")
WsClient.SetParameter("GetUsersOfRoleResultSpecified", "false")
WsClient.SetParameter("lastError", "")
WsClient.SetParameter("users", "")
Result = WsClient.CallMethod("GetUsersOfRole")
RefParamsCount = WsClient.RefParamsCount
PropertiesCount = WsClient.PropertiesCount

; Evaluation
Protocol("Result of GetUsersOfRole: {0}", 8, Result)
System.Console.WriteLine("Result of GetUsersOfRole: {0}", Result)
a = 0
while (a < RefParamsCount)
```

```
  pName = WsClient.GetRefParamNameByIndex(a)
  pValue = WsClient.GetRefParamValueByIndex(a)
  Protocol("(RefParam) {0}: {1}", 8,pName, pValue)
  System.Console.WriteLine("(RefParam) {0}: {1}", pName, pValue)
  a = a + 1
end-while

test = WsClient.GetRefParamValueByName("users")
if (test = "")
  Raise("Mur GetUsersOfRole RefParam users", "Result not as expected")
end-if
Protocol("users: {0}", 8, test)
System.Console.WriteLine("users: {0}", test)

; destructor
FreeObject("WsClient")
```

Example: Calling Web Services

```
GetObject("wscObj", "MLWsClient")
GetObject("xmlDoc","MLXmlDocument")

; WebProxy-Definition
; If Webproxy is required, specify it here
wscObj.DefineWebProxy("http://10.12.9.101:8080")

; Wsdl-Uri
wscObj.WsdlUri = "http://www.webservicex.net/ConvertTemperature.asmx?WSDL"

; CallMethod 'ConvertTemp'
wscObj.SetParameter("Temperature", "0")
```

```
wscObj.SetParameter("FromUnit", "kelvin")
wscObj.SetParameter("ToUnit", "degreeCelsius")
Result = wscObj.CallMethod("ConvertTemp")
RefParamsCount = wscObj.RefParamsCount
PropertiesCount = wscObj.PropertiesCount

; Output
Protocol("Result: {0}", 8, Result)
a = 0
while (a < RefParamsCount)
  pName = wscObj.GetRefParamNameByIndex(a)
  pValue = wscObj.GetRefParamValueByIndex(a)
  Protocol("(RefParam) {0}: {1}", 8,pName, pValue)
  a = a + 1
end-while

a = 0
while (a < PropertiesCount)
  pName = wscObj.GetPropertyNameByIndex(a)
  pValue = wscObj.GetPropertyValueByIndex(a)
  Protocol("Property[{0}] {1}: {2}", 8, a, pName, pValue)
  a = a + 1
end-while
if (Result<>-273.15)
  Raise("TemperatureConversionFailed","The Temperature conversion test failed! Expected: -273,15 Result:
" + Result)
end-if
ParamErrorEncountered = false

try
```

```
; wscObj.SetParameter("Temperature", "0")
  wscObj.SetParameter("FromUnit", "kelvin")
  wscObj.SetParameter("ToUnit", "degreeCelsius")
  Result = wscObj.CallMethod("ConvertTemp")
  OnError
  Protocol("WebService Call failed. This is expected!",8,"")
  ParamErrorEncountered = true
end-try

if(ParamErrorEncountered = false)
  Raise("WebserviceCallFailTestFailed","The previous WebService call should have failed due to a missing parameter, but it succeeded!")
end-if

ExpectedTestErrorEncountered = false
try
  wscObj.GetArrayLength()
  OnError
  ExpectedTestErrorEncountered = true
end-try

if(ExpectedTestErrorEncountered = false)
  Raise("ExpectedExtensionMethodHasnotfailed","The GetArrayLength Method should have failed, but it succeeded!")
end-if
ExpectedTestErrorEncountered = false
try
  wscObj.GetObjectFromArray(-1)
  OnError
  ExpectedTestErrorEncountered = true
```

```
end-try
if(ExpectedTestErrorEncountered = false)
  Raise("ExpectedExtensionMethodHasnotfailed","The GetObjectFromArray Method should have failed because of an invalid index, but it did succeed!")
end-if
ExpectedTestErrorEncountered = false
try
  wscObj.GetObjectFromArray(0)
  OnError
  ExpectedTestErrorEncountered = true
end-try
if(ExpectedTestErrorEncountered = false)
  Raise("ExpectedExtensionMethodHasnotfailed","The GetObjectFromArray Method should have failed because of calling it on a non-array object result, but it did succeed!")
end-if

; CallMethod 'getAirportInformationByISOCountryCode'
wscObj.WsdlUri = "http://www.webservicex.net/airport.asmx?WSDL"
wscObj.SetParameter("CountryAbbrviation", "DE")
Result = wscObj.CallMethod("getAirportInformationByISOCountryCode")
RefParamsCount = wscObj.RefParamsCount
PropertiesCount = wscObj.PropertiesCount

a = 0
while (a < RefParamsCount)
  pName = wscObj.GetRefParamNameByIndex(a)
  pValue = wscObj.GetRefParamValueByIndex(a)
  Protocol("(RefParam) {0}: {1}", 8,pName, pValue)
  a = a + 1
end-while
```

```
a = 0
while (a < PropertiesCount)
  pName = wscObj.GetPropertyNameByIndex(a)
  pValue = wscObj.GetPropertyValueByIndex(a)
  Protocol("Property[{0}] {1}: {2}", 8, a, pName, pValue)
  a = a + 1
end-while

xmlDoc.LoadXml(Result)
nodeList = xmlDoc.SelectNodes("//NewDataSet/Table/AirportCode[.='STR']")
if(nodeList.Count <> 2)
  Raise("WebserviceComplexReturnValueTestFailed","The Webservice Test with a complex return type
failed.")
end-if

; CallMethod 'GetGeoIP'
wscObj.WsdlUri = "http://www.webservicex.net/geoipservice.asmx?WSDL"
wscObj.SetParameter("IPAddress", "127.0.0.1")
Result = wscObj.CallMethod("GetGeoIP")
RefParamsCount = wscObj.RefParamsCount
PropertiesCount = wscObj.PropertiesCount
xmlDoc.LoadXml(Result)
a = 0
while (a < RefParamsCount)
  pName = wscObj.GetRefParamNameByIndex(a)
  pValue = wscObj.GetRefParamValueByIndex(a)
  Protocol("(RefParam) {0}: {1}", 8,pName, pValue)
  a = a + 1
end-while
```

```
a = 0
while (a < PropertiesCount)
  pName = wscObj.GetPropertyNameByIndex(a)
  pValue = wscObj.GetPropertyValueByIndex(a)
  Protocol("Property[{0}] {1}: {2}", 8, a, pName, pValue)
  a = a + 1
end-while

if(wscObj.GetPropertyValueByName("ReturnCode")<>1)
  Raise("WebserviceComplexReturnValueTestExtendedFailed","1. ReturnCode does not match.")
end-if
if(wscObj.GetPropertyValueByName("ReturnCodeDetails")<>"Success")
  Raise("WebserviceComplexReturnValueTestExtendedFailed","1. ReturnCodeDetails does not match.")
end-if
if(wscObj.GetPropertyValueByName("CountryName")<>"Reserved")
  Raise("WebserviceComplexReturnValueTestExtendedFailed","1. CountryName does not match.")
end-if
if(wscObj.GetPropertyValueByName("CountryCode")<>"ZZZ")
  Raise("WebserviceComplexReturnValueTestExtendedFailed","1. CountryCode does not match.")
end-if
ResultObject = wscObj.ResultObject
if(ResultObject.ReturnCode <> wscObj.GetPropertyValueByName("ReturnCode"))
  Raise("WebserviceComplexReturnValueTestExtendedFailed","2. ReturnCode does not match.")
end-if
if(ResultObject.ReturnCodeDetails <> wscObj.GetPropertyValueByName("ReturnCodeDetails"))
  Raise("WebserviceComplexReturnValueTestExtendedFailed","2. ReturnCodeDetails does not match.")
end-if
if(ResultObject.CountryName <> wscObj.GetPropertyValueByName("CountryName"))
  Raise("WebserviceComplexReturnValueTestExtendedFailed","2. CountryName does not match.")
```

```
end-if
if(ResultObject.CountryCode <> wscObj.GetPropertyValueByName("CountryCode"))
  Raise("WebserviceComplexReturnValueTestExtendedFailed","2. CountryCode does not match.")
end-if

namespacemanager = xmlDoc.GetNamespaceManager()
namespacemanager.AddNamespace("def", "http://www.webservicex.net/")
xmlNode = xmlDoc.SelectSingleNode("//GeoIP/def:ReturnCode",namespacemanager)
if(ResultObject.ReturnCode <> xmlNode.InnerText)
  Raise("WebserviceComplexReturnValueTestExtendedFailed","3. ReturnCode does not match.")
end-if

xmlNode = xmlDoc.SelectSingleNode("//GeoIP/def:ReturnCodeDetails",namespacemanager)
if(ResultObject.ReturnCodeDetails <> xmlNode.InnerText)
  Raise("WebserviceComplexReturnValueTestExtendedFailed","3. ReturnCodeDetails does not match.")
end-if

xmlNode = xmlDoc.SelectSingleNode("//GeoIP/def:CountryName",namespacemanager)
if(ResultObject.CountryName <> xmlNode.InnerText)
  Raise("WebserviceComplexReturnValueTestExtendedFailed","3. CountryName does not match.")
end-if

xmlNode = xmlDoc.SelectSingleNode("//GeoIP/def:CountryCode",namespacemanager)
if(ResultObject.CountryCode <> xmlNode.InnerText)
  Raise("WebserviceComplexReturnValueTestExtendedFailed","3. CountryCode does not match.")
end-if
wscObj.SetParameter("IPAddress", "127.0.1")
Result = wscObj.CallMethod("GetGeoIP")
RefParamsCount = wscObj.RefParamsCount
PropertiesCount = wscObj.PropertiesCount
```

```
xmlDoc.LoadXml(Result)

a = 0
while (a < RefParamsCount)
  pName = wscObj.GetRefParamNameByIndex(a)
  pValue = wscObj.GetRefParamValueByIndex(a)
  Protocol("(RefParam) {0}: {1}", 8,pName, pValue)
  a = a + 1
end-while

a = 0
while (a < PropertiesCount)
  pName = wscObj.GetPropertyNameByIndex(a)
  pValue = wscObj.GetPropertyValueByIndex(a)
  Protocol("Property[{0}] {1}: {2}", 8, a, pName, pValue)
  a = a + 1
end-while

if(wscObj.GetPropertyValueByName("ReturnCode")<>0)
  Raise("WebserviceComplexReturnValueTestExtended2Failed","1. ReturnCode does not match.")
end-if
if(wscObj.GetPropertyValueByName("ReturnCodeDetails")<>"Invalid IP address")
  Raise("WebserviceComplexReturnValueTestExtended2Failed","1. ReturnCodeDetails does not match.")
end-if
if(wscObj.GetPropertyValueByName("CountryName")<>"void")
  Raise("WebserviceComplexReturnValueTestExtended2Failed","1. CountryName does not match.")
end-if
if(wscObj.GetPropertyValueByName("CountryCode")<>"void")
  Raise("WebserviceComplexReturnValueTestExtended2Failed","1. CountryCode does not match.")
end-if
```

```
ResultObject = wscObj.ResultObject
if(ResultObject.ReturnCode <> wscObj.GetPropertyValueByName("ReturnCode"))
  Raise("WebserviceComplexReturnValueTestExtended2Failed","2. ReturnCode does not match.")
end-if
if(ResultObject.ReturnCodeDetails <> wscObj.GetPropertyValueByName("ReturnCodeDetails"))
  Raise("WebserviceComplexReturnValueTestExtended2Failed","2. ReturnCodeDetails does not match.")
end-if
; CountryName and CountryCode are not set and therefore the Xml Elements do not exist and the Object Properties result in a MLUnknown Monalisa Object Variable which can not be compared
if(ResultObject.CountryName <> wscObj.GetPropertyValueByName("CountryName"))
  Raise("WebserviceComplexReturnValueTestExtended2Failed","2. CountryName does not match.")
end-if
if(ResultObject.CountryCode <> wscObj.GetPropertyValueByName("CountryCode"))
  Raise("WebserviceComplexReturnValueTestExtended2Failed","2. CountryCode does not match.")
end-if

xmlNode = xmlDoc.SelectSingleNode("//GeoIP/def:ReturnCode",namespacemanager)
if(ResultObject.ReturnCode <> xmlNode.InnerText)
  Raise("WebserviceComplexReturnValueTestExtended2Failed","3. ReturnCode does not match.")
end-if

xmlNode = xmlDoc.SelectSingleNode("//GeoIP/def:ReturnCodeDetails",namespacemanager)
if(ResultObject.ReturnCodeDetails <> xmlNode.InnerText)
  Raise("WebserviceComplexReturnValueTestExtended2Failed","3. ReturnCodeDetails does not match.")
end-if
; CountryName and CountryCode are not set and therefore the Xml Elements do not exist and the Object Properties result in a MLUnknown Monalisa Object Variable which cannot be compared
xmlNode = xmlDoc.SelectSingleNode("//GeoIP/def:CountryName",namespacemanager)
if(ResultObject.CountryName <> xmlNode.InnerText)
```

```
  Raise("WebserviceComplexReturnValueTestExtended2Failed","3. CountryName does not match.")
end-if

xmlNode = xmlDoc.SelectSingleNode("//GeoIP/def:CountryCode",namespacemanager)
if(ResultObject.CountryCode <> xmlNode.InnerText)
  Raise("WebserviceComplexReturnValueTestExtended2Failed","3. CountryCode does not match.")
end-if

; Destructor
FreeObject("wscObj")
FreeObject("xmlDoc")
```

Example - Upload a Document to a Saperion System

```
; To execute this script, make sure the file C:\temp\Saperion.7.5.pdf is available or modify the filePath and fileName variables.
filePath = "C:\temp\Saperion.7.5.pdf"
fileName = "Saperion.7.5.pdf"
saperionWebserviceUriBase = "http://192.168.241.128:8080/scr-webservices/soap/" ; Modify this URL according to the Saperion system you want to access.
saperionAuthenticationServiceUri = saperionWebserviceUriBase + "AuthenticationService?wsdl"
saperionArchiveServiceUri = saperionWebserviceUriBase + "ArchiveService?wsdl"

try
  #region GetObjects
    GetObject("WsClient", "MLWsClient")
    GetObject("dos","MLDos")
    GetObject("array", "MLArrays")
    GetObject("objects","MLObjects")
  #endregion
  loginToken = -1
```

```
  #region CallMethod 'login' ; Login to the Saperion Webservice to get a token which is needed for subsequent calls to the Saperion system
    WsClient.WsdlUri = saperionAuthenticationServiceUri
    WsClient.SetParameter("username", "Administrator") ; Modify the user according to the Saperion system you want to access.
    WsClient.SetParameter("password", "saperion"); Modify the password according to the Saperion system you want to access.
    WsClient.SetParameter("type", 1) ; Logontypes: Index = 1 / Query = 2
    WsClient.SetParameter("mandant", "SYSTEM") ; SYSTEM is the default value for non-client systems.
    loginToken = WsClient.CallMethod("login")
    Protocol("Login Result: {0}",5,loginToken)
  #endregion

  #region CallMethod 'create'
    WsClient.Namespace = "SaperionSoapProxy"
    WsClient.WsdlUri = saperionArchiveServiceUri

    ; The following parameters and methods called are written according to the example specified under the following link: http://portal.saperion.com/docsxml/en/saperion-75-sp2/programming/e-programming_webservices.html

    #region metadatas
      ; Modify the metadata according to the Saperion system you want to access.
      ; The following metadata depends on the target 'definition' used in 'create' call.
      ; In this example the target 'definition' is called 'test' and contains the fields 'name' and 'firstname'
      array.Clear()
      metaDataParameterType = wsClient.GetWebserviceProxyTypeFromString(WsClient.Namespace + ".saWsProperty")
      metaDataInstance = objects.CreateNewInstanceOfType(metaDataParameterType)
      metaDataInstance.name = "name"
      metaDataInstance.value = fileName
```

```
        array.AddToArray(metaDataInstance)
        metaDataInstance = objects.CreateNewInstanceOfType(metaDataParameterType)
        metaDataInstance.name = "firstname"
        metaDataInstance.value = "A File with the name:"
        array.AddToArray(metaDataInstance)
        metaDataArray =array.GetArray(metaDataParameterType)
    #endregion


    #region container ; This object contains the actual document to archive.
        array.Clear()
        contentParameterType = wsClient.GetWebserviceProxyTypeFromString(WsClient.Namespace +
".saWsContent")
        contentInstance = objects.CreateNewInstanceOfType(contentParameterType)
        contentInstance.content = dos.FileReadAllBytes(filePath) ; Specify the full path to the file to upload to
the Saperion archive.
        contentInstance.filename = fileName
        array.AddToArray(contentInstance)
        contentArray =array.GetArray(contentParameterType)
        contentContainer = wsClient.GetWebserviceProxyTypeFromString(WsClient.Namespace +
".saWsContentContainer")
        contentContainerInstance = objects.CreateNewInstanceOfType(contentContainer)
        contentContainerInstance.content = contentArray
        modeType = objects.GetTypeOfObject(contentContainerInstance.mode)
        enumValue = objects.GetEnumValue(modeType,"REPLACE")
        contentContainerInstance.mode = enumValue
    #endregion


    WsClient.SetParameter("token", loginToken)
    WsClient.SetParameter("definition", "test") ; Modify according to the Saperion system accessed: The
target definition to which the document uploads.
    WsClient.SetParameter("metadatas", metaDataArray) ; The metadata.
```

```
    WsClient.SetParameter("container", contentContainerInstance) ; The actual document data.
    WsClient.SetParameter("versionComment", "Some versioning comment via Content Composer
MonaLisa") ; Comment of the document version.
    WsClient.SetParameter("acls", "")
    Result = WsClient.CallMethod("create")
    RefParamsCount = WsClient.RefParamsCount
    PropertiesCount = WsClient.PropertiesCount
  #endregion

  createdDocumentId = WsClient.GetPropertyValueByName("HDoc")
  createdDocumentSysRowId = WsClient.GetPropertyValueByName("sysRowId")
  Protocol("The document was successfully created and has HDoc: {0} (SysRowId: {1})",8,
createdDocumentId,createdDocumentSysRowId)

  #region Debug / Output
    Protocol("Result: {0}", 8, Result)
    a = 0
    while (a < RefParamsCount)
      pName = WsClient.GetRefParamNameByIndex(a)
      pValue = WsClient.GetRefParamValueByIndex(a)
      Protocol("(RefParam) {0}: {1}", 8,pName, pValue)
      a = a + 1
    end-while
    a = 0
    while (a < PropertiesCount)
      pName = WsClient.GetPropertyNameByIndex(a)
      pValue = WsClient.GetPropertyValueByIndex(a)
      Protocol("Property[{0}] {1}: {2}", 8, a, pName, pValue)
      a = a + 1
    end-while
```

```
  #endregion
Finally
  if(loginToken <> -1)
    #region CallMethod 'logoff' ; Log off the Saperion system regardless if the upload was successful
      WsClient.WsdlUri = saperionAuthenticationServiceUri
      WsClient.SetParameter("token", loginToken)
      Result = WsClient.CallMethod("logoff")
      Protocol("Logoff Result: {0}",5,Result)
    #endregion
  end-if

  #region FreeObjects
    FreeObject("WsClient")
    FreeObject("array")
    FreeObject("dos")
    FreeObject("objects")
  #endregion
End-try
```

User Attributes

About MLUserRepository

The MonaLisa object MLUserRepository enables you to read user profile properties and rights from LDAP. For more information, refer to "UserRepository.config" in the *Content Composer Advanced Design and Setup Guide*.

To load the object, use the MonaLisa function GetObject.

Example

```
GetObject("UserRepository", "MLUserRepository")
```

Functions of the Class MLUserRepository

GetCurrentUser

Returns the MonaLisa user object for the current logged in user.

GetCurrentUser	
SYNTAX	GetCurrentUser()
RETURN	MonaLisa user object.

GetUser

Returns the MonaLisa user object for the specified user. An error is thrown if the user does not exists in the user repository.

GetUser		
SYNTAX	GetUser(userName)	
ARGUMENTS	userName	Name of the user.
RETURN	MonaLisa user object.	

Functions of the MonaLisa User Object

GetAttribute

Returns the value of the attribute.

The attributes UserId (user name) and UserRoleNames (comma separated roles) are always present, all other attributes depend on a profile and its mapping to LDAP.

GetAttribute		
SYNTAX	GetAttribute(attribute)	
ARGUMENTS	attribute	User attributes.
RETURN	Attribute value or an empty string if the attribute does not exist.	

HasPermission

Show All

Checks whether a user has the corresponding rights.

HasPermission

SYNTAX

HasPermission(int permission)

HasPermission(ComposerPermission permission)

ARGUMENTS

permission — User right. The parameter can be returned as an integer value or an enumerator.

EnumeratorComposerPermission

Enumerator	Integer Value	Definition
ModusPermission.None	0	No right.
ModusPermission.ExecuteModusStudio	1	Is allowed to start Content Composer Studio.
ModusPermission.ShowNavigator	2	Shows the navigator.
ModusPermission.MwsShowNavigator	3	Shows the navigator in Content Composer Client.
ModusPermission.CommonSettings	4	General user settings (Main menu -> Settings).
ModusPermission.ChangeSystem	5	Allows a user to change

		systems.
ModusPermission.MwsChangeSystem	6	Allows a user to change systems in the Content Composer Client.
ModusPermission.ChangeFolder	7	Allows changing to a lower folder but not in a parent folder.
ModusPermission.Search	8	Search in current system is allowed.
ModusPermission.ShowTypeFilter	9	Show object filter for the various object types in the navigator.
ModusPermission.TransferSystem	10	System can be transferred.
ModusPermission.SynchronizeSystem	11	System can be synchronized.
ModusPermission.Administrate	12	User can perform administratio n tasks.
ModusPermission.MwsDownloadDocument	13	Download of created

		documents to Content Composer Client.
ModusPermission.MwsProcessList 14	14	
ModusPermission.MwsForwaredProcessList	15	
ModusPermission.MwsInsertDocument	16	
ModusPermission.MwsInsertBlock	17	
ModusPermission.MwsSetBundlePrinter	18	
ModusPermission.MwsSetDocumentPrinter	19	
ModusPermission.MwsSetDocumentPrinter Wizard	20	
ModusPermission.MwsManualForwardProc ess	21	
ModusPermission.MwsCreateDocuments	22	
ModusPermission.MwsDataSelection	23	
ModusPermission.MwsPrintAndArchive	24	
ModusPermission.MwsAdministrate	25	
ModusPermission.MwsProcessListFromDefi nedUser	26	
ModusPermission.MwsChangeFolder	27	
ModusPermission.MwsViewProcess	28	
ModusPermission.MwsViewStack	29	

ModusPermission.MwsViewOpenEnvelope	30	
ModusPermission.MwsViewOpenJob	31	
ModusPermission.MwsClearCache	32	
ModusPermission.ModifyFontFace	2000	
ModusPermission.ModifyFontSize	2001	
ModusPermission.ModifyFontColor	2002	
ModusPermission.ModifyFontStyleBold	2003	
ModusPermission.ModifyFontStyleItalic	2004	
ModusPermission.ModifyFontStyleUnderline	2005	
ModusPermission.ModifyFontAlignment	2006	
ModusPermission.CanPrintInPreview	2011	
ModusPermission.OdinProcessUnlock	3000	User can unlock a locked process.
ModusPermission.OdinProcessSetBack	3001	(Resume) User can reset a process from **Locked** to **Rendition created** or **Imported**.
ModusPermission.OdinProcessCreateStack	3002	User can create a stack for print /

		delete from process jobs.
ModusPermission.OdinProcessDelete	3003	User can set processes to the status **Delete**.
ModusPermission.OdinProcessSuspend	3004	User can lock a process.
ModusPermission.OdinOpenJobCreateStack	3005	User can create a stack for print / delete from open jobs.
ModusPermission.OdinStackEdit	3500	User can edit stacks.
ModusPermission.OdinStackFree	3501	User can release stacks.
ModusPermission.OdinStackDelete	3502	User can delete stacks.
ModusPermission.OdinStackSetBack	3503	User can reset a stack to the status **Wait**.
ModusPermission.OdinStackSetBackToStreaming	3504	User can reset a stack for streaming.
ModusPermission.OdinStackLock	3505	User can lock a stack.

ModusPermission.OdinStackUnlock	3506	User can unlock a stack.
ModusPermission.OdinStackSetBackTo	3507	User can reset a stack to a specific stack type.
ModusPermission.OdinJobSetState	3508	User can reset jobs to a specific status.
ModusPermission.Test	3509	User can test documents and bundles in Content Composer Client integrated in Content Composer Studio.
ModusPermission.OdinJobSetPrinter	3510	User can change the printer of a job.
ModusPermission.OdinStackEnvelopeRemove	3511	User can remove an envelope from a stack.
ModusPermission.OdinJobRemove	3512	User can remove an job from an envelope or stack.

RETURN	TRUE = The user has the corresponding object rights. FALSE = The user does not have the corresponding object rights.

HasObjectPermission

Show All

Checks whether a user has the specified object rights (create, read, update, delete) for an object.

HasObjectPermission

SYNTAX

HasObjectPermission(int objectType, int permission)

HasObjectPermission(ComposerSuiteObjectType objectType, ComposerObjectActionPermission permission)

ARGUMENTS

objectType — The parameter can be returned as an integer value or an enumerator.

Enumerator	Integer	Object type
ComposerSuiteObjectType.TextComponent	4	Text block
ComposerSuiteObjectType.TextDocument	6	Document template
ComposerSuiteObjectType.TextDocumentCollection	8	Bundles
ComposerSuiteObjectType.Script	9	Script
ComposerSuiteObjectType.Condition	10	Conditions
ComposerSuiteObjectType.LookupTable	11	Lookup

		table
ComposerSuiteObjectType.Process	12	Process / Workflow
ComposerSuiteObjectType.Folder	13	Folders
ComposerSuiteObjectType.DataObject	22	Data object (Xdata)
ComposerSuiteObjectType.Selection	24	Selection
ComposerSuiteObjectType.DataObjectScript	27	Data object script
ComposerSuiteObjectType.DataObjectQuery	28	Data object query
ComposerSuiteObjectType.TransferJob	30	Transfer job
ComposerSuiteObjectType.Annex	34	Annex
ComposerSuiteObjectType.Envelope	36	Envelope
ComposerSuiteObjectType.PostageDefinition	38	Postage definition
ComposerSuiteObjectType.XsdSelection	39	XSD selection
ComposerSuiteObjectType.Printer	40	Odin online printer
ComposerSuiteObjectType.AliasTable	42	Alias tables in Content Composer
ComposerSuiteObjectType.Form	49	Odin form
ComposerSuiteObjectType.ValueHelp	52	Value help

		object for variables
ComposerSuiteObjectType.DataProviderDefinition	53	Database query definition
ComposerSuiteObjectType.MwsProcess	59	MWS process
ComposerSuiteObjectType.StandardProcess	61	Standard process
ComposerSuiteObjectType.EMailPrinter	62	Odin email printer
ComposerSuiteObjectType.OutsourcingPrinter	63	Odin outsourcing printer
ComposerSuiteObjectType.CustomUI	64	CustomUI definition (ribbon)
ComposerSuiteObjectType.Profile	69	Profile definition (profile attribute collection)
ComposerSuiteObjectType.Role	71	Role
ComposerSuiteObjectType.SimpleTextContainer	72	Container for simple, unformatted text blocks to be inserted in Word documents

ComposerSuiteObjectType.RoleMapping	73	Stores the mapping of external roles to Content Composer roles

permission

The parameter can be returned as an integer value or an enumerator.

Enumerator	Integer
ComposerObjectActionPermission.None	0
ComposerObjectActionPermission.Create	1
ComposerObjectActionPermission.Read	2
ComposerObjectActionPermission.Update	4
ComposerObjectActionPermission.Delete	8

RETURN

TRUE = The user has the corresponding object rights.

FALSE = The user does not have the corresponding object rights.

IsInRole

Checks whether the user is a member of the corresponding role.

IsInRole

SYNTAX IsInRole(roleName)

ARGUMENTS

roleName	Name of the role.

RETURN	TRUE = The user is a member of the role. FALSE = The user is not a member of the role.

Sample Script MLUserRepository

```
; Initialize MonaLisa UserRepository object
GetObject("userRepository", "MLUserRepository")

try
  ; Get current logged in user
  currentUser = userRepository.GetCurrentUser()

  ; Get user from UserRepository
  user = userRepository.GetUser("JKlinger")

  ; UserName ("UserId" always exists per default)
  userName = user.GetAttribute("UserId")

  ; Roles of the user ("UserRoleNames" always exists per default)
  roles = user.GetAttribute("UserRoleNames")

  ; Check if user is in a specific role
  isAdmin = user.IsInRole("AdminRole")

  ; Check if user has permission to modify a system
  canChangeSystem = user.HasPermission(ModusPermission.ChangeSystem)
  ; same with integer instead of enum
  canChangeSystem_int = user.HasPermission(5)
```

```
  ; Check if user has object-permission to create or delete a script
  canCreateScript = user.HasObjectPermission(ModusSuiteObjectType.Script,
ComposerObjectActionPermission.Create)

  canDeleteScript = user.HasObjectPermission(ModusSuiteObjectType.Script,
ComposerObjectActionPermission.Delete)

  ; same with integer instead of enum
  canCreateScript_int = user.HasObjectPermission(9, 1)
  canDeleteScript_int = user.HasObjectPermission(9, 8)
  canCreateDeleteScript_int = user.HasObjectPermission(9, 1+8)  ; combined create+delete -> 1+8

OnError

  Protocol("Error reading user from UserRepository.", 0)

end-try
```

Configuration Files

Configuration files such as Composer.Studio.exe.config require the following section:

```
<configuration>
  <configSections>
  ...
    <section name="userRepository"
type="ModusSuite.UserRepository.DataAccess.UserRepositoryConfiguration,
ModusSuite.UserRepository.DataAccess"/>
    <section name="userRepository_Ldap"
type="ModusSuite.UserRepository.DataAccess.Ldap.LdapConfiguration,
ModusSuite.UserRepository.DataAccess"/>
  ...
  </configSections>
  <userRepository configSource = "UserRepository.config"/>
  <userRepository_Ldap configSource = "UserRepository_ldap.config"/>
```

</configuration>

Archive Functions

About MLPerceptiveArchive

The MonaLisa object MLPerceptiveArchive enables you to add documents to the Content Composer Archive (Perceptive Content). To load the object, use the MonaLisa function GetObject.

Example

GetObject("INIS","MLPerceptiveArchive")

Functions of the Class MLPerceptiveArchive

AddDocumentPage

Adds a file as a new page to an existing Perceptive Content document.

AddDocumentPage		
SYNTAX	AddDocumentPage(string documentId, string filePath)	
ARGUMENTS	documentId	ID of the Perceptive Content document to which you want to add the new page.
	filePath	File to upload and store as a page in the Perceptive Content document.
RETURN	The ID of the newly created page.	
EXAMPLE	addedPageId = INIS.AddDocumentPage(docId,"%ComposerDir%Admin\MS-SQL\MSSql_Rep_DB_Schema.sql")	

AddToWorkflow

This function adds a document or folder to a workflow queue, which creates a workflow item.

AddToWorkflow	
SYNTAX	`AddToWorkflow (string objectId, string workflowQueueId, string itemType, string priority)`

ARGUMENTS	objectId	The unique ID of the document you want to update.
	workflowQueueID	The unique ID of the workflow queue.
	itemType	Indicates whether the item is a document or a folder. Possible values are “DOCUMENT” or “FOLDER”.
	priority	The priority level for the workflow item. Possible values are “LOW”, “MEDIUM”, or “HIGH”.
	Return value	The unique ID of the newly created workflow item.

Connect

Opens a connection and authenticates to the Perceptive Content Integration Server.

Syntax

Connect(string serverURI,string userName, string password)

Parameter	Description
serverURI	Root URI of the Hyland Integration Server.
userName	User name to log in to the Hyland Integration Server.
password	Encrypted password to log in to the Hyland Integration Server. **Note** To encode the password, use the tool **Encoder.exe** in the installation directory.

Return value

-

CreateDocument

Creates an Perceptive Content document.

CreateDocument	
SYNTAX	CreateDocument(filePath, documentName, documentTypeName, drawerName)

ARGUMENTS	filePath	File to upload and store as a Perceptive Content document.
	documentName	Name of the Perceptive Content document to create. The maximum string length is 128 characters.
	documentTypeName	Type of document to create. The type must exist in Perceptive Content.
	drawerName	Drawer in which to store the document in Perceptive Content. The drawer must exist in Perceptive Content.
RETURN	ID of the created document.	

CreateDocumentWithFields

Creates a Perceptive Content document. It is possible to specify values for the standard Perceptive Content fields 1-5. However, either the document name or one of the field values must be a non-empty string to successfully create the Perceptive Content document.

CreateDocumentWithFields

SYNTAX CreateDocumentWithFields(filePath, documentName, documentTypeName, drawerName, field1, field2, field3, field4, field5)

ARGUMENTS	filePath	The path where the file is uploaded and stored as a Perceptive Content document.
	documentName	Name of the Perceptive Content document to create. The maximum string length is 128 characters.
	documentTypeName	Type of the document to create. The type must exist in Perceptive Content.
	drawerName	Name of the drawer where the document is stored in Perceptive Content. The drawer must exist in Perceptive Content.
	field1 - field5	A value for the Perceptive Content field attribute.
RETURN	ID of the created document.	

Disconnect

Closes an existing connection.

Disconnect	
SYNTAX	Disconnect()

GetDocumentInformation

Retrieves document information for a Perceptive Content document that matches the given ID..

Syntax

GetDocumentInformation(documentId)

Arguments

documentId. ID of Perceptive Content document.

Return

XML String that contains information about Perceptive Content document.

GetDocumentViewResult

Returns the result of the specified document view additionally filtered by the optional specified VSL query.

GetDocumentViewResult		
SYNTAX	GetDocumentViewResult(viewId, vslQuery)	
ARGUMENTS	viewId	ID of the view to execute. The view must be a DOCUMENT view.
	vslQuery	String that contains a filter in VSL query syntax. Use an empty string to return the unfiltered result of the view. For more information on VSL syntax, refer to "VSL" in Perceptive Integration Server Help.
RETURN	Proprietary XML structure that contains the definition of the view itself and the result of the view.	

GetDocumentViews

Returns an XML structure that contains all existing views in the DOCUMENT category.

GetDocumentViews	
SYNTAX	GetDocumentViews()
RETURN	XML structure that contains all existing views in the DOCUMENT category.

GetFolderContentViewResult

Returns the result of the specified folder content view additionally filtered by the optional specified VSL query.

GetFolderContentViewResult		
SYNTAX	GetFolderContentViewResult(viewId, vslQuery)	
ARGUMENTS	viewId	ID of the view to execute. The view must be a FOLDER_CONTENT view.
	vslQuery	String that contains a filter in VSL query syntax. Use an empty string to return the unfiltered result of the view. For more information on VSL syntax, refer to "VSL" in Perceptive Integration Server Help.
RETURN	Proprietary XML structure that contains the definition of the view itself and the result of the view.	

GetFolderViewResult

Returns the result of the specified folder view additionally filtered by the optional specified VSL query.

GetFolderViewResult	
SYNTAX	GetFolderViewResult(viewId, vslQuery)

ARGUMENTS	viewId	ID of the view to execute. The view must be a FOLDER view.
	vslQuery	String that contains a filter in VSL query syntax. Use an empty string to return the unfiltered result of the view. For more information on VSL syntax, refer to "VSL" in Perceptive Integration Server Help.
RETURN	Proprietary XML structure that contains the definition of the view itself and the result of the view.	

GetFolderViews

Returns an XML structure that contains all existing views in the FOLDER category.

GetFolderViews	
SYNTAX	GetFolderViews()
RETURN	XML structure that contains all existing views in the FOLDER category.

GetPageInformation

Retrieves page information from the Perceptive Content Server for the document that matches the given ID. The Perceptive Content Server response includes a section for each page. It provides the page ID, filename, extension and page number. In Perceptive Content, a document may contain multiple files, where each file is considered a page.

Syntax

GetPageInformation(documentId)

Arguments

documentId. ID of Perceptive Content document.

Return

XML string that contains information about Perceptive Content pages stored for a document.

GetPageFile

This API call requests the file contents from the Perceptive Content Server for the page that matches the given ID. The request includes the ID for the document and the ID for the page within the document that you want to retrieve.

Syntax

GetPageFile(documentId, pageId, destinationFilePath)

Arguments

- documentId: ID of the Perceptive Content document.
- pageId: ID of the Perceptive Content document page.
- destinationFilePath: Filepath used to store the file retrieved from the Perceptive Content archive.

GetWorkflowQueues

This API call requests the list of workflow queues stored in the Perceptive Content Server that the user has the privileges to access.

Syntax

GetWorkflowQueues()

Return

Returns a XML string that contains the workflow queues that the user can access. It is the same XML that is returned by the Integration Server call GET: /workflow.

Example of return string

```
<workflowQueues>
  <workflowQueue>
    <name>Queue 1</name>
    <id>301YW1Z_0003DBRLH000001</id>
    <processName>Process</processName>
    <processId>301YW1Z_0003DPRLH000001</processId>
  </workflowQueue>
</workflowQueues>
```

Example script

The following script shows how to call GetWorkflowQueues() and use the Monalisa XML functions to extract the attributes of a specific workflow queue.

```
GetObject("INIS","MLPerceptiveArchive")
```

```
INIS.Timeout = 10000
session = ""
try
   INIS.Connect("http://10.184.245.139:8080/integrationserver","Administrator","secret_password")
   session = INIS.SessionId
   ;get list of workflowQueueus
   resXml = INIS.GetWorkflowQueues()
   ;get id for queue with specific name
   GetObject("Xml","MLXmlDocument")
   try
      Xml.LoadXml(resXml)
      searchName = "queue 1"
      xpath = FormatStr("//workflowQueues/workflowQueue[name = '%s']/id", searchName)
      node = Xml.SelectSingleNode(xpath)
      workflowId = node.InnerText
   OnError
      workflowId = "NOT FOUND"
   Finally
      Protocol("WorkFlowQueue: '{0}', Id: '{1}'", 8, searchName, workflowId)
   end-try
OnError
   errorContent = INIS.LastErrorResponseContent
   Protocol("Error: {0}",0,Error.Message)
   if(errorContent <> "")
      Protocol("Error: {0}",0, errorContent)
   end-if
Finally
   try
      if(session <> "")
         INIS.Disconnect()
```

```
    end-if
    FreeObject("INIS")
  OnError
    Protocol("Error while disconnecting: {0}",0,Error.Message)
  end-try
end-try
```

MoveDocumentToFolder

Moves an Perceptive Content document to an Perceptive Content folder.

MoveDocumentToFolder		
SYNTAX	MoveDocumentToFolder(documentId, targetFolderId)	
ARGUMENTS	documentId	ID of the Perceptive Content document you want to move.
	targetFolderId	ID of the Perceptive Content folder to move the Perceptive Content document to.

SearchForDocument

Returns the result of the view **SysDocumentAll**, additionally filtered by the optional specified VSL query.

SearchForDocument		
SYNTAX	SearchForDocument(vslQuery)	
ARGUMENTS	vslQuery	String that contains a filter in VSL query syntax. Use an empty string to return the unfiltered result of the view. For more information on VSL syntax, refer to "VSL" in Perceptive Integration Server Help.
RETURN	Proprietary XML structure that contains the definition of the view itself and the result of the view.	

UpdateDocumentDateAttribute

This function updates a custom document attribute of type “DATE”.

UpdateDocumentDateAttribute

SYNTAX

```
UpdateDocumentDateAttribute(string documentId, string attributeId, string dateFormat, string dateStringValue, bool isUTCTime)
```

ARGUMENTS

documentId	The unique ID of the document you want to update.
attributeID	The unique ID of the attribute to update.
dateFormat	The format rules that define the date string passed with argument "'dateStringValue'. For more information, refer to Format specifiers.
dateStringValue	The date value.
isUTCTime	This parameter returns the value true if the specified date time is UTC time, otherwise it returns the value false.
Return value	-

UpdateDocumentFlagAttribute

This function updates a custom document attribute of type “FLAG” (Boolean).

UpdateDocumentFlagAttribute

SYNTAX

```
UpdateDocumentFlagAttribute(string documentId, string attributeId, object attributeValue)
```

ARGUMENTS

documentId	The unique ID of the document you want to update.
attributeID	The unique ID of the attribute to update
AttributeValue	The flag value to write. The possible values are Boolean values `TRUE` or `FALSE` or string values containing “true” or “false”.
Return value	-

UpdateDocumentKey

Selectively updates any one of the five Perceptive Content document key fields (Field1-Field5).

UpdateDocumentKey		
SYNTAX	UpdateDocumentKey(documentId, keyFieldIndex, value)	
ARGUMENTS	documentId	ID of the Perceptive Content document you want to update.
	keyFieldIndex	Number of the key field to update. **Example** • 1 for Field1 • 2 for Field2
	value	Value to write into the key field. The maximum string length is 40 characters.

UpdateDocumentNumberAttribute

This function updates a custom document attribute of type “NUMBER”.

UpdateDocumentNumberAttribute		
SYNTAX	`UpdateDocumentNumberAttribute(string documentId, string attributeId, object attributeValue)`	
ARGUMENTS	documentId	The unique ID of the document you want to update.
	attributeID	The unique ID of the attribute to update.
	attributeValue	The number value.
	Return value	-

UpdateDocumentStringAttribute

This function updates a custom document attribute of type “STRING”.

UpdateDocumentStringAttribute

SYNTAX `UpdateDocumentStringAttribute(string documentId, string attributeId, object attributeValue)`

ARGUMENTS

documentId	The unique ID of the document you want to update.
attributeID	The unique ID of the attribute to update.
attributeValue	The string value.
Return value	-

Properties of the Class MLPerceptiveArchive

Property	Description
BaseURI	Root URI of the Restful Web Service to access.
Connected	TRUE = A connection to the Perceptive Integration Server exists. FALSE = No connection to the Perceptive Integration Server exists.
LastErrorResponseContent	Contains the error response content of the last executed request.
ServerVersion	Contains the version of the Perceptive Integration Server API to which a connection is established or an empty string if no connection is established.
SessionID	Contains the current `SessionID` if connected.
Timeout	Timeout in milliseconds used for requests and responses.

Sample Script MLPerceptiveArchive

```
GetObject("INIS","MLPerceptiveArchive")
INIS.Timeout = 10000
session = ""
try
```

```
  Protocol("Connecting to IS ...",5)
  INIS.Connect
("http://10.184.107.75:8080/integrationserver","Administrator","ysKNoeLwWW9uyrlDjU2m5A==")
  session = INIS.SessionId
  Protocol("Connected: to {1} (API v{2}) - Session: {0}",5, session, INIS.BaseUri, INIS.ServerVersion)
  Protocol("Getting available document views ...",5)
  result = INIS.GetDocumentViews()
  ;Protocol("The following document views exist on {0}",5, INIS.BaseUri)
  ;Protocol(result,5) ;For debugging purposes - output the complete result xml
  result = INIS.GetFolderViews()
  ;Protocol("The following folder views exist on {0}",5, INIS.BaseUri)
  ;Protocol(result,5) ;For debugging purposes - output the complete result xml
  vslQuery = "[name] startswith 'test'" ; For information about the VSL Syntax refer to
https://docs.perceptivesoftware.com/robohelp/robo/server/6.7/PDM/en_
US/help.htm#Subsystems/integrationserver/VSL/VSLStatementSyntax.htm%3FTocPath%3DDeveloper%2
0Topics|Using%20Integration%20Server|VSL|_____2
  Protocol("Searching for documents (Query: '{0}')...",5, vslQuery)
  result = INIS.SearchForDocument(vslQuery)
  ; Some useful XPath statements when working with the result XML of SearchForDocument
  ;//view/columns/column[name[.='Document ID']]/columnId
  ;//view/columns/column[name[.='Name']]/columnId
  ;//view/columns/column[name[.='Field1']]/columnId
  ;//viewResults/resultRows/resultRow/fields/field[columnId[.='???']]/value

  ;Protocol(result,5) ;For debugging purposes - output the complete result xml
  GetObject("ResultXml","MLXmlDocument")
  ResultXml.LoadXml(result)
  nameColumnIdNode = ResultXml.SelectSingleNode("//view/columns/column[name[.='Name']]/columnId")
  nameColumnId = nameColumnIdNode.InnerText
  docIdColumnIdNode = ResultXml.SelectSingleNode("//view/columns/column[name[.='Document
ID']]/columnId")
```

```
  docIdColumnId = docIdColumnIdNode.InnerText
  resultRowList = ResultXml.SelectNodes("//viewResults/resultRows/resultRow/fields")
  docResultCounter = resultRowList.Count
  Protocol("The following {1} Document(s) were found by query '{0}':",5, vslQuery, docResultCounter)
  resultEnumerator = resultRowList.GetEnumerator()
  readNext = resultEnumerator.MoveNext()
  while (readNext)
    currentNode = resultEnumerator.Current
    documentNameValueNode = currentNode.SelectSingleNode("field[columnId[.='" + nameColumnId +
"']]/value")
    documentIdValueNode = currentNode.SelectSingleNode("field[columnId[.='" + docIdColumnId +
"']]/value")
    documentName = documentNameValueNode.InnerText
    documentId = documentIdValueNode.InnerText
    Protocol("Search result: {0} (ID: {1})",5,documentName, documentId)
    readNext = resultEnumerator.MoveNext()
  end-while

  result = INIS.GetFolderViewResult("SysProjectsAll","")
  ;Protocol(result,5) ;For debugging purposes - output the complete result xml

  Protocol("--------- Search test finished moving on to modification tests ---------",5)
  GetObject("User","MLUserRepository")
  currentUser = User.GetCurrentUser()
  userName = currentUser.GetAttribute("UserId")
  roleNames = currentUser.GetAttribute("UserRoleNames")
  FreeObject("User")
  docName = "Test Document (from Content Composer Monalisa Script)"
  field1 = "Content Composer-User: " + userName
  field2 = "Content Composer-Roles: " + roleNames
```

```
  field5 = "Some other useful information"
  field4 = "Fields can not be longer than 40 chars."
  result = INIS.SearchForDocument("[name] = '" + docName + "' and [field1] = '" + field1 + "' and [field2] = '" + field2 + "' and [field3] = $NULL and [field4] = '" + field4 + "' and [field5] = '" + field5 + "'")
  ResultXml.LoadXml(result)
  resultRowList = ResultXml.SelectNodes("//viewResults/resultRows/resultRow/fields")
  resultCount = resultRowList.Count
  if(resultCount > 0)
    node = ResultXml.SelectSingleNode("//viewResults/resultRows/resultRow/fields/field[columnId[.='" + docIdColumnId + "']]/value")
    docId = node.InnerText
    Protocol("The document '{0}' already exists {2} time(s) and the 1st of {2} has ID {1} ! No document will be created !",1, docName, docId, resultCount)
  else
    docId = INIS.CreateDocument("%ComposerDir%Admin\MS-SQL\MSSql_
CreateUser.sql",docName,"Default","Folders")
    Protocol("Document '{1}' ({0}) created",5, docId, docName)
  end-if

  INIS.UpdateDocumentKey(docId,2,field2)
  INIS.UpdateDocumentKey(docId,1,field1)
  INIS.UpdateDocumentKey(docId,5,field5)
  Protocol("Document '{1}' ({0}) Field 1,2 and 5 updated",5, docId, docName)
  addedPageId = INIS.AddDocumentPage(docId,"%ComposerDir%Admin\MS-SQL\MSSql_Rep_DB_
Schema.sql")
  Protocol("Created new page {2} in document '{1}' ({0})",5, docId, docName,addedPageId)
  INIS.UpdateDocumentKey(docId,4,field4)
  Protocol("Document '{1}' ({0}) Field 4 updated",5, docId, docName)

  folderQuery = "[name] = 'TestFolder'"
  result = INIS.GetFolderViewResult("SysProjectsAll",folderQuery)
```

```
  ResultXml.LoadXml(result)
  folderIdColumnIdNode = ResultXml.SelectSingleNode("//view/columns/column[name[.='Folder
ID']]/columnId")
  folderIdColumnId = folderIdColumnIdNode.InnerText
  resultRowList = ResultXml.SelectNodes("//viewResults/resultRows/resultRow/fields")
  resultCount = resultRowList.Count
  if(resultCount > 0)
    node = ResultXml.SelectSingleNode("//viewResults/resultRows/resultRow/fields/field[columnId[.='" +
folderIdColumnId + "']]/value")
    targetFolderId = node.InnerText
    Protocol("Found {0} folder(s) as possible move target (Query: {1}). Using the 1st of {0} ... using Folder ID
{2}",5, resultCount,folderQuery, targetFolderId)
    INIS.MoveDocumentToFolder(docId,targetFolderId)
    Protocol("Document '{1}' ({0}) moved into folder with ID {2}",5, docId, docName, targetFolderId)
  else
    Protocol("No folder was found as a possible move target (Query: {0}). No document was
moved!",1,folderQuery)
  end-if
OnError
  errorContent = INIS.LastErrorResponseContent
  Protocol("Error: {0}",0,Error.Message)
  if(errorContent <> "")
    Protocol("Error: {0}",0, errorContent)
  end-if
Finally
  try
    if(session <> "")
      Protocol("Disconnecting from IS ...",5)
      INIS.Disconnect()
      Protocol("Disconnected: {0}",5,session)
    end-if
```

```
    FreeObject("ResultXml")
    FreeObject("INIS")
  OnError
    Protocol("Error while disconnecting: {0}",0,Error.Message)
  end-try
end-try
```

Sample Script - Retrieve Files from Archive

```
GetObject("INIS","MLPerceptiveArchive")
INIS.Timeout = 10000
session = ""
try
  GetObject("xml","MLXmlDocument")
  GetObject("xml2","MLXmlDocument")
  Protocol("Connecting to IS ...",5)
  INIS.Connect("http://10.111.111.111:8080/integrationserver","Administrator","SECRET")
  session = INIS.SessionId
  Protocol("Connected: to {1} (API v{2}) - Session: {0}",5, session, INIS.BaseUri, INIS.ServerVersion)
  documentId = "321YZCH_00021M8QQ00001Q"
  resultDocumentXml = INIS.GetDocumentInformation(documentId)
  Protocol(resultDocumentXml,5)

  xml.LoadXml(resultDocumentXml)
  resultPagesList = xml.SelectNodes("//document/pages/page")

  pagesResultCounter = resultPagesList.Count
  Protocol("The document has {0} pages(s) ",5, pagesResultCounter)
  resultEnumerator = resultPagesList.GetEnumerator()
  readNext = resultEnumerator.MoveNext()
  while (readNext)
```

```
      currentNode = resultEnumerator.Current
      xml2.LoadXml(currentNode.OuterXml)
      currentNode = xml2.DocumentElement

      pageId = xml.GetXPathValue(currentNode, "//page/id")
      pageName = xml.GetXPathValue(currentNode, "//page/name")
      pageExt = xml.GetXPathValue(currentNode, "//page/extension")
      pageNo = xml.GetXPathValue(currentNode, "//page/pageNumber")

      fileName = System.IO.Path.Combine("c:\temp\", FormatStr("%s.%s", pageId, pageExt))

      ;resultPageXml = INIS.GetPageInformation(documentId)
      ;Protocol(resultPageXml,5)
      INIS.GetPageFile(documentId, pageId, fileName)

      readNext = resultEnumerator.MoveNext()
   end-while
OnError
   errorContent = INIS.LastErrorResponseContent
   Protocol("Error: {0}",0,Error.Message)
   if(errorContent <> "")
      Protocol("Error: {0}",0, errorContent)
   end-if
Finally
   try
      if(session <> "")
         Protocol("Disconnecting from IS ...",5)
         INIS.Disconnect()
         Protocol("Disconnected: {0}",5,session)
```

```
    end-if
    FreeObject("xml")
    FreeObject("xml2")
    FreeObject("INIS")
  OnError
    Protocol("Error while disconnecting: {0}",0,Error.Message)
  end-try
end-try
```

Array Functions

About Array Functions

The MonaLisa object MLArrays enables you to work with object arrays.

To load the object, use the MonaLisa function GetObject.

Example

```
GetObject("array","MLArrays")
```

Properties of the Class MLArrays

Property	Description
Count	Number of elements in the array.

Functions of the Class MLArrays

AddToArray

Appends an element - as a specific type if specified - at the end of the array.

AddToArray		
SYNTAX	AddToArray(obj, addAsType)	
ARGUMENTS	obj	Object to add.
	addAsType	Element type. This parameter is optional.

Clear

Clears all elements of the array.

Clear	
SYNTAX	Clear()

GetArray

Returns a typed .NET array.

GetArray		
SYNTAX	GetArray(type)	
ARGUMENTS	type	Type of the array.
RETURN	Typed .NET array.	

GetAtIndex

Retrieves the element in the array at the specified position.

GetAtIndex		
SYNTAX	GetAtIndex(index)	
ARGUMENTS	index	Zero-based position from which to retrieve the element.
RETURN	Object contained in the array.	

InitFromArrayObject

Initializes the class as array with all elements from the provided arrayObject.

InitFromArrayObject		
SYNTAX	InitFromArrayObject(arrayObject)	
ARGUMENTS	arrayObject	Object that implements the IList interface.

InsertIntoArray

Inserts an element - as a specific type if specified - into the array at the specified position.

InsertIntoArray		
SYNTAX	InsertIntoArray(index, obj, addAsType)	
ARGUMENTS	index	Zero-based index at which the value inserts.
	obj	Object to insert.
	addAsType	Element type. This parameter is optional.

Object Functions

About Object Functions

The MonaLisa object MLObjects enables you to work with types and create dynamic object instances.

To load the object, use the MonaLisa function GetObject.

Example

GetObject("object","MLObjects")

Functions of the Class MLObjects

CreateNewInstanceOfType

Creates a new object instance of the specified type.

CreateNewInstanceOfType

SYNTAX CreateNewInstanceOfType(type, args)

ARGUMENTS

type	Type for which to create an instance.
args	Arguments for the instance. This parameter is optional.

RETURN New object instance of the type.

GetEnumValue

Returns the enumeration value of a specific type by a specific untyped value.

GetEnumValue

SYNTAX GetEnumValue(type, value)

ARGUMENTS

type	Enumeration type.
value	Untyped value of the enumeration value.

RETURN Typed enumeration value.

GetModule

Returns the module in which the specified type is defined.

GetModule

SYNTAX GetModule(type)

ARGUMENTS

type	Type for which to get the module.

RETURN Module in which the specified type is defined.

GetTypeInModule

Resolves the typestring to a real type within a specific module.

GetTypeInModule		
SYNTAX	GetTypeInModule(type, module)	
ARGUMENTS	type	Typestring used to retrieve the type.
	module	Module in which the type is specified.
RETURN	Real type.	

GetTypeOfObject

Returns the type of the specified object instance.

GetTypeOfObject		
SYNTAX	GetTypeOfObject(obj)	
ARGUMENTS	obj	Object instance.
RETURN	Type of the specified object instance.	

GetTypeOfObjectAsString

Returns the typestring of the specified object instance.

GetTypeOfObjectAsString		
SYNTAX	GetTypeOfObjectAsString(obj)	
ARGUMENTS	obj	Object instance.
RETURN	Typestring of the specified object instance.	

MLIdAutomation

About MLIdAutomation

The MonaLisa object MLIdAutomation enables you to call IDAutomation's **Native Windows Barcode Font Encoder DLL** API.

This product is not included in the Content Composer setup. To purchase the product, contact **IDAutomation.com**.

Prerequisites

- A valid license for the product **Native Windows Barcode Font Encoder DLL**
- The DLL IDAutomationNativeFontEncoder64.dll. This file must be present in the **%Composerdir%** directory.
- The IDAutomation bar code font files

Note This MonaLisa object is only available for 64bit installations of Content Composer.

To load the object, use the MonaLisa function GetObject.

Example

GetObject("barcode", "MLIdAutomation")

Functions of the Class MLIdAutomation

Code128

For more information, see “IDAutomation_Code128” in the *Native Windows Barcode Font Encoder DLL* documentation.

Code128		
SYNTAX	Code128(stringtoEncode, useTilde)	
ARGUMENTS	stringtoEncode	The bar code value to encode as string.
	useTilde	TRUE = Offers various options. For more information, see “IDAutomation_Code128” in the *Native Windows Barcode Font Encoder DLL* documentation. FALSE = Format the data as is.
RETURN	The encoded bar code value as string.	
EXAMPLE	code128 = barcode.Code128("HelloWorld", false)	

Code128a

For more information, see “IDAutomation_Code128a” in the *Native Windows Barcode Font Encoder DLL* documentation.

Code128a	
SYNTAX	Code128a(stringtoEncode)
ARGUMENTS	stringtoEncode — The bar code value to encode as string.
RETURN	The encoded bar code value as string.
EXAMPLE	code128a = barcode.Code128a("HelloWorld")

Code128b

For more information, see “IDAutomation_Code128b” in the *Native Windows Barcode Font Encoder DLL* documentation.

Code128b	
SYNTAX	Code128b(stringtoEncode)
ARGUMENTS	stringtoEncode — The bar code value to encode as string.
RETURN	The encoded bar code value as string.
EXAMPLE	code128b = barcode.Code128b("HelloWorld")

Code128c

For more information, see “IDAutomation_Code128c” in the *Native Windows Barcode Font Encoder DLL* documentation.

Code128c

SYNTAX	Code128c(stringtoEncode)
ARGUMENTS	stringtoEncode — The bar code value to encode as string.
RETURN	The encoded bar code value as string.
EXAMPLE	code128c = barcode.Code128c("0815")

Interleaved2of5

For more information, see “IDAutomation_Interleaved2of5” in the *Native Windows Barcode Font Encoder DLL* documentation.

Interleaved2of5

SYNTAX	Interleaved2of5(stringtoEncode)
ARGUMENTS	stringtoEncode — The bar code value to encode as string.
RETURN	The encoded bar code value as string.
EXAMPLE	interleaved2of5 = barcode.Interleaved2of5("123456789")

Interleaved2of5Mod10

For more information, see IDAutomation_Interleaved2of5Mod10” in the *Native Windows Barcode Font Encoder DLL* documentation.

Interleaved2of5

SYNTAX	Interleaved2of5Mod10(stringtoEncode)
ARGUMENTS	stringtoEncode — The bar code value to encode as string.
RETURN	The encoded bar code value as string.
EXAMPLE	interleaved2of5Mod10 = barcode.Interleaved2of5Mod10("123456789")

Sample Script MLIdAutomation

```
GetObject("barcode", "MLIdAutomation")

stringtoEncode = "HelloWorld"

useTilde = false

code128 = barcode.Code128(stringtoEncode, useTilde)

code128a = barcode.Code128a(stringtoEncode)

code128b = barcode.Code128b(stringtoEncode)

;Variant C is for encoding of numbers

code128c = barcode.Code128c("0815")

;Interleaved2of5 is for encoding of numbers

Interleaved2of5 = barcode.Interleaved2of5("123456789")

Interleaved2of5Mod10 = barcode.Interleaved2of5Mod10("123456789")
```

OnBase Document API

About MLOnBaseDocumentApi

The MonaLisa object **MLOnBaseDocumentApi** enables you to interact with **OnBase** by Hyland using the OnBase Document REST API.

The object provides the functionality to add a document file with keywords to an OnBase archive.

To load the object, use the MonaLisa function GetObject.

```
GetObject("OnBaseDocApi","MLOnBaseDocumentApi")
```

Functions of the Class MLOnBaseDocumentApi

AddFile

Adds an input file to the combined PDF archive file.

Use this function to combine multiple input files into one PDF archive file.

AddFile		
SYNTAX	AddFile(inputFilePath)	
ARGUMENTS	inputFilePath	Path of the input file.

ArchiveFile

Archives a single file.

ArchiveFile

SYNTAX ArchiveFile(inputFilePath, documentTypeName)

ARGUMENTS

inputFilePath	Path of the input file.
documentTypeName	Name of the OnBase document type.

RETURN OnBase document handle

ArchivePdfFile

Converts a single file to PDF file format and archives it.

ArchivePdfFile

SYNTAX ArchivePdfFile(inputFilePath, documentTypeName)

ARGUMENTS

inputFilePath	Path of the input file.
documentTypeName	Name of the OnBase document type.

RETURN OnBase document handle.

ClearKeywords

Clears all OnBase keywords in this MonaLisa object.

ClearKeywords

SYNTAX ClearKeywords()

FinishArchivePdfFile

Combines all added input files into one PDF file and archives this PDF file in OnBase.

Syntax

FinishArchivePdfFile	
SYNTAX	FinishArchivePdfFile()
RETURN	OnBase document handle.

LoadKeywordsFromFile

Loads OnBase keywords from an XML file.

LoadKeywordsFromFile		
SYNTAX	LoadKeywordsFromFile(keywordFileName)	
ARGUMENTS	keywordFileName	File path of the XML file that contains the OnBase keywords.
RETURN	Number of keywords read from the file.	

SetDocumentComment

Sets the comment of an OnBase document.

The default value is an empty string.

SetDocumentComment		
SYNTAX	SetDocumentComment(comment)	
ARGUMENTS	comment	The comment text.

SetDocumentDate

Sets the OnBase document date.

The default value is the current date.

SetDocumentDate

SYNTAX SetDocumentDate(datestring, dateformat)

ARGUMENTS

datestring	A date string.
dateformat	A date format string that describes the date format of the parameter *dateString.*

SetKeyword

Sets the OnBase string keyword.

SetKeyword

SYNTAX SetKeyword(keywordTypeName, value)

ARGUMENTS

keywordTypeName	The OnBase keyword type name.
value	The keyword value.

SetKeywordDate

Sets the OnBase keyword of type *date*.

SetKeywordDate

SYNTAX SetKeywordDate(keywordTypeName, value, dateFormat)

ARGUMENTS

keywordTypeName	The OnBase keyword type name.
value	The keyword value.
dateFormat	A date format string that describes the date format of the parameter *value.*

SetKeywordDateTime

Sets the OnBase keyword of type *date and time*.

SetKeywordDateTime

SYNTAX SetKeywordDateTime(keywordTypeName, value, dateTimeFormat)

ARGUMENTS

keywordTypeName	The OnBase keyword type name.
value	The keyword value.
dateTimeFormat	A date time format string that describes the date time format of the parameter *value.*

SetKeywordNumber

Sets the OnBase keyword of data types *numeric*, *currency* and *floating point*.

SetKeywordNumber

SYNTAX SetKeywordNumber(keywordTypeName, value, cultureName)

ARGUMENTS

keywordTypeName	The OnBase keyword type name.
value	The keyword value.
cultureName	The culture format of the parameter *value.*

SetLicenseFilePath

Sets the full file path of the Compart license file.

SetLicenseFilePath

SYNTAX SetLicenseFilePath(licenseFilePath)

ARGUMENTS	licenseFilePath	The full file path of the Compart license file. **Note**: You can also use Windows environment variables in the path.

SetOnBaseDocumentApiUrl

Sets the URL of the *OnBase Document API*.

SetOnBaseDocumentApiUrl

SYNTAX SetOnBaseDocumentApiUrl(licenseFilePath)

ARGUMENTS	onBaseApiUrl	URL of the *OnBase Document API*.

SetOnBaseDocumentApiUrlFromConfiguration

Reads the URL of the *OnBase Document API* from the app setting *onbaseRestApiUrl*.

SetOnBaseDocumentApiUrlFromConfiguration

SYNTAX SetOnBaseDocumentApiUrlFromConfiguration()

RETURN URL of the *OnBase Document API*.

SetProfilePath

Sets the path to the Compart profile directory.

SetProfilePath

SYNTAX SetProfilePath(profilePath)

ARGUMENTS	profilePath	The full file path of the Compart profile directory. **Note**: You can also use Windows environment variables in the path.

SetRequestTimeout

Sets the number of seconds to wait before the HTTP request times out.

The default value is 120 seconds.

SetRequestTimeout

SYNTAX	SetRequestTimeout(timeout)
ARGUMENTS	timeout HTTP request time out in seconds.

StartArchivePdfFile

Starts merging multiple input files into one PDF archive file.

StartArchivePdfFile

SYNTAX	StartArchivePdfFile(documentTypeName)
ARGUMENTS	documentTypeName Name of the OnBase document type.

Sample Script: Convert one Input File into one PDF File

This script converts one input file into one PDF file and sends the PDF file along with the specified keywords to the OnBase archive.

```
GetObject("OnBaseDocApi","MLOnBaseDocumentApi")
;Set configuration properties
OnBaseDocApi.SetOnBaseDocumentApiUrl("http://in01-devpdc117.pvi.com/ApiServer/onbase/core/")
;Set Compart properties (only needed when converting to PDF)
OnBaseDocApi.SetLicenseFilePath("%Composerdir%\cpsdk.lic")
OnBaseDocApi.SetProfilePath("%Composerdir%\profiles\")
;Set OnBase keywords
OnBaseDocApi.ClearKeywords()
```

```
OnBaseDocApi.SetKeyword("Description", "test description") ; string value

OnBaseDocApi.SetKeyword("Name", "test-name") ; string value

OnBaseDocApi.SetKeywordDate("Launch Date (Date Only)", "24.12.2020", "dd.MM.yyyy") ; date value

OnBaseDocApi.SetKeywordNumber("gross tonnage (decimal accepted)", "1.105,99", "de") ; float value

OnBaseDocApi.SetKeywordNumber("length (ft)", "11", "de") ; numeric value

OnBaseDocApi.SetKeywordNumber("cost", "5,000.98", "en") ; currency value

OnBaseDocApi.SetKeywordDateTime("maiden voyage (date & time)", "1999-01-30 23:55:55", "yyyy-MM-dd
HH:mm:ss") ; date time value

;Set OnBase meta data

OnBaseDocApi.SetDocumentComment("Test COMP-6138")

;Optional: specify a document date. Default value is the current date

;OnBaseDocApi.SetDocumentDate("05.05.1999", "dd.MM.yyyy")

;Archive one file (no conversion to pdf)

onBaseDocumentTypeName = "HylandAPIIntegration"

inputFilePath = "c:\data\temp\hello_world.xps"

docId = OnBaseDocApi.ArchivePdfFile(inputFilePath, onBaseDocumentTypeName)

Protocol("OnBase document id: " + docId, 5)

FreeObject("OnBaseDocApi")
```

Sample Script: Combine Multiple Input Files into one PDF File

This script combines multiple input files into one PDF file and sends the PDF file along with the specified keywords to the OnBase archive.

```
GetObject("OnBaseDocApi","MLOnBaseDocumentApi")

;Set configuration properties

;OnBaseDocApi.SetOnBaseDocumentApiUrl("http://in01-devpdc117.pvi.com/ApiServer/onbase/core/")

url = OnBaseDocApi.SetOnBaseDocumentApiUrlFromConfiguration() ;reads URL of OnBase Document API
from app setting 'onbaseRestApiUrl'

;Set Compart properties (only needed when converting to PDF)

OnBaseDocApi.SetLicenseFilePath("%Composerdir%\cpsdk.lic")

OnBaseDocApi.SetProfilePath("%Composerdir%\profiles\")

;Set OnBase keywords
```

```
OnBaseDocApi.ClearKeywords()
OnBaseDocApi.SetKeyword("Description", "test description") ; string value
OnBaseDocApi.SetKeyword("Name", "test-name") ; string value
OnBaseDocApi.SetKeywordDate("Launch Date (Date Only)", "24.12.2020", "dd.MM.yyyy") ; date value
OnBaseDocApi.SetKeywordNumber("gross tonnage (decimal accepted)", "1.105,99", "de") ; float value
OnBaseDocApi.SetKeywordNumber("length (ft)", "11", "de") ; numeric value
OnBaseDocApi.SetKeywordNumber("cost", "5,000.98", "en") ; currency value
OnBaseDocApi.SetKeywordDateTime("maiden voyage (date & time)", "1999-01-30 23:55:55", "yyyy-MM-dd HH:mm:ss") ; date time value
;Set OnBase meta data
OnBaseDocApi.SetDocumentComment("Test comment")
;Optional: specify a document date. Default value is the current date
;OnBaseDocApi.SetDocumentDate("05.05.1999", "dd.MM.yyyy")
onBaseDocumentTypeName = "HylandAPIIntegration"
OnBaseDocApi.StartArchivePdfFile(onBaseDocumentTypeName)
OnBaseDocApi.AddFile("c:\data\temp\hello_world_1.xps")
OnBaseDocApi.AddFile("c:\data\temp\hello_world_2.xps")
OnBaseDocApi.AddFile("c:\data\temp\hello_world_3.xps")
docId = OnBaseDocApi.FinishArchivePdfFile()
Protocol("OnBase document id: " + docId, 5)
FreeObject("OnBaseDocApi")
```

Sample Script: Send one File to the OnBase Archive

This script sends one file along with the specified keywords to the OnBase archive without converting the file format of the input file.

```
GetObject("OnBaseDocApi","MLOnBaseDocumentApi")
;Set configuration properties
OnBaseDocApi.SetOnBaseDocumentApiUrl("http://in01-devpdc117.pvi.com/ApiServer/onbase/core/")
;OnBaseDocApi.SetOnBaseDocumentApiUrlFromConfiguration() ;reads URL of OnBase Document API from app setting 'onbaseRestApiUrl'
;Set OnBase keywords
```

```
OnBaseDocApi.ClearKeywords()

OnBaseDocApi.SetKeyword("Description", "test description") ; string value

OnBaseDocApi.SetKeyword("Name", "test-name") ; string value

OnBaseDocApi.SetKeywordDate("Launch Date (Date Only)", "24.12.2020", "dd.MM.yyyy") ; date value

OnBaseDocApi.SetKeywordNumber("gross tonnage (decimal accepted)", "1.105,99", "de") ; float value

OnBaseDocApi.SetKeywordNumber("length (ft)", "11", "de") ; numeric value

OnBaseDocApi.SetKeywordNumber("cost", "5,000.98", "en") ; currency value

OnBaseDocApi.SetKeywordDateTime("maiden voyage (date & time)", "1999-01-30 23:55:55", "yyyy-MM-dd
HH:mm:ss") ; date time value

;Set OnBase meta data

OnBaseDocApi.SetDocumentComment("Test comment")

;Optional: specify a document date. Default value is the current date

;OnBaseDocApi.SetDocumentDate("05.05.1999", "dd.MM.yyyy")

;Archive one file (no file format conversion for input file)

onBaseDocumentTypeName = "HylandAPIIntegration"

inputFilePath = "c:\data\temp\hello_world.pdf"

docId = OnBaseDocApi.ArchiveFile(inputFilePath, onBaseDocumentTypeName)

Protocol("OnBase document id: " + docId, 5)

FreeObject("OnBaseDocApi")
```

HTTP API

About HTTP API

The HTTP Monalisa HTTP API consists of the following classes:

- MLHttpClient
- MLHttpRequest
- MLHttpResponse

MLHttpClient

About MLHttpClient

The class MLHttpClient provides a set of Monalisa functions for executing HTTP calls. You can use it to call HTTP methods of a third-party HTTP server or a REST API.

To load the object, use the MonaLisa function GetObject.

Example

```
GetObject("Client","MLHttpClient")
```

To free an instance of this class, use the MonaLisa function FreeObject.

Functions of the Class MLHttpClient

AddDefaultHeader

Adds an HTTP header to be sent with each HTTP request. If a header with this name already exists, the value is added to this header.

AddDefaultHeader		
SYNTAX	AddDefaultHeader(string headerName, string headerValue)	
ARGUMENTS	headerName	Name of the HTTP header.
	headerValue	Value of the HTTP header.

ConvertFromBase64

Converts a Base64 encoded string to a string.

ConvertFromBase64		
SYNTAX	ConvertFromBase64(string base64)	
ARGUMENTS	base64	The Base64 encoded string.
RETURN	The decoded string.	

ConvertToBase64

Converts a string to a Base64 encoded string.

ConvertToBase64	
SYNTAX	ConvertToBase64(string input)
ARGUMENTS	input A string.
RETURN	The Base64 encoded string.

CreateRequest

Creates a new HTTP request.

CreateRequest	
SYNTAX	CreateRequest()
RETURN	A new instance of MLHttpRequest. to do link

Delete

Executes a HTTP DELETE request.

Delete	
SYNTAX	Delete(MLHttpRequest request)
ARGUMENTS	request An instance of MLHttpRequest.
RETURN	An instance of MLHttpResponse containing the response from the server. to do link

Get

Executes a HTTP GET request.

Get

SYNTAX	Get(MLHttpRequest request)
ARGUMENTS	request — An instance of MLHttpRequest.
RETURN	An instance of MLHttpResponse containing the response from the server.

GetCookie

Returns the value of the specified HTTP cookie.

GetCookie

SYNTAX	GetCookie(string cookieName)
ARGUMENTS	cookieName — Cookie name.
RETURN	The cookie value or an empty string if the cookie does not exist.

GetDefaultHeader

Returns the HTTP header value at the specified index.

GetDefaultHeader

SYNTAX	GetDefaultHeader(string headerName, int index)
ARGUMENTS	headerName — Name of the HTTP header. index — A zero based index.
RETURN	The value of the HTTP header at the specified index position or an empty string if no value exists for the specified index.

GetDefaultHeaderCount

Returns the number of values for of this HTTP header name.

GetDefaultHeaderCount

SYNTAX GetDefaultHeaderCount(string headerName)

ARGUMENTS

headerName	Name of the HTTP header.

RETURN Number of header values or 0 (zero) if the specified header name does not exist.

Post

Executes a HTTP POST request.

Post

SYNTAX Post(MLHttpRequest request)

ARGUMENTS

request	An instance of MLHttpRequest.

RETURN An instance of MLHttpResponse containing the response from the server.

Put

Executes a HTTP PUT request.

Put

SYNTAX Put(MLHttpRequest request)

ARGUMENTS

request	An instance of MLHttpRequest.

RETURN An instance of MLHttpResponse containing the response from the server.

RemoveDefaultHeader

Removes the specified HTTP header.

RemoveDefaultHeader		
SYNTAX	RemoveDefaultHeader(string headerName)	
ARGUMENTS	headerName	Name of the HTTP header.
RETURN	TRUE: Successfully removed the header. FALSE: The header was not found.	

SetBaseAddress

Sets the HTTP DNS host name or IP address. This property may include the service IP port number if it differs from the default port for the URI.

SetBaseAddress		
SYNTAX	SetBaseAddress(string baseAddress)	
ARGUMENTS	baseAddress	The HTTP DNS hostname with optional IP port number.

SetCookie

Sets the value of the specified HTTP cookie.

SetCookie		
SYNTAX	SetCookie(string cookieName, string cookieValue)	
ARGUMENTS	cookieName	The Cookie name.
	cookieValue	The Cookie value.

SetIdpTokenMinValidPeriodMinutes

Sets the minimum validity period of the IDP token in minutes.

SetIdpTokenMinValidPeriodMinutes

SYNTAX SetIdpTokenMinValidPeriodMinutes(double minValidPeriodMinutes)

ARGUMENTS

minValidPeriodMinutes	Minimum period of validity of the token in minutes.

SetRequestTimeout

Sets the seconds to wait before the HTTP request times out. The default value is 120 seconds.

SetRequestTimeout

SYNTAX SetRequestTimeout(int timeout)

ARGUMENTS

timeout	HTTP request timeout in seconds.

UseOAuth2ClientCredentials

Executes an OAuth2 authentication of type **Client Credential**. Authenticates against the IdP server and then sets the HTTP authorization header.

UseOAuth2ClientCredentials

SYNTAX UseOAuth2ClientCredentials(string authority, string clientId, string scope, string encryptedIdpClientSecret)

ARGUMENTS

authority	The IdP authority string.
clientId	The IdP client ID.
scope	The IdP scope string.
encryptedIdpClientSecret	The encrypted IdP client secret. Use the Content Composer tool **Encoder** to encrypt this value.

UseOAuth2PasswordGrant

Executes an OAuth2 authentication of type **Password Grant** Authenticates against the IdP server and then sets the HTTP authorization header.

UseOAuth2PasswordGrant

SYNTAX UseOAuth2PasswordGrant(string authority, string clientId, string scope, string userName, string encryptedIdpPassword, string encryptedIdpClientSecret)

ARGUMENTS

authority	The IdP authority string.
clientId	The IdP client ID.
scope	The IdP scope string.
userName	The IdP user name.
encryptedIdpPassword	The encrypted password of the IdP user. Use the Content Composer tool **Encoder** to encrypt this value.
encryptedIdpClientSecret	The encrypted IdP client secret. Use the Content Composer tool **Encoder** to encrypt this value.

MLHttpRequest

About MLHttpRequest

This class is a MonaLisa wrapper for a HTTP request.

To create an HTTP request:

1. Call the Monalisa function CreateRequest of the class MLHttpClient.
2. Prepare the HTTP request.
3. Pass the HTTP request to one of the following methods of the class MLHttpClient:

- Delete
- Get
- Post
- Put

To load the object, use the MonaLisa function GetObject.

Example

```
GetObject("Request","MLHttpRequest")
```

To free an instance of this class, use the MonaLisa function FreeObject.

Functions of the Class MLHttpRequest

AddHeader

Adds an HTTP header that is sent only with this HTTP request. If a header with this name already exists, the value is added to this header.

AddHeader		
SYNTAX	AddHeader(string headerName, string headerValue)	
ARGUMENTS	headerName	Name of the HTTP header.
	headerValue	Value of the HTTP header.

SetByteContent

SetByteContent(byte[] bytes)

Assigns a byte array to the content (body) of the HTTP request.

SetByteContent		
SYNTAX	SetByteContent(byte[] bytes)	
ARGUMENTS	bytes	A byte array.

SetByteContent(Byte[], Int32, Int32)

Assigns the specified chunk of the byte array to the content (body) of the HTTP request. Use this function to upload only a chunk of a byte array.

SetByteContent

SYNTAX	SetByteContent(byte[] bytes, int offset, int count)
ARGUMENTS	bytes — The complete byte array. offset — The offset in bytes. This is the start position of the chunk. count — The number of bytes. This is the size of the chunk in the bytes.

SetByteContentFromFile

Assigns the content of a file to the content (body) of the HTTP request.

SetByteContentFromFile

SYNTAX	SetByteContentFromFile(string filePath)
ARGUMENTS	filePath — The file to load.

SetStringContent

Assigns a string to the content (body) of the HTTP request.

SetStringContent

SYNTAX	SetStringContent(string content, string contentType)
ARGUMENTS	content — String for the HTTP content (body). contentType — Specifies the media content type of the body. **Example of possible values:** • 'application/json' • 'application/xml'

SetUri

Sets the URI used for the HTTP request. If the request URI is a relative URI, it is combined with the base address of the HTTP client.

SetUri		
SYNTAX	SetUri(string requestUri)	
ARGUMENTS	requestUri	Request URI.

MLHttpResponse

About MLHttpResponse

This class is a Monalisa wrapper for a HTTP response.

The following methods of the class **MLHttpClient** return an instance of **MLHttpResponse**:

- Delete
- Get
- Post
- Put

To load the object, use the MonaLisa function GetObject.

Example

```
GetObject("Response","MLHttpResponse")
```

To free an instance of this class, use the MonaLisa function FreeObject.
Functions of the Class MLHttpResponse

GetByteContent

Returns the content (body) of the response message as a byte array.

GetByteContent	
SYNTAX	GetByteContent()
RETURN	The response body content as a byte array.

GetHeader

Returns the response header value at the specified index.

GetHeader		
SYNTAX	GetHeader(string headerName, int index)	
ARGUMENTS	headerName	Name of the HTTP header.
	index	A zero based index.
RETURN	The value of the HTTP header at the specified index position or an empty string if no value exists for the specified index.	

GetHeaderCount

Returns the number of values for this HTTP response header name. Returns zero if the HTTP header does not exist.

GetHeaderCount		
SYNTAX	GetHeaderCount(string headerName)	
ARGUMENTS	headerName	Name of the HTTP header.
RETURN	Number of header values or 0 (zero) if the specified header name does not exist.	

GetReasonPhrase

Returns the reason phrase that is usually sent by servers along with the status code.

GetReasonPhrase	
SYNTAX	GetReasonPhrase()
RETURN	The reason phrase sent by the server.

GetStatusCode

Returns the status code of the HTTP response.

GetStatusCode	
SYNTAX	GetStatusCode()
RETURN	The status code of the HTTP response.

GetStringContent

Returns the content (body) of the response message as a string.

GetStringContent	
SYNTAX	GetStringContent()
RETURN	The response body content as a string.

IsSuccessStatusCode

Gets a value indicating whether the HTTP response was successful.

IsSuccessStatusCode	
SYNTAX	IsSuccessStatusCode()
RETURN	TRUE: The status code is in the range between 200 and 299. FALSE: The HTTP response was not successful

SaveByteContentToFile

Saves the content (body) of the response message as a byte array to a file.

SaveByteContentToFile	
SYNTAX	SaveByteContentToFile(string filePath)

ARGUMENTS	filePath	Path of the output file.

Sample Script HTTP API

This script calls the public and free REST API 'CatFact' and executes a HTTP GET for the URI `https://catfact.ninja/fact`.

The response is a JSON object that contains a random fact about cats in the property `fact`.

```
GetObject("httpClient", "MLHttpClient")
GetObject("json", "MLJson")
try
  httpClient.SetBaseAddress("https://catfact.ninja")
  httpRequest = httpClient.CreateRequest()
  httpRequest.SetUri("/fact")
  httpResponse = httpClient.Get(httpRequest)
  if (httpResponse.IsSuccessStatusCode())
    ; Success
    ; Use JSON API to read the fact about cats from JSON property 'fact'
    response = httpResponse.GetStringContent()
    json.LoadString(response)
    catFact = json.GetStringValue("fact")
    Protocol("The fact about cats is: {0}", 5, catFact)
  else
    ; Error handler
    statusCode = httpResponse.GetStatusCode()
    reasonPhrase = httpResponse.GetReasonPhrase()
    Protocol("The GET call failed with reason phrase {0}, status code {1}", 5, reasonPhrase, statusCode)
  end-if
finally
```

```
  FreeObject("httpResponse")
  FreeObject("httpRequest")
  FreeObject("httpClient")
end-try
```

JSON API

About JSON API

The Monalisa JSON API consists of the following classes:

- MLJson
- MLJsonArray
- MLJsonObject

Some functions use a `JSON Path` string to select a specific subset of the JSON data structure.

`JSON Path` is a query language for JSON, similar to XPath for XML. See the JSON Path online documentation for more information.

MLJson

About MLJson

This class provides MonaLisa functions to read, create, and modify a JSON data structure.

To load the object, use the MonaLisa function GetObject.

Example

```
GetObject("JSON","MLJson")
```

Functions of the Class MLJson

AddArray

Adds a JSON array as a child of the JSON object specified with `jsonPath`.

AddArray		
SYNTAX	AddArray(string jsonPath, string arrayName)	
ARGUMENTS	jsonPath	The JSON path to identify the root object in the JSON structure. Pass an empty string to add the array under the top-level root object.

arrayName	Name of the new array.

RETURN The added JSON array.

AddObject

Adds a JSON object as a child of the JSON object specified with `jsonPath`.

AddObject

SYNTAX AddObject(string jsonPath, string objectName)

ARGUMENTS

jsonPath	The JSON path to identify the root object in the JSON structure. Pass an empty string to add the object under the top-level root object.
objectName	Name of the new object.

RETURN The added JSON object.

AddPropertyValue

Adds a JSON property as a child of the JSON object specified with `jsonPath`.

AddPropertyValue

SYNTAX AddPropertyValue(string jsonPath, string propertyName, object propertyValue)

ARGUMENTS

jsonPath	The JSON path to identify the root object in the JSON structure. Pass an empty string to add the property under the top-level root object.
propertyName	Name of the new property.
propertyValue	Value of the new property.

Clear

Removes everything from this JSON structure.

Clear	
SYNTAX	Clear()

Exists

Checks if the specified JSON path exists.

Exists		
SYNTAX	Exists(string jsonPath)	
ARGUMENTS	jsonPath	The JSON path to check.
RETURN	TRUE: Path exists. FALSE: Path does not exist.	

GetArray

Returns an instance of type **MLJsonArray** that offers an API for a JSON array.

GetArray		
SYNTAX	GetArray(string jsonPath)	
ARGUMENTS	jsonPath	The JSON path to identify the array in the JSON structure.
RETURN	The array specified with jsonPath.	

GetObject

Returns an instance of type MLJsonObject that offers an API for a JSON object.

GetObject	
SYNTAX	GetObject(string jsonPath)
ARGUMENTS	jsonPath — The JSON path to identify the object in the JSON structure.
RETURN	The object specified with jsonPath

GetStringValue

Returns a JSON property value as a string. Readable data types of the JSON property are: String, Number, and Boolean.

GetStringValue	
SYNTAX	GetStringValue(string jsonPath)
ARGUMENTS	jsonPath — The JSON path to identify the property in the JSON structure.
RETURN	The property value or an empty string for the JSON value `null`.

LoadArrayFromFile

Loads a JSON array from a file. Use this function if the JSON string starts with **[**.

LoadArrayFromFile	
SYNTAX	LoadArrayFromFile(string jsonFilePath)
ARGUMENTS	jsonFilePath — The path of the JSON file. Environment variables are supported.
RETURN	The loaded array.

LoadArrayFromString

Loads a JSON array from string. Use this function if the JSON string starts with **[**.

LoadArrayFromFile		
SYNTAX	LoadArrayFromString(string json)	
ARGUMENTS	json	The JSON string.
RETURN	The loaded array.	

LoadFile

Loads a JSON object from file. Use this function if the JSON string starts with **{**.

LoadFile		
SYNTAX	LoadFile(string jsonFilePath)	
ARGUMENTS	jsonFilePath	The path of the JSON file. Environment variables are supported.

LoadString

Loads a JSON object from string. Use this function if the JSON string starts with **{**.

LoadString		
SYNTAX	LoadString(string json)	
ARGUMENTS	json	The JSON string.

Remove

Removes the property, array, or object specified with the JSON path.

Remove

SYNTAX	Remove(string jsonPath)
ARGUMENTS	jsonPath — The JSON path identifying the element to be removed from the JSON structure.
RETURN	TRUE: Successfully removed. FALSE: The JSON path does not exist.

SaveToFile

Saves the current JSON object to a file.

LoadString	
SYNTAX	SaveToFile(string jsonFilePath)
ARGUMENTS	jsonFilePath — The path of the JSON file. Environment variables are supported.

SetPropertyValue

Sets a value for an existing JSON property of data type String, Number, or Boolean.

SetPropertyValue	
SYNTAX	SetPropertyValue(string jsonPath, object value)
ARGUMENTS	jsonPath — The JSON path to identify the property. value — The property value to set.

ToString

Returns the current JSON structure as a string.

ToString	
SYNTAX	ToString()
RETURN	The JSON string.

MLJsonArray

About MLJsonArray

This class provides MonaLisa functions to create, read, and modify a JSON array.

To create an instance of this class, call a Monalisa JSON function that returns an MLJson array.

Note: It is not possible to use the function **GetObject** to create an instance of MLJsonArray in a MonaLisa script.

Functions of the Class MLJsonArray

AddArray

Adds a new array to this array at the specified index.

AddArray		
SYNTAX	AddArray(int index)	
ARGUMENTS	index	The zero based index. To add the array at the end of this array, pass **-1** .
RETURN	The added JSON array.	

AddElementValue

Adds a new array element. Use it to set the JSON data types String, Number, and Boolean.

AddElementValue	
SYNTAX	AddElementValue(int index, object value)

ARGUMENTS	index	The zero based index. Pass -1 for the index to add the array at the end of this array.
	value	The value of the new array element.

AddObject

Adds a new JSON object to the array at the specified zero based index.

AddObject	
SYNTAX	AddObject(int index)
ARGUMENTS	index: The zero based index. Pass -1 for the index to add the array at the end of this array.
RETURN	The added JSON object.

Clear

Removes all items from this JSON array.

Clear	
SYNTAX	Clear()

GetArray

Returns the JSON array at the specified array index.

GetArray	
SYNTAX	GetArray(int index)
ARGUMENTS	index: The zero based index.

RETURN	The JSON array.

GetCount

Returns the number of array items.

GetCount	
SYNTAX	GetCount()
RETURN	The number of array items.

GetObject

Returns the JSON object at the specified array index.

GetObject		
SYNTAX	GetObject(int index)	
ARGUMENTS	index	The zero based index.
RETURN	The JSON object.	

GetStringValue

Returns the JSON property value at the specified array index as a string.

GetStringValue		
SYNTAX	GetStringValue(int index)	
ARGUMENTS	index	The zero based index.
RETURN	The property value or an empty string for the JSON value `null`.	

Remove

Removes the array item at the specified index.

Remove		
SYNTAX	Remove(int index)	
ARGUMENTS	index	The zero based index.
RETURN	TRUE: Successfully removed. FALSE: The item does not exist.	

SaveToFile

Saves this JSON array to a file.

SaveToFile		
SYNTAX	SaveToFile(string jsonFilePath)	
ARGUMENTS	jsonFilePath	The JSON path identifying the element to be removed from the JSON structure.
RETURN	The path to the JSON file. Environment variables are supported.	

SetElementValue

Sets a value for an existing array element. Use it to set the JSON data types String, Number, and Boolean.

SetElementValue		
SYNTAX	SetElementValue(int index, object value)	
ARGUMENTS	index	The zero based index.
	value	The new value of the array element.

ToString

Returns the current JSON array as a string.

ToString	
SYNTAX	ToString()
RETURN	A JSON string.

MLJsonObject

About MLJsonObject

This class provides MonaLisa functions to read, create, and modify a JSON object.

You create an instance of this class by calling a Monalisa JSON function that returns a MLJson object.

Note: It is not possible to use the function **GetObject** to create an instance of MLJsonArray in a MonaLisa script.

Functions of the Class MLJsonObject

AddArray

Adds a new JSON array property to this object.

AddArray		
SYNTAX	AddArray(string propertyName)	
ARGUMENTS	propertyName	The name of the new property.
RETURN	The new JSON array.	

AddObject

Adds a new JSON object property to this object.

AddObject

SYNTAX	AddObject(string propertyName)
ARGUMENTS	propertyName — The name of the new property.
RETURN	The new JSON object.

AddPropertyValue

Adds a new property. Use it to set the JSON data types String, Number, and Boolean.

AddPropertyValue

SYNTAX	AddPropertyValue(string propertyName, object value)
ARGUMENTS	propertyName — The name of the new property. value — The value of the new property.

Clear

Removes all properties from this JSON object.

Remove

SYNTAX	Clear()

Exists

Checks whether the JSON property with the specified name exists.

Exists

SYNTAX	Exists(string propertyName)
ARGUMENTS	propertyName — The property name.

RETURN	TRUE: The property exists. FALSE: The property does not exist.

GetArray

Returns the JSON array of the specified property.

GetArray		
SYNTAX	GetArray(string propertyName)	
ARGUMENTS	propertyName	The property name.
RETURN	The JSON array.	

GetObject

Returns the JSON object of the specified property.

GetObject		
SYNTAX	GetObject(string propertyName)	
ARGUMENTS	propertyName	The property name.
RETURN	The JSON object.	

GetStringValue

Returns the value of the specified JSON property as a string.

GetStringValue	
SYNTAX	GetStringValue(string propertyName)

ARGUMENTS	propertyName	The property name.
RETURN	The property value or an empty string for the JSON value `null`.	

Remove

Removes the specified property.

Remove		
SYNTAX	Remove(string propertyName)	
ARGUMENTS	propertyName	The property name.
RETURN	TRUE: Successfully removed. FALSE: The property does not exist.	

SetPropertyValue

Sets the value of an existing property. Use it to set the JSON data types String, Number, and Boolean.

SetPropertyValue		
SYNTAX	SetPropertyValue(string propertyName, object value)	
ARGUMENTS	propertyName	The property name.
	value	The new value of the property.

ToString

Returns the JSON object as a string.

ToString	
SYNTAX	ToString()
RETURN	The JSON object as string.

Sample Script JSON API: Create Array

This script creates the following JSON structure with a top level array:

```
;[
; {
;   "name": "BMW",
;   "color": "red"
; },
; {
;   "name": "Mercedes",
;   "color": "green"
; }
;]
```

```
GetObject("json","MLJson")
; Create an empty top level array
jsonArray = json.LoadArrayFromString("[]")
; Add an object element to array
car1 = jsonArray.AddObject(-1)
car1.AddPropertyValue("name", "BMW")
car1.AddPropertyValue("color", "red")
;add a second object element to array
car2 = jsonArray.AddObject(-1)
car2.AddPropertyValue("name", "Mercedes")
car2.AddPropertyValue("color", "green")
; Get JSON string
jsonString = jsonArray.ToString()
; Save JSON string to file
jsonArray.SaveToFile("%USERPROFILE%\Json_Create_Array_Output.json")
```

Sample Script JSON API: Create Object

```
; This script creates the following JSON structure with a top level object:
;{
;  "sample_string": "Hello World",
;  "sample_bool": true,
;  "sample_number": 55,
;  "sample_float": 59.99,
;  "sample_date-ISO8601": "2022-11-22T17:42:50.1593057+01:00",
;  "sample_object": {
;    "id": "12345",
;    "names": [
;      "Sarah",
;      "Jonathan"
;    ]
;  }
;}

GetObject("json","MLJson")
; Adds a string property to root object
testString = "Hello World"
json.AddPropertyValue("", "sample_string", testString)
; Adds a boolean property to root object
testBool = true
json.AddPropertyValue("", "sample_bool", testBool)
; Adds an integer property to root object
testNumber = 55
json.AddPropertyValue("", "sample_number", testNumber)
; Adds a float property to root object
testFloat = 59.99
```

```
json.AddPropertyValue("", "sample_float", testFloat)
; Adds a date property to root using  ISO 8601 date string
GetObject("Date","MLDate")
CurrDateISO8601 = date.CurrentDateToString("o")
json.AddPropertyValue("", "sample_date-ISO8601", CurrDateISO8601)
; Add an object under the root object
newObject = json.AddObject("", "sample_object")
newObject.AddPropertyValue("id", "12345")
; Add an array to this object
newArray = newObject.AddArray("names")
newArray.AddElementValue(-1, "Sarah")
newArray.AddElementValue(-1, "Jonathan")
; Get JSON string
jsonString = json.ToString()
; Save JSON string to file
json.SaveToFile("%USERPROFILE%\Json_Create_Object_Output.json")
```

Sample Script JSON API: Read Data

This script reads data from the following JSON structure:

```
;{
;  "customer": {
;    "firstname": "mary",
;    "lastname": "higgins"
;  },
;  "cars": [
;    {
;     "color": "blue",
;     "name": "Ford",
;     "owners": [ "Mary", "Jon", "Snoopy" ]
;    },
```

```
;   {
;     "color": "red",
;     "name": "Bmw",
;     "owners": [ "Ben", "Sue"]
;   }
; ]
;}

GetObject("json", "MLJson")
json.LoadFile("%USERPROFILE%\Json_Read_Input.json")
; Direct access to read a property using Json Path
Firstname = json.GetStringValue("customer.firstname"); returns 'mary'
SecondOwnerOfFord = json.GetStringValue("cars[0].owners[1]") ;returns 'Jon'
; Loop through 'cars' array
Cars = json.GetArray("cars")
MaxIndexCar = Cars.GetCount()
IndexCar=0
while (IndexCar < MaxIndexCar)
  Car = Cars.GetObject(IndexCar);get 'car' item
  CarName = Car.GetStringValue("name")
  Protocol("owners of car: {0}", 5, CarName)
  ; Loop through array of car owners
  Owners = Car.GetArray("owners")
  MaxIndexOwner = Owners.GetCount()
  IndexOwner = 0
  while (IndexOwner < MaxIndexOwner)
    OwnerName = Owners.GetStringValue(IndexOwner)
    Protocol("owner name: {0}", 5, OwnerName)
    IndexOwner = IndexOwner + 1
  end-while
```

```
  IndexCar = IndexCar + 1
end-while
```

Properties and Functions of Contexts

Context FlatFileConverter

Properties of the Context FlatFileConverter

The following table describes the properties of context FlatFileConverter.

Property	Description
SaveXMLtoFile	This property is read and writable. It can have values of `TRUE` and `FALSE`. • **TRUE**. If `TRUE`, the XML content is saved to a file by the FlatFileConverter component by default. • **FALSE**. If `FALSE`, the script saves the XML content to a different data store.
XMLContent	This property contains the XML output string that will be saved to a file. This property is readonly.
IsMultiXML	This property contains the XML output string that is saved to a file. This property is readonly.
JobElementPath	The value provided in ParserDefinition-Attribut "jobelementpath". This property is readonly.
ConfigurationName	The name of the currently used parser definition. This property is readonly.

Sample Script for context FlatFileConverter

In case, the XML is saved to a file, no script is needed because this is the default processing. If the XML is written to a DB an appropriate script is needed.

This sample script shows how to write the data to an example DB. You can modify the DB table and column names accordingly.

The default setting is `true` which results in saving the XML to file.

```
SaveXmlToFile = false


For testing purpose
```

```
;if (SaveXmlToFile = false)
;  ModusProcess = SystemPool.MyProcess
;  ModusProcess.SendInfoMessage(FormatStr("writing XmlContent to DB"))
;  Protocol("===== {0}: writing XmlContent to DB", 8, ModusProcess.InstanceId )
;end-if

Example for getting data from the generated XML
if (XmlContent <> "")
 if ((ConfigurationName = "default"))
   GetObject("xml", "MLXmlDocument")
   xml.LoadXml(XmlContent)
   root = xml.DocumentElement
   ; just for testing purpose getting value for CustomerId, CustomerName
   ; and count of Orders
   KundenNummer = xml.GetXPathValue(root, "/Customer/@CustomerId")
   KundenName = xml.GetXPathValue(root, "/Customer/@CustomerName")
   orderList = root.SelectNodes("/Customer/Order")
   orderCount = orderList.Count
 end-if
end-if
InitDatabase("AUFTRAGSDATEN")
InitDataSet("modusauftrag", "AUFTRAGSDATEN")
; setting test values
State = 7
BdlName = "FF2Xml_BdlName"
SelName = "FF2Xml_SelName"
SysOid = "FF2Xml_SysOid"
SelParams = "xmlData"
;Contextvariable 'XmlContent' holds the resulting Xml-String
Data = XmlContent
```

```
StringListClear("SqlDefList")
StringListAdd("SqlDefList", "Insert into modus_xml_job ")
StringListAdd("SqlDefList", "(State, BundleName, SelectionName, SystemOId, SelParams, Data) ")
StringListAdd("SqlDefList", "values ")
StringListAdd("SqlDefList", "(:State, :BdlName, :SelName, :SysOid, :SelParams, :Data)")
StringListClear("ParamDescList")
StringListSetValue("ParamDescList", "State", "ii") ; integer
StringListSetValue("ParamDescList", "BdlName", "is") ; string
StringListSetValue("ParamDescList", "SelName", "is") ; string
StringListSetValue("ParamDescList", "SysOid", "is") ; string
StringListSetValue("ParamDescList", "SelParams", "is") ; string
StringListSetValue("ParamDescList", "Data", "is") ; string
SetExecSQL("modusauftrag", "SqlDefList", "ParamDescList")
```

Context ManualVariable

About the Context ManualVariable

Use this context for scripts assigned to text block variables that contain a value help.

Properties of the Context ManualVariable

Property	Description
AssignedValueHelpType	Type of value help assigned to a text block variable. Read-only property. **Possible values** Value: ValueHelpType.Unknown — Description: No value help assigned. Value: ValueHelpType.Textbox — Description: Text box with input mask.

ValueHelpType.Combobox	Drop-down list
ValueHelpType.Calendar	Popup calendar (DateTimePicker)
ValueHelpType.SpinEdit	Numeric input field (SpinEdit)
ValueHelpType.MultilineTextbox	Multiline textbox

You can specify value help settings depending on the returned value, using the appropriate object with its properties and functions.

Example

```
if (AssignedValueHelpType = ValueHelpType.Textbox)
...
end-if
```

Object	Description
Textbox	Allows you to query or change property values of the text box with input mask value help within the script, if `AssignedValueHelpType` returns the value `ValueHelpType.Textbox`.

Property	Description
Textbox.InputMask	Sets or gets the **InputMask** value.

Example

```
if (AssignedValueHelpType = ValueHelpType.Textbox)
Textbox.InputMask = "00"
end-if
```

Combobox	Allows you to query or change property values or to call functions of the drop-down value help within the script, if `AssignedValueHelpType` returns the value `ValueHelpType.Combobox`.

Property	Description
Combobox.Sorted	Sets or gets the **Sorted** flag of the valuehelp (Boolean).
Combobox.ItemCount	Gets the number of items from the items collection.

Functions

AddItem

ClearItems

ItemExists

Example

```
if (AssignedValueHelpType = ValueHelpType.Combobox)
cnt = Combobox.ItemCount
itemisadded = Combobox.AddItem("Max")
Combobox.Sorted = true
end-if
```

DateTime Picker

Allows you to query or change property values of the popup calendar value help within the script, if `AssignedValueHelpType` returns the value `ValueHelpType.Calendar`.

Property	Description
DateTimePicker.CustomFormat	Sets or gets the **CustomFormat** format specifier.
DateTimePicker.DateTimeFormat	Sets or gets the **DateTimeFormat**. **Possible values** • Short • Long • Custom

Example

```
if (AssignedValueHelpType = ValueHelpType.Calendar)
DateTimePicker.CustomFormat = "dd.MM.yyyy"
if (DateTimePicker.DateTimeFormat <> "Custom")
DateTimePicker.DateTimeFormat = "Custom"
end-if
end-if
```

SpinEdit

Allows you to query or change property values of the numeric input field value help within the script, if `AssignedValueHelpType` returns the value `ValueHelpType.SpinEdit`.

Property	Description
SpinEdit.DecimalPlaces	Sets or gets the **DecimalPlaces** value.
SpinEdit.Increment	Sets or gets the **Increment** value.
SpinEdit.Maximum	Sets or gets the **Maximum** value.
SpinEdit.Minimum	Sets or gets the **Minimum** value.

Example

```
if (AssignedValueHelpType = ValueHelpType.SpinEdit)
SpinEdit.DecimalPlaces = 0
SpinEdit.Increment =5
SpinEdit.Maximum = 100
SpinEdit.Minimum = 0
end-if
```

MultilineTextbox

Allows you to query or change property values of the multiline textbox value help within the script, if `AssignedValueHelpType` returns the value ValueHelpType.MultilineTextbox.

Property	Description
MultilineTextbox.MaxLength	Sets or gets the **MaxLength** value. To disable the textlength restriction, set this value to 0.

Example

```
if (AssignedValueHelpType = ValueHelpType.MultilineTextbox)
MultiLineTextbox.MaxLength = 12
end-if
```

Functions of the Context ManualVariable

AddItem

Adds the given item to the items collection.

The usage of the parameters ingoreCase and trim does not influence the item itself. If the search does not find any item, it is added as provided in the item parameter.

AddItem

SYNTAX	AddItem(item) AddItem(item, ignoreCase, trim)

ARGUMENTS

item	Item you want to add.
ignoreCase	TRUE = Search is not case-sensitive. FALSE = Search is case-sensitive.
trim	TRUE = Ignore leading and trailing blanks. FALSE = Do not ignore leading and trailing blanks.

RETURN	TRUE = The item was added. FALSE = The item could not be added because an identical item already exists.

ClearItems

Clears the items collection.

ClearItems	
SYNTAX	ClearItems()
EXAMPLE	ClearItems()

ItemExists

Searches for the given item in the items collection.

Depending on the boolean parameters ignoreCase, and trim, it is possible to search for the identical item, perform a case insensitive search, or ignore leading and trailing blanks when searching.

InitDataset		
SYNTAX	ItemExists(item, ignoreCase, trim)	
ARGUMENTS	item	Item to search for.
	ignoreCase	TRUE = Search is case insensitive. FALSE = Search is case sensitive.
	trim	TRUE = Ignore leading and trailing blanks. FALSE = Consider leading and trailing blanks.

Context ModusBatch

About the Context ModusBatch

Use this context for scripts in batch processes.

Properties of the Context ModusBatch

Property	Description

ErrorMessage	Error message. If a value is assigned, document generation aborts with the corresponding error message.
InsertObject	TRUE = Continue processing. FALSE = Stop processing.
InsertObjectS	Y = Continue processing. N = Stop processing.
DocumentCollection	Bundle to generate.
SystemOId (System Name)	Object ID of the system in which the bundle resides.
Directory	Directory in which the generated bundles are saved. Read-only property.

Functions of the Context ModusBatch

AssignData

Assigns XML data with a data ID.

AssignData

SYNTAX	AssignData(dataId, xmlData)
ARGUMENTS	dataId — Data ID xmlData — XML
EXAMPLE	`AssignData(System.DataId, System.LastXml)`

DeleteDirectory

Deletes the directory defined in the Directory property.

DeleteDirectory	
SYNTAX	DeleteDirectory()

GetData

Retrieves the XML data of the given DataID.

GetData		
SYNTAX	GetData(dataId)	
ARGUMENTS	dataId	Data ID
RETURN	XML data	
EXAMPLE	GetData(System.DataId)	

SetDataContext

Specifies the XML data to navigate.

SetDataContext		
SYNTAX	SetDataContext(xmlData)	
ARGUMENTS	xmlData	XML data
EXAMPLE	`SetDataContext(System.LastXml)`	

Context MwsProcess

About the Context MwsProcess

Use this context for scripts in MWS processes.

Properties of the Context MwsProcess

Show All

Property	Description
ChangeAction	Executes the script assigned to the event **OnStatusChange** and provides an enumerator with the following properties. Possible properties: Description None: No action Toggle: Modify status SetStructure: Refresh structure SetData: Assign data
IsObjectSet	TRUE = The object has been set or configured. FALSE = The object has not been set or configured. Read-only property.
ObjectData	XML data for data retrieval. Read-only property.
ProcessBlocked	TRUE = Forward the bundle. FALSE = Do not forward the bundle. Read-only property.
ProcessDescription	Process description.
ProcessDirectory	Directory in which the processes are saved.
ProcessId	Process ID.
ProcessTitle	Process title.

ProcessType — Process type.

ModusProcess — Provides the following properties and functions.

Possible properties	Description
ObjectName	Name of the process.
SystemOId	ID of the system in which the process runs.

Functions

Cancel

Cancels the process.

Cancel	
SYNTAX	Cancel()
EXAMPLE	`ModusProcess.Cancel()`

GetComponent

Enables access to the properties and methods of components.

GetComponent		
SYNTAX	GetComponent(string componentName)	
ARGUMENTS	componentName	Name of the component
EXAMPLE	DocumentCollection = ModusProcess.GetComponent ("DocumentCollectionManager")	

SendInfoMessage

Sends an information message to logged on remote control clients. A time stamp is inserted before each message.

SendInfoMessage		
SYNTAX	SendInfoMessage(string message)	
ARGUMENTS	message	The information message you want to send.
EXAMPLE	ModusProcess.SendInfoMessage("This is the message")	

Stop

Ends running processes.

Stop	
SYNTAX	Stop()
EXAMPLE	`ModusProcess.Stop()`

Functions of the Context MwsProcess

GetActivityStatus

Returns properties and functions for accessing an MWSActivity.

GetActivityStatus		
SYNTAX	GetActivityStatus(statusName)	
ARGUMENTS	statusName	Name of the MWSActivityStatus
RETURN	IsReady	TRUE = Activity is ready to run. FALSE = Activity is not ready to run.

IsBusy	TRUE = Activity is busy. FALSE = Activity is not busy.
IsFinished	TRUE = Activity is finished. FALSE= Activity is not finished.
IsConfigured	TRUE = Activity is configured. FALSE = Activity is not configured.
InformationXML	Information to exchange with the client.
SetLastError(int errorCode, string errorMessage)	Sets an error for the activity with an error message and error number.
EnforceConfiguration(string informationXml)	Forces the process to halt execution because additional configuration data is required.
EnforceConfiguration(string informationXml, string infoType)	Forces the process to halt execution because additional configuration data is required. The processing is returned to the client or the caller with the specified information. The information type returned is defined as MWS type in the XML.

EXAMPLE

```
DocumentCollection = ModusProcess.GetComponent("DocumentCollection")
ForwardActivity = GetActivityStatus("FORWARD")
if (ForwardActivity.IsFinished)
  Exit()
end-if
if (ForwardActivity.IsConfigured = false)
  if((DocumentCollection.IsBlocked = true) and (ProcessBlocked = true))
    ForwardActivity.EnforceConfiguration(GetForwardInformation(), "FORWARD")
    Exit()
  end-if
end-if
ForwardActivity.IsFinished = true
```

GetForwardInformation

Returns forwarding information.

GetForwardInformation	
SYNTAX	GetForwardInformation()
RETURN	XML string with DestUser and DestUserGroup.

GetObjectSelectionParameters

Returns selection parameters.

GetObjectSelectionParameters		
SYNTAX	GetObjectSelectionParameters(systemOId, folder, filter)	
ARGUMENTS	systemOId	Object ID of the system.
	folder	Subdirectory of the system.
	filter	Object type filter. Currently only "P" for bundles is possible.
RETURN	Selection parameter as XML string.	

GetOptions

Returns the options for the given OptionsID.

GetOptions		
SYNTAX	GetOptions(optionsId)	
ARGUMENTS	optionsId	Options ID.
RETURN	Options of the options ID, if found.	

IsInformationRequested

Checks whether information has been requested of the information ID that has been passed on.

IsInformationRequested		
SYNTAX	GetSectionNames(informationId)	
ARGUMENTS	informationId	Information ID.
RETURN	TRUE = Information has been requested. FALSE = Information has not been requested.	

SetObjectSelectionParameters

Sets selection parameters.

SetObjectSelectionParameters		
SYNTAX	SetObjectSelectionParameters(xmlOptions)	
ARGUMENTS	xmlOptions	XML options

Context OdinArchive

About the Context OdinArchive

Use this context in archiving scripts.

Properties of the Context OdinArchive

Show All

Parameter	Description
CurrentEnvelope	Current envelope, if found. Read-only property.

Functions

GetValue

Returns the value of the given column.

GetValue	
SYNTAX	GetValue(string columnName)
ARGUMENTS	columnName Column name
RETURN	Column value If the data type of the database column is DATE, DATETIME or any valid date type, the date read generally returns in the format MM/DD/YYYY hh:mm:ss.
EXAMPLE	GetObject("DateVar","MLDate") … StringDateVar = CurrentEnvelope.GetValue("ODE_CREATE_AT") ;Convert date to the required format with the help of the MLDate object IntDateVar = DateVar.DateStringToInt64("MM/dd/yyyy hh:mm:ss", StringDateVar) v_LETTERDATE= DateVar.Int64ToDateString("MM.dd.yyyy hh:mm:ss", IntDateVar)

GetFieldNameEnumerator

Returns an enumerator.

GetFieldNameEnumerator	
SYNTAX	GetFieldNameEnumerator()
RETURN	Enumerator with the following methods.

Method	Description
Current	Returns the current object.
MoveNext	Moves the position to the next object to be processed.
Reset	Resets the enumerator to the beginning of the first element.

CurrentJob

Current job, if found.

Read-only property.

Functions

GetValue

Returns the value of the given column.

GetValue

SYNTAX GetValue(string columnName)

ARGUMENTS

columnName	Column name

RETURN Column value

If the data type of the database column is DATE, DATETIME or any valid date type, the date read generally returns in the format MM/DD/YYYY hh:mm:ss.

EXAMPLE

```
GetObject("DateVar","MLDate")
…
StringDateVar = CurrentEnvelope.GetValue("ODE_CREATE_AT")
;Convert date to the required format with the help of the MLDate object
IntDateVar = DateVar.DateStringToInt64("MM/dd/yyyy hh:mm:ss", StringDateVar )
v_LETTERDATE= DateVar.Int64ToDateString("MM.dd.yyyy hh:mm:ss", IntDateVar)
```

GetFieldNameEnumerator

Returns an enumerator.

GetFieldNameEnumerator

SYNTAX GetFieldNameEnumerator()

RETURN Enumerator with the following methods.

Method	Description
Current	Returns the current object.
MoveNext	Moves the position to the next object to be processed.
Reset	Resets the enumerator to the beginning of the first element.

CurrentStack

Current stack.

Read-only property.

Functions

GetValue

Returns the value of the given column.

GetValue

SYNTAX GetValue(string columnName)

ARGUMENTS

columnName	Column name

RETURN Column value

If the data type of the database column is DATE, DATETIME or any valid date type, the date read generally returns in the format MM/DD/YYYY hh:mm:ss.

EXAMPLE	GetObject("DateVar","MLDate") … StringDateVar = CurrentEnvelope.GetValue("ODE_CREATE_AT") ;Convert date to the required format with the help of the MLDate object IntDateVar = DateVar.DateStringToInt64("MM/dd/yyyy hh:mm:ss", StringDateVar) v_LETTERDATE= DateVar.Int64ToDateString("MM.dd.yyyy hh:mm:ss", IntDateVar)

GetFieldNameEnumerator

Returns an enumerator.

GetFieldNameEnumerator

SYNTAX	GetFieldNameEnumerator()
RETURN	Enumerator with the following methods.

Method	Description
Current	Returns the current object.
MoveNext	Moves the position to the next object to be processed.
Reset	Resets the enumerator to the beginning of the first element.

Functions of the Context OdinArchive

HasData

Checks whether data exists for the given type.

HasData

SYNTAX	HasData(OdinDataRowType dataType)
ARGUMENTS	dataType — **Possible values** OdinDataRowType.Process OdinDataRowType.Stack OdinDataRowType.Envelope OdinDataRowType.Job OdinDataRowType.VirtualJob
RETURN	TRUE = The data row specified exists. FALSE = The data row specified does not exist.

Context OdinConvert

About the Context OdinConvert

Use this context in conversion scripts.

Properties of the Context OdinConvert

Show All

Property	Description
ConvertDocument	TRUE = Convert and add the document to the output file. FALSE = Do not convert and add the document to the output file. ConvertDocument is only evaluated after the **OnDocumentConvert** event.
CurrentDocument	Current document, if found. Read-only property. **Functions** GetDocumentFileName Returns the name of the original document.

GetDocumentFileName

SYNTAX GetDocumentFileName()

RETURN File name

GetDuplex

Returns the duplex mode for this document.

GetDuplex

SYNTAX GetDuplex()

RETURN Enumerator with the following functions:

Method	Description
OdinDuplexSetting.Simplex	No duplex
OdinDuplexSetting.Horizontal	Duplex on the short edge
OdinDuplexSetting.Vertical	Duplex on the long edge

GetFileName

Returns the rendition file name of the document.

GetFileName

SYNTAX GetFileName()

RETURN Rendition file name

GetFirstInputBin

Returns the tray for the first page of the document.

GetFirstInputBin

SYNTAX	GetFirstInputBin()
RETURN	Name of the tray.

GetId

Returns the ID of the document. For example, job ID, StackCoverPage or EnvelopeCoverPage.

GetId	
SYNTAX	GetId()
RETURN	Document Id

GetNextInputBin

Returns the follow on tray for the 2-n pages of a document.

GetNextInputBin	
SYNTAX	GetNextInputBin()
RETURN	NextInputBin

GetPageCount

Returns the number of pages.

GetPageCount	
SYNTAX	GetPageCount()
RETURN	Number of pages.

IsPortrait

Returns the portrait mode.

IsPortrait

SYNTAX	IsPortrait(int pageNumber)
ARGUMENTS	pageNumber Page number
RETURN	TRUE = Portrait FALSE = Landscape

SetDuplex

Specifies the duplex mode for this document.

SetDuplex	
SYNTAX	SetDuplex(duplexMode)
ARGUMENTS	duplexMode Duplex mode. **Possible values** OdinDuplexSetting.Simplex = No duplex OdinDuplexSetting.Horizontal = Duplex on the short edge OdinDuplexSetting.Vertical = Duplex on the long edge

SetFirstInputBin

Sets the tray for the first page of the document.

SetFirstInputBin	
SYNTAX	SetFirstInputBin(string inputBin)
ARGUMENTS	inputBin Tray

SetNextInputBin

Set the follow on tray for the 2-n pages of a document.

	SetNextInputBin SYNTAX: SetNextInputBin(string inputBin) ARGUMENTS: inputBin — Tray
HasDocument	TRUE = A document is available. FALSE = No document is available. Read-only property.
Current Envelope	Current envelope, if found. Read-only property.

Functions

GetValue

Returns the value of the given column.

	GetValue
SYNTAX	GetValue(string columnName)
ARGUMENTS	columnName — Column name
RETURN	Column value If the data type of the database column is DATE, DATETIME or any valid date type, the date read generally returns in the format MM/DD/YYYY hh:mm:ss.
EXAMPLE	GetObject("DateVar","MLDate") … StringDateVar = CurrentEnvelope.GetValue("ODE_CREATE_AT") ;Convert date to the required format with the help of the MLDate object IntDateVar = DateVar.DateStringToInt64("MM/dd/yyyy hh:mm:ss",

```
StringDateVar )
v_LETTERDATE= DateVar.Int64ToDateString("MM.dd.yyyy hh:mm:ss ",
IntDateVar)
```

GetFieldNameEnumerator

Returns an enumerator.

GetFieldNameEnumerator

SYNTAX GetFieldNameEnumerator()

RETURN Enumerator with the following methods.

Method	Description
Current	Returns the current object.
MoveNext	Moves the position to the next object to be processed.
Reset	Resets the enumerator to the beginning of the first element.

Current Job

Current job, if found.

Read-only property.

Functions

GetValue

Returns the value of the given column.

GetValue

SYNTAX GetValue(string columnName)

ARGUMENTS

columnName	Column name

RETURN	Column value If the data type of the database column is DATE, DATETIME or any valid date type, the date read generally returns in the format MM/DD/YYYY hh:mm:ss.
EXAMPLE	GetObject("DateVar","MLDate") … StringDateVar = CurrentEnvelope.GetValue("ODE_CREATE_AT") ;Convert date to the required format with the help of the MLDate object IntDateVar = DateVar.DateStringToInt64("MM/dd/yyyy hh:mm:ss", StringDateVar) v_LETTERDATE= DateVar.Int64ToDateString("MM.dd.yyyy hh:mm:ss ", IntDateVar)

GetFieldNameEnumerator

Returns an enumerator.

GetFieldNameEnumerator

SYNTAX	GetFieldNameEnumerator()
RETURN	Enumerator with the following methods.

Method	Description
Current	Returns the current object.
MoveNext	Moves the position to the next object to be processed.
Reset	Resets the enumerator to the beginning of the first element.

Current Stack

Current stack.

Read-only property.

Functions

GetValue

Returns the value of the given column.

GetValue

SYNTAX GetValue(string columnName)

ARGUMENTS

columnName	Column name

RETURN Column value

If the data type of the database column is DATE, DATETIME or any valid date type, the date read generally returns in the format MM/DD/YYYY hh:mm:ss.

EXAMPLE

```
GetObject("DateVar","MLDate")
…
StringDateVar = CurrentEnvelope.GetValue("ODE_CREATE_AT")
;Convert date to the required format with the help of the MLDate object
IntDateVar = DateVar.DateStringToInt64("MM/dd/yyyy hh:mm:ss", StringDateVar )
v_LETTERDATE= DateVar.Int64ToDateString("MM.dd.yyyy hh:mm:ss ", IntDateVar)
```

GetFieldNameEnumerator

Returns an enumerator.

GetFieldNameEnumerator

SYNTAX GetFieldNameEnumerator()

RETURN Enumerator with the following methods.

Method	Description

Current	Returns the current object.
MoveNext	Moves the position to the next object to be processed.
Reset	Resets the enumerator to the beginning of the first element.

Functions of the Context OdinConvert

HasData

Checks whether data exists for the given type.

HasData

SYNTAX HasData(OdinDataRowType dataType)

ARGUMENTS

dataType	**Possible values**
	OdinDataRowType.Process
	OdinDataRowType.Stack
	OdinDataRowType.Envelope
	OdinDataRowType.Job
	OdinDataRowType.VirtualJob

RETURN TRUE = The data row specified exists.

FALSE = The data row specified does not exist.

Context OdinCustomProcess

About the Context OdinCustomProcess

Use this context in stack processing scripts.

Properties of the Context OdinCustomProcess

Show All

Parameter	Description
CurrentEnvelope	Current envelope, if found. Read-only property. **Functions** GetValue Returns the value of the given column.

GetValue

SYNTAX	GetValue(string columnName)
ARGUMENTS	columnName Column name
RETURN	Column value If the data type of the database column is DATE, DATETIME or any valid date type, the date read generally returns in the format MM/DD/YYYY hh:mm:ss.
EXAMPLE	GetObject("DateVar","MLDate") … StringDateVar = CurrentEnvelope.GetValue("ODE_CREATE_AT") ;Convert date to the required format with the help of the MLDate object IntDateVar = DateVar.DateStringToInt64("MM/dd/yyyy hh:mm:ss", StringDateVar) v_LETTERDATE= DateVar.Int64ToDateString("MM.dd.yyyy hh:mm:ss", IntDateVar)

GetFieldNameEnumerator

Returns an enumerator.

GetFieldNameEnumerator

SYNTAX GetFieldNameEnumerator()

RETURN Enumerator with the following methods.

Method	Description
Current	Returns the current object.
MoveNext	Moves the position to the next object to be processed.
Reset	Resets the enumerator to the beginning of the first element.

CurrentJob

Current job, if found.

Read-only property.

Functions

GetValue

Returns the value of the given column.

GetValue

SYNTAX GetValue(string columnName)

ARGUMENTS

columnName	Column name

RETURN Column value

If the data type of the database column is DATE, DATETIME or any valid date type, the date read generally returns in the format MM/DD/YYYY hh:mm:ss.

EXAMPLE

```
GetObject("DateVar","MLDate")
…
StringDateVar = CurrentEnvelope.GetValue("ODE_CREATE_AT")
```

```
;Convert date to the required format with the help of the MLDate object
IntDateVar = DateVar.DateStringToInt64("MM/dd/yyyy hh:mm:ss", StringDateVar )
v_LETTERDATE= DateVar.Int64ToDateString("MM.dd.yyyy hh:mm:ss", IntDateVar)
```

GetFieldNameEnumerator

Returns an enumerator.

GetFieldNameEnumerator

SYNTAX GetFieldNameEnumerator()

RETURN Enumerator with the following methods.

Method	Description
Current	Returns the current object.
MoveNext	Moves the position to the next object to be processed.
Reset	Resets the enumerator to the beginning of the first element.

CurrentStack

Current stack.

Read-only property.

Functions

GetValue

Returns the value of the given column.

GetValue

SYNTAX GetValue(string columnName)

ARGUMENTS	columnName Column name
RETURN	Column value If the data type of the database column is DATE, DATETIME or any valid date type, the date read generally returns in the format MM/DD/YYYY hh:mm:ss.
EXAMPLE	GetObject("DateVar","MLDate") … StringDateVar = CurrentEnvelope.GetValue("ODE_CREATE_AT") ;Convert date to the required format with the help of the MLDate object IntDateVar = DateVar.DateStringToInt64("MM/dd/yyyy hh:mm:ss", StringDateVar) v_LETTERDATE= DateVar.Int64ToDateString("MM.dd.yyyy hh:mm:ss", IntDateVar)

GetFieldNameEnumerator

Returns an enumerator.

GetFieldNameEnumerator

SYNTAX	GetFieldNameEnumerator()
RETURN	Enumerator with the following methods.

Method	Description
Current	Returns the current object.
MoveNext	Moves the position to the next object to be processed.
Reset	Resets the enumerator to the beginning of the first element.

Functions of the Context OdinCustomProcess

HasData

Checks whether data exists for the given type.

HasData	
SYNTAX	HasData(OdinDataRowType dataType)
ARGUMENTS	dataType — **Possible values** OdinDataRowType.Process OdinDataRowType.Stack OdinDataRowType.Envelope OdinDataRowType.Job OdinDataRowType.VirtualJob
RETURN	TRUE = The data row specified exists. FALSE = The data row specified does not exist.

Context OdinDbTask

About the Context OdinDbTask

Use this context to access CurrentStack / CurrentEnvelope / CurrentJob data in OnDeleteJob / OnDeleteStack events of the OdinDbTasks component.

Properties of the Context OdinDbTask

Show All

Object	Description
CurrentEnvelope	Current envelope, if found. Read-only property. **Functions** GetValue

Returns the value of the given column.

GetValue

SYNTAX GetValue(string columnName)

ARGUMENTS

columnName	Column name

RETURN Column value

If the data type of the database column is DATE, DATETIME or any valid date type, the date read generally returns in the format MM/DD/YYYY hh:mm:ss.

EXAMPLE

```
GetObject("DateVar","MLDate")
…
StringDateVar = CurrentEnvelope.GetValue("ODE_CREATE_AT")
;Convert date to the required format with the help of the MLDate object
IntDateVar = DateVar.DateStringToInt64("MM/dd/yyyy hh:mm:ss", StringDateVar )
v_LETTERDATE= DateVar.Int64ToDateString("MM.dd.yyyy hh:mm:ss", IntDateVar)
```

GetFieldNameEnumerator

Returns an enumerator.

GetFieldNameEnumerator

SYNTAX GetFieldNameEnumerator()

RETURN Enumerator with the following methods.

Method	Description

Current	Returns the current object.
MoveNext	Moves the position to the next object to be processed.
Reset	Resets the enumerator to the beginning of the first element.

CurrentJob

Current job, if found.

Read-only property.

Functions

GetValue

Returns the value of the given column.

GetValue

SYNTAX GetValue(string columnName)

ARGUMENTS

columnName	Column name

RETURN Column value

If the data type of the database column is DATE, DATETIME or any valid date type, the date read generally returns in the format MM/DD/YYYY hh:mm:ss.

EXAMPLE

```
GetObject("DateVar","MLDate")
…
StringDateVar = CurrentEnvelope.GetValue("ODE_CREATE_AT")
;Convert date to the required format with the help of the MLDate object
IntDateVar = DateVar.DateStringToInt64("MM/dd/yyyy hh:mm:ss", StringDateVar )
v_LETTERDATE= DateVar.Int64ToDateString("MM.dd.yyyy hh:mm:ss", IntDateVar)
```

GetFieldNameEnumerator

Returns an enumerator.

GetFieldNameEnumerator

SYNTAX GetFieldNameEnumerator()

RETURN Enumerator with the following methods.

Method	Description
Current	Returns the current object.
MoveNext	Moves the position to the next object to be processed.
Reset	Resets the enumerator to the beginning of the first element.

CurrentStack

Current stack.

Read-only property.

Functions

GetValue

Returns the value of the given column.

GetValue

SYNTAX GetValue(string columnName)

ARGUMENTS

columnName	Column name

RETURN Column value

If the data type of the database column is DATE, DATETIME or any valid date type, the date read generally returns in the format MM/DD/YYYY hh:mm:ss.

EXAMPLE

```
GetObject("DateVar","MLDate")
…
StringDateVar = CurrentEnvelope.GetValue("ODE_CREATE_AT")
```

```
;Convert date to the required format with the help of the MLDate object
IntDateVar = DateVar.DateStringToInt64("MM/dd/yyyy hh:mm:ss", StringDateVar )
v_LETTERDATE= DateVar.Int64ToDateString("MM.dd.yyyy hh:mm:ss", IntDateVar)
```

GetFieldNameEnumerator

Returns an enumerator.

GetFieldNameEnumerator

SYNTAX GetFieldNameEnumerator()

RETURN Enumerator with the following methods.

Method	Description
Current	Returns the current object.
MoveNext	Moves the position to the next object to be processed.
Reset	Resets the enumerator to the beginning of the first element.

Functions of the Context OdinDbTask

HasData

Checks whether data exists for the given type.

HasData

SYNTAX HasData(OdinDataRowType dataType)

ARGUMENTS

dataType	**Possible values**

	OdinDataRowType.Process OdinDataRowType.Stack OdinDataRowType.Envelope OdinDataRowType.Job OdinDataRowType.VirtualJob
RETURN	TRUE = The data row specified exists. FALSE = The data row specified does not exist.

Context OdinEmail

About the Context OdinEmail

Use this context in scripts that send emails. The context provides scripts with the **Email** object that you can use to define the properties of the email.

The email properties of the SMTP host, already defined in the email printer, automatically set for the **Email** object.

The recipient (**MailTo**) is retrieved from the field **ODJ_EMAIL_TO** of the Odin job. The email printer automatically calls the function **Send()**.

To send an email, you must set the following properties in the script.

- MailFrom
- MailTo
- MailSubject
- BodyText or BodyHtml

Properties of the Context OdinEmail

Property	Description
SmtpHost	Host on which the SMTP server runs.
SmtpPort	Port of the SMTP server.
SmtpUserName	User name for SMTP server authentication.

SmtpPassword	Encoded password to authenticate to the SMTP server. **Note** To encode the password, use the tool **Encoder.exe** in the installation directory.
SmtpDomain	Domain or computer where login information is verified.
UseDefaultCredentials	TRUE = Authenticate to SMTP server using the logged on user. FALSE = Authenticate to SMTP server using SmptUserName and SmtpPassword.
OdinSmtpDeliveryMethod	Specifies how emails will be sent.

Possible properties	Description
OdinSmtpDeliveryMethod.Network	Emails are sent through the configured SMTP server. The following properties require a value. • SmtpHost • SmtpPort • SmtpUserName • SmtpPassword
OdinSmtpDeliveryMethod.SpecifiedPickupDirectory	Emails are copied to the directory specified in the **SmtpPickupDirectory** property and must be sent by an external program. The following property requires a value. • SmtpPickupDirectory
OdinSmtpDeliveryMethod.PickupDirectoryFromIis	Emails are copied to the directory configured in the Internet Information Server (IIS).

OdinDeliveryNotificationOptions

Defines the notification options for Emails.

Possible properties	Description
OdinDeliveryNotificationOptions.Network	Emails are sent through the configured SMTP server.
OdinDeliveryNotificationOptions.None	No notification
OdinDeliveryNotificationOptions.OnSuccess	Notification on successful delivery
OdinDeliveryNotificationOptions.OnFailure	Notification on unsuccessful delivery
OdinDeliveryNotificationOptions.Delay	Notification on delayed delivery.
OdinDeliveryNotificationOptions.Never	No notification whatsoever

OdinMailPriority

Email priority

Possible properties	Description
OdinMailPriority.Normal	The email has normal priority.
OdinMailPriority.Low	The email has low priority.
OdinMailPriority.High	The email has high priority.

IsBodyHtml

Defines whether the email body text is in HTML format.

TRUE = BodyHtml is used as HTML text, BodyText as alternative plain text.

FALSE = Only the BodyText is used as plain text.

MailEncoding

You can use the following values as encoding parameters.

- Empty string. In this case, the default encoding (ANSI code page of the operating system) is used.

- Code page, for example, "1252"
- Name, for example, "windows-1252"
- Friendly name, for example, "Western European" (Windows)

Possible encoding parameters

Code page	name	Display name
37	IBM037	IBM EBCDIC (USA-Canada)
437	IBM437	OEM USA
500	IBM500	IBM EBCDIC (International)
708	ASMO-708	Arabic (ASMO 708)
720	DOS-720	Arabic (DOS)
737	ibm737	Greek (DOS)
775	ibm775	Baltic (DOS)
850	ibm850	Western European (DOS)
852	ibm852	Eastern European (DOS)
855	IBM855	OEM Cyrillic
857	ibm857	Turkish (DOS)
858	IBM00858	OEM Multilingual Latin I
860	IBM860	Portuguese (DOS)
861	ibm861	Icelandic (DOS)
862	DOS-862	Hebrew (DOS)
863	IBM863	French, Canada (DOS)
864	IBM864	Arabic (864)
865	IBM865	Nordic (DOS)
866	cp866	Cyrillic (DOS)
869	ibm869	Greek, modern (DOS)
870	IBM870	IBM EBCDIC (Multilingual Latin-2)
874	windows-874	Thai (Windows)
875	cp875	IBM EBCDIC (Greek, modern)
932	shift_jis	Japanese (Shift-JIS)
936	gb2312	Simplified Chinese (GB2312)
949	ks_c_5601-1987	Korean
950	big5	Chinese (Traditional) (Big5)

1026	IBM1026	IBM EBCDIC (Turkish, Latin-5)
1047	IBM01047	IBM Latin-1
1140	IBM01140	IBM EBCDIC (USA-Canada-European)
1141	IBM01141	IBM EBCDIC (Germany-European)
1142	IBM01142	IBM EBCDIC (Denmark-Norway-European)
1143	IBM01143	IBM EBCDIC (Finland-Sweden-European)
1144	IBM01144	IBM EBCDIC (Italian-European)
1145	IBM01145	IBM EBCDIC (Spain-European)
1146	IBM01146	IBM EBCDIC (Great Britain-European)
1147	IBM01147	IBM EBCDIC (France-European)
1148	IBM01148	IBM EBCDIC (International-European)
1149	IBM01149	IBM EBCDIC (Icelandic-European)
1200	utf-16	Unicode
1201	unicodeFFFE	Unicode (Big Endian)
1250	windows-1250	Central European (Windows)
1251	windows-1251	Cyrillic (Windows)
1252	Windows-1252	Western European (Windows)
1253	windows-1253	Greek (Windows)
1254	windows-1254	Turkish (Windows)
1255	windows-1255	Hebrew (Windows)
1256	windows-1256	Arabic (Windows)
1257	windows-1257	Baltic (Windows)
1258	windows-1258	Vietnamese (Windows)
1361	Johab	Korean (Johab)
10000	macintosh	Western European (Mac)
10001	x-mac-japanese	Japanese (Mac)
10002	x-mac-chinesetrad	Chinese traditional (Mac)
10003	x-mac-korean	Korean (Mac)
10004	x mac-arabic	Arabic (Mac)
10005	x-mac-hebrew	Hebrew (Mac)

10006	x-mac-greek	Greek (Mac)
10007	x-mac-cyrillic	Cyrillic (Mac)
10008	x-mac-chinesesimp	Chinese simplified (Mac)
10010	x-mac-romanian	Romanian (Mac)
10017	x-mac-ukrainian	Ukrainian (Mac)
10021	x-mac-thai	Thai (Mac)
10029	x-mac-ce	Central European (Mac)
10079	x-mac-icelandic	Icelandic (Mac)
10081	x-mac-turkish	Turkish (Mac)
10082	x-mac-croatian	Croatian (Mac)
12000	utf-32	Unicode (UTF-32)
12001	utf-32BE	Unicode (UTF-32-Big Endian)
20000	x-Chinese-CNS	Chinese traditional (CNS)
20001	x-cp20001	TCA Taiwan
20002	x-Chinese-Eten	Chinese traditional (Eten)
20003	x-cp20003	IBM5550 Taiwan
20004	x-cp20004	TeleText Taiwan
20005	x-cp20005	Wang Taiwan
20105	x-IA5	Western European (IA5)
20106	x-IA5-German	German (IA5)
20107	x-IA5-Swedish	Swedish (IA5)
20108	x-IA5-Norwegian	Norwegian (IA5)
20127	us-ascii	US-ASCII
20261	x-cp20261	T.61
20269	x-cp20269	ISO-6937
20273	IBM273	IBM EBCDIC (Germany)
20277	IBM277	IBM EBCDIC (Denmark-Norway)
20278	IBM278	IBM EBCDIC (Finland-Sweden)
20280	IBM280	IBM EBCDIC (Italy)
20284	IBM284	IBM EBCDIC (Spain)
20285	IBM285	IBM EBCDIC (Great Britain)
20290	IBM290	IBM EBCDIC (Japanese Katakana)

20297	IBM297	IBM EBCDIC (France)
20420	IBM420	IBM EBCDIC (Arabic)
20423	IBM423	IBM EBCDIC (Greek)
20424	IBM424	IBM EBCDIC (Hebrew)
20833	x-EBCDIC-KoreanExtended	IBM EBCDIC (Korean, extended)
20838	IBM-Thai	IBM EBCDIC (Thai)
20866	koi8-r	Cyrillic (KOI8-R)
20871	IBM871	IBM EBCDIC (Icelandic)
20880	IBM880	IBM EBCDIC (Cyrillic, Russian)
20905	IBM905	IBM EBCDIC (Turkish)
20924	IBM00924	IBM Latin-1
20932	EUC-JP	Japanese (JIS 0208-1990 and 0212-1990)
20936	x-cp20936	GB2312-80 Chinese (simple)
20949	x-cp20949	Korean Wansung
21025	cp1025	IBM EBCDIC (Cyrillic, Serbian-Bulgarian)
21866	koi8-u	Cyrillic (KOI8-U)
28591	iso-8859-1	Western European (ISO)
28592	iso-8859-2	Central European (ISO)
28593	iso-8859-3	Latin 3 (ISO)
28594	iso-8859-4	Baltic (ISO)
28595	iso-8859-5	Cyrillic (ISO)
28596	iso-8859-6	Arabic (ISO)
28597	iso-8859-7	Greek (ISO)
28598	iso-8859-8	Hebrew (ISO-Visual)
28599	iso-8859-9	Turkish (ISO)
28603	iso-8859-13	Estonian (ISO)
28605	iso-8859-15	Latin 9 (ISO)
29001	x-Europa	Europe
38598	iso-8859-8-i	Hebrew (ISO-Logical)
50220	iso-2022-jp	Japanese (JIS)

50221	csISO2022JP	Japanese (JIS, 1 Byte Kana allowed)
50222	iso-2022-jp	Japanese (JIS, 1 Byte Kana allowed - SO/SI)
50225	iso-2022-kr	Korean (ISO)
50227	x-cp50227	ISO-2022 Chinese (simplified)
51932	euc-jp	Japanese (EUC)
51936	EUC-CN	Chinese simplified (EUC)
51949	euc-kr	Korean (EUC)
52936	hz-gb-2312	Chinese simplified (HZ)
54936	GB18030	GB18030 Chines simplified
57002	x-iscii-de	ISCII Devanagari
57003	x-iscii-be	ISCII Bengali
57004	x-iscii-ta	ISCII Tamil
57005	x-iscii-te	ISCII Telugu
57006	x-iscii as	ISCII Assamese
57007	x-iscii-or	ISCII Oriya
57008	x-iscii-ka	ISCII Kannada
57009	x-iscii-ma	ISCII Malayalam
57010	x-iscii-gu	ISCII Gujarati
57011	x-iscii-pa	ISCII Punjabi
65000	utf-7	Unicode (UTF-7)
65001	utf-8	Unicode (UTF-8)

MailEncodingCodepage	Encoding or code page for the email.
MailSubject	Email Subject
BodyText	Sets or gets the email text in plain text.
BodyHtml	Sets or gets the email text in HTML format.
BodyHtmlImages	Returns or sets the comma separated string of BodyHtmlImages.
SmtpPickupDirectory	Returns or sets the directory to which emails are saved, if SmtpDeliveryMethod.SpecifiedPickupDirectory is set.

MailFrom	Returns or sets the email sender.
MailTo	Returns or sets the email CC recipient (comma separated).
MailCC	Returns or sets the email CC recipient (comma separated).
MailBcc	Returns or sets the email Bcc recipient (comma separated).
MailReplyTo	Returns or sets the ReplyTo address for email.
MailSender	Specifies the originating email address used for the SMTP "MAIL FROM" command. This is also the return or bounce address in case the message is not delivered.
AttachmentCount	Returns the number of email attachments.

Notes

- Specify email addresses in the following format.
  `"DisplayName" <email@domain>`
- Specify comma separated email addresses in the following format.
  - "DisplayName" <email@domain>,<email@domain>,email@domain

Functions of the Context OdinEmail

Example

"displayName-1" <email-1@domain-1.de>,"displayName-N

AddAttachment

Adds an attachment to an email.

AddAttachment		
SYNTAX	AddAttachment(string fileName)	
ARGUMENTS	filename	Name of the attachment file.
EXAMPLE	email.AddAttachment("c:\temp\test.pdf")	

AddBodyHtmlImage

Adds an image to the HTML body.

AddBodyHtmlImage

SYNTAX AddBodyHtmlImage(string id, string imageFileName)

ARGUMENTS

id	ID of the image (cid). If no ID is specified, the file name without its extension is used.
imageFileName	File name of the image.

EXAMPLE email.AddBodyHtmlImage("", "c:\temp\no-fear.jpg")

AddToHeader

Adds a name and value entry to an email header.

AddToHeader

SYNTAX AddToHeader(string name, string value)

ARGUMENTS

name	Name of the entry. If the entry already exists, its value is overwritten. Otherwise, a new entry is created.
value	Entry value.

EXAMPLE

```
Email.AddToHeader("Organization", "Hyland Software")
```

ClearAttachments

Deletes all loaded attachments.

ClearAttachments

SYNTAX ClearAttachments()

EXAMPLE ClearAttachments()

ClearBodyHtmlImages

Deletes all images loaded in the HTML body.

ClearBodyHtmlImages	
SYNTAX	ClearBodyHtmlImages()
EXAMPLE	ClearBodyHtmlImages()

ClearCertificates

Deletes all loaded or defined certificates.

ClearCertificates	
SYNTAX	ClearCertificates()

GetAttachments

Returns a comma separated string containing the file names of all attachments.

GetAttachments	
SYNTAX	GetAttachments()

LoadBodyFromFile

Loads a body text from a file.

LoadBodyFromFile		
SYNTAX	LoadBodyFromFile(fileName, isHtml)	
ARGUMENTS	fileName	File name,
	isHtml	TRUE = Is HTML FALSE = Is not HTML

LoadCertificateFromCertFile

Loads the X.509 certificate from the specified file to send the email.

LoadCertificateFromCertFile		
SYNTAX	LoadCertificateFromCertFile(fileName)	
ARGUMENTS	fileName	File name.

LoadCertificateFromSignedFile

Loads the X.509 certificate from the specified file to send the email.

LoadCertifacteFromSignedFile		
SYNTAX	LoadCertifacteFromSignedFile(fileName)	
ARGUMENTS	fileName	File name.

RemoveAttachment

Removes an attachment from an email.

RemoveAttachment		
SYNTAX	RemoveAttachment(fileName)	
ARGUMENTS	fileName	Name of the file to remove from the attachments.

RemoveFromHeader

Removes an entry from an email header.

RemoveFromHeader

SYNTAX	RemoveFromHeader(name)
ARGUMENTS	name — Name of the entry to remove.
EXAMPLE	Email.RemoveFromHeader("Organisation")

Send

Send the email with the current settings.

Send	
SYNTAX	Send()

SetCredentials

Sets the user credentials user name, password, and domain.

SetCredentials	
SYNTAX	SetCredentials(userName, password, domain)
ARGUMENTS	userName — User name password — Encoded password **Note** To encode the password, use the tool Encoder.exe in the installation directory. domain — Domain
RETURN	TRUE = Attribute value is set. FALSE = Attribute value could not be set.

SetSmtpServer

Sets the SMTP server properties host, port, and useDefaultCredentials.

SetSmtpServer

SYNTAX SetSmtpServer(host, port, useDefaultCredentials)

ARGUMENTS

host	Host on which the SMTP server runs
port	Port of the SMTP server
useDefaultCredentials	TRUE = Authenticate to SMTP server using the logged on user. FALSE = Authenticate to SMTP server using SmptUserName and SmtpPassword.

Sample Script OdinEmail

```
; Define email settings such as sender or priority.
Email.MailFrom = """info"" <info@perceptivesoftware.com>"
Email.MailPriority = OdinMailPriority.High
;
; Was a document converted to body text?
if ((Email.BodyHtml = "") or (Email.BodyText = ""))
  ; No, then set the body text as plain text
  Email.BodyText = "this is the body text!!"
  Email.IsBodyHtml = false
;
end-if
;
; Was a document converted to the subject?
if (Email.MailSubject = "")
  ; No, so set a subject
  Email.MailSubject = "subject line"
end-if
```

Context OdinEmailConvert

Properties of the Context OdinEmailConvert

Show All

Property	Description
ConvertOptions	Comma separated string containing the metadata and options used during conversion. You can also set these options in the OnConvertStart event to use, for example, the stack data to define the metadata. Metadata

TITLE	Document title.
SUBJECT	Document subject.
AUTHOR	Document author.
KEYWORDS	Document keywords.

Options

TEXTNOTE_CHANGES	PDF 1.3/1.4 TRUE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `True`. FALSE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `False`.
CHANGES	PDF 1.3/1.4 TRUE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to

	`True`. FALSE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `False`.
FILL	PDF 1.4 TRUE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `True`. FALSE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `False`.
COPY	PDF 1.4 TRUE = PDF security attributes for Document Assembly, Copy Content, Copy Content for Accessibility, Page Extraction, Comment, Sign and Create Template are set to `True`. FALSE = PDF security attributes for Document Assembly, Copy Content, Copy Content for Accessibility, Page Extraction, Comment, Sign and Create Template are set to `False`.
DIGITAL_COPY	PDF 1.4 TRUE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `True`. FALSE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `False`.
ASSEMBLE	PDF 1.4 TRUE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `True`. FALSE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `False`. Assembling more than one document is not allowed.

ACCESSIBILITY	PDF 1.4 TRUE = PDF security attributes for Copy Content, Copy Content for Accessibility, Page Extraction are set to `True`. FALSE = PDF security attributes for Copy Content, Copy Content for Accessibility, Page Extraction are set to `False`.
DUPLEX_ ADDPAGE	TRUE = This option specifies whether an empty page should be added to a duplex document with an odd number of pages. FALSE = Default setting. This option sets simplex mode for the last page of a duplex document with an odd number of pages.
DUPLEX_ OPTIMIZE	TRUE = Duplex settings are only set in the output stream, when duplex mode changes. FALSE = Duplex settings are set in the output stream.
MAP_COLOR	TRUE/FALSE This parameter is used to define whether color mapping for the AFP-HighlightColor Definition is executed or not. The individual color definitions can be specified as MapColor_1 to MapColor_n. Example: Map_Color=TRUE,MapColor_1=-1,20,black,MapColor_ 2=0,20,red Note As MapColor parameters contain comma separated values themselves, the complete string must be enclosed in quotation marks. The structure is as follows: MapColor_???= HSB-Color,Threshold,ColorName HSB-Color=HSB Color Angle Threshold= Threshold value of the HSB color ColorName= Name of the color as defined in the AFP profile

	under HighlightColorList. The example above requires the following definitions in the AFP profile: </highlightcolorlist <highlightcolor name='red' type='OCA' number='0x0002' foreground-rgb="#00FF0000" /> <highlightcolor name='black' type='OCA' number='0x0008' foreground-rgb="#00000000" /> </highlightcolorlist> and <optimizepa value="never"/> <color value="FALSE" /> <usehighlightcolor value="TRUE" /> <enablebilevelimage value="TRUE" />
SHOW_ BOOKMARKS	TRUE = Bookmarks show in the opened PDF. FALSE = Bookmarks do not show in the opened PDF.
OVERLAYS	TRUE = OVERLAYS are created. FALSE = Default. OVERLAYS are not created.
MERGE_FONTS	TRUE: Default FALSE
MERGE_ SPLITWORDS	TRUE = Default, PCL output optimization is activated. FALSE = PCL output optimization is deactivated. This option should be set to FALSE if you have character spacing problems in a generated PCL.

	The default value is TRUE.
ROTATE_ LANDSCAPE	TRUE = If page orientation is landscape, the page rotates by 90°. FALSE = The page does not rotate. The default value is TRUE.

CurrentDocument

Returns the current document (IOdinStreamDocument interface) to be converted.

Functions of the IOdinStreamDocument-Interface

AddBookmark

EditPage

GetInputBin

Image

IsDuplex

IsPortrait

Line

PageCount

Post

Rectangle

SetFont

SetInputBin

TextOut

InputBinName

Returns the name of the tray currently used to convert or print the document.

InputBinType

Type of tray currently used to convert the document. Corresponds to the BinType property of the email tray.

Possible values	Description
OdinEmailPrinterBinType.Attachment	The document is converted to the

		corresponding output format, for example PDF, and then added to the email as a separate attachment.
	OdinEmailPrinterBinType.AttachmentCollect	All documents are converted to a single attachment file in the corresponding output format.
	OdinEmailPrinterBinType.Body	The document is converted to text or HTML format and used as the body text of the email.
	OdinEmailPrinterBinType.Subject	Enables a single line document to be converted to an individual reference text of the email. The document is converted to text format and used as the subject text.
OutputFileName	Returns or sets the name of the attachment. The default file name of the converted document is the job ID with an extension corresponding to the output format defined for the tray. It is possible to alter the name, but any path specifications and extensions are ignored.	

Functions of the Context OdinEmailConvert

SetPdfSigner

The OnConvert event of an email printer is called for every document to convert.

The SetPdfSigner() function of the OdinEmailConvert context can pass an initialized PDF signer object that can be used to sign a document when the output format is PDF.

If a password is set in the ConvertOptions, the PDF signer object must be initialized with the password. Otherwise, access to the PDF is not allowed.

SetPdfSigner	
SYNTAX	SetPdfSigner(pdfSigner)

ARGUMENTS	pdfSigner	The initialized PDF-signer object created by the OdinFileConvert component. For more information, refer to CreatePdfSigner.

Functions of IOdinStreamDocument Interface

AddBookmark

Adds a bookmark to a document. If the output format is PDF, a bookmark is created for every bookmark defined. The IDs must be unique for each bookmark.

AddBookmark

SYNTAX

AddBookmark(string id, string parentId, string title)

AddBookmark(string id, string parentId, string title, double x, double y)

ARGUMENTS

id	Bookmark ID
parentId	An empty string here means the bookmark is inserted at the highest level. Otherwise, the bookmark is inserted as a child of the bookmark specified.
title	Text for the bookmark.
x	X-position on the page.
y	Y-position on the page.

EXAMPLE

```
document.AddBookmark("doc1_page1", "doc1", "Page 1")
document.AddBookmark("doc1_page2", "doc1", "Page 2", 0, 0)
```

EditPage

EditPage sets the given page to edit mode so that character commands can be executed on the page.

EditPage

SYNTAX EditPage(pageNumber)

ARGUMENTS

pageNumber	Number of the page to be processed, beginning with 0.

GetInputBin

Returns the input bin of a page. If the input bin is not set, it returns the tray for the first page or for the subsequent tray, depending on the page number.

GetInputBin

SYNTAX GetInputBin(pageNumber)

ARGUMENTS

pageNumber	Page number.

RETURN Input bin.

Image

Inserts an image file.

Image

SYNTAX Image(x, y, fileName)

Image(x, y, imageFileName, layer)

ARGUMENTS

x	X-position.
y	Y-position.
filename	File name.
layer	**Possible Values**

	OdinPageLayer.Foreground = Drawing is done on top of the original document. OdinPageLayer.Background = Drawing is done behind the original document.
RETURN	TRUE = The user has the corresponding object rights. FALSE = The user does not have the corresponding object rights.

IsDuplex

Returns the duplex mode.

IsDuplex	
SYNTAX	IsDuplex()
RETURN	TRUE = Duplex FALSE = Simplex

IsPortrait

Returns the portrait mode.

IsPortrait		
SYNTAX	IsPortrait(pageNumber)	
ARGUMENTS	pageNumber	Page number.
RETURN	TRUE = Portrait FALSE = Landscape	

Line

Draws a line on the current page.

Line

SYNTAX

Line(x1, y1, x2, y2)

Line(x1, y1, x2, y2, lineWidth, lineStyle, color)

Line(x1, y1, x2, y2, lineWidth, lineStyle, color, layer)

ARGUMENTS

x1	X-position of the starting position of the line.
y1	Y-position of the starting position of the line.
x2	X-position of the end position of the line.
y2	Y-position of the end position of the line.
lineWidth	Line width.
lineStyle	**Possible values** OdinLineStyle.Solid OdinLineStyle.Dashed OdinLineStyle.Dotted
color	Color **Possible value** The HTML color name or the ARGB value in format #AARRGGBB (AA=Alpha-Transparency, RR=Red, GG=Green, BB=Blue).

HTML color name	ARGB value
AliceBlue	#FFF0F8FF
AntiqueWhite	#FFFAEBD7
Aqua	#FF00FFFF

Aquamarine	#FF7FFFD4
Azure	#FFF0FFFF
Beige	#FFF5F5DC
Bisque	#FFFFE4C4
Black	#FF000000
BlanchedAlmond	#FFFFEBCD
Blue	#FF0000FF
BlueViolet	#FF8A2BE2
Brown	#FFA52A2A
BurlyWood	#FFDEB887
CadetBlue	#FF5F9EA0
Chartreuse	#FF7FFF00
Chocolate	#FFD2691E
Coral	#FFFF7F50
CornflowerBlue	#FF6495ED
Cornsilk	#FFFFF8DC
Crimson	#FFDC143C
Cyan	#FF00FFFF
DarkBlue	#FF00008B
DarkCyan	#FF008B8B

DarkGoldenrod	#FFB8860B
DarkGray	#FFA9A9A9
DarkGreen	#FF006400
DarkKhaki	#FFBDB76B
DarkMagenta	#FF8B008B
DarkOliveGreen	#FF556B2F
DarkOrange	#FFFF8C00
DarkOrchid	#FF9932CC
DarkRed	#FF8B0000
DarkSalmon	#FFE9967A
DarkSeaGreen	#FF8FBC8B
DarkSlateBlue	#FF483D8B
DarkSlateGray	#FF2F4F4F
DarkTurquoise	#FF00CED1
DarkViolet	#FF9400D3
DeepPink	#FFFF1493
DeepSkyBlue	#FF00BFFF
DimGray	#FF696969
DodgerBlue	#FF1E90FF
Firebrick	#FFB22222

FloralWhite	#FFFFFAF0
ForestGreen	#FF228B22
Fuchsia	#FFFF00FF
Gainsboro	#FFDCDCDC
GhostWhite	#FFF8F8FF
Gold	#FFFFD700
Goldenrod	#FFDAA520
Gray	#FF808080
Green	#FF008000
GreenYellow	#FFADFF2F
Honeydew	#FFF0FFF0
HotPink	#FFFF69B4
IndianRed	#FFCD5C5C
Indigo	#FF4B0082
Ivory	#FFFFFFF0
Khaki	#FFF0E68C
Lavender	#FFE6E6FA
LavenderBlush	#FFFFF0F5
LawnGreen	#FF7CFC00
LemonChiffon	#FFFFFACD

LightBlue	#FFADD8E6
LightCoral	#FFF08080
LightCyan	#FFE0FFFF
LightGoldenrodYellow	#FFFAFAD2
LightGray	#FFD3D3D3
LightGreen	#FF90EE90
LightPink	#FFFFB6C1
LightSalmon	#FFFFA07A
LightSeaGreen	#FF20B2AA
LightSkyBlue	#FF87CEFA
LightSlateGray	#FF778899
LightSteelBlue	#FFB0C4DE
LightYellow	#FFFFFFE0
Lime	#FF00FF00
LimeGreen	#FF32CD32
Linen	#FFFAF0E6
Magenta	#FFFF00FF
Maroon	#FF800000
MediumAquamarine	#FF66CDAA
MediumBlue	#FF0000CD

MediumOrchid	#FFBA55D3
MediumPurple	#FF9370DB
MediumSeaGreen	#FF3CB371
MediumSlateBlue	#FF7B68EE
MediumSpringGreen	#FF00FA9A
MediumTurquoise	#FF48D1CC
MediumVioletRed	#FFC71585
MidnightBlue	#FF191970
MintCream	#FFF5FFFA
MistyRose	#FFFFE4E1
Moccasin	#FFFFE4B5
NavajoWhite	#FFFFDEAD
Navy	#FF000080
OldLace	#FFFDF5E6
Olive	#FF808000
OliveDrab	#FF6B8E23
Orange	#FFFFA500
OrangeRed	#FFFF4500
Orchid	#FFDA70D6
PaleGoldenrod	#FFEEE8AA

PaleGreen	#FF98FB98
PaleTurquoise	#FFAFEEEE
PaleVioletRed	#FFDB7093
PapayaWhip	#FFFFEFD5
PeachPuff	#FFFFDAB9
Peru	#FFCD853F
Pink	#FFFFC0CB
Plum	#FFB0E0E6
Purple	#FF800080
Red	#FFFF0000
RosyBrown	#FFBC8F8F
RoyalBlue	#FF4169E1
SaddleBrown	#FF8B4513
Salmon	#FFFA8072
SandyBrown	#FFF4A460
SeaGreen	#FF2E8B57
SeaShell	#FFFFF5EE
Sienna	#FFA0522D
Silver	#FFC0C0C0
SkyBlue	#FF87CEEB

SlateBlue	#FF6A5ACD
SlateGray	#FF708090
Snow	#FFFFFAFA
SpringGreen	#FF00FF7F
SteelBlue	#FF4682B4
Tan	#FFD2B48C
Teal	#FF008080
Thistle	#FFD8BFD8
Tomato	#FFFF6347
Turquoise	#FF40E0D0
Violet	#FFEE82EE
Wheat	#FFF5DEB3
White	#FFFFFFFF
WhiteSmoke	#FFF5F5F5
Yellow	#FFFFFF00
YellowGreen	#FF9ACD32

layer	**Possible values** OdinPageLayer.Foreground OdinPageLayer.Background

Pagecount

Returns the number of pages of the current document.

PageCount	
SYNTAX	PageCount()
RETURN	Number of pages

Post

Writes character commands in the document layer.

Post	
SYNTAX	Post()

Rectangle

Draws a rectangle on the current page.

Rectangle	
SYNTAX	Rectangle(x, y, witdh, height, fill, fillcolor) Rectangle(x, y, width, height, lineWidth, lineStyle, lineColor, fill, fillColor) Rectangle(x, y, width, height, lineWidth, lineStyle, lineColor, fill, fillColor, layer)
ARGUMENTS	x — X-position of the top left corner of the rectangle. y — Y-position of the top left corner of the rectangle. If the string list is empty, all fields are read. The string list must exist. width — Width of the rectangle. height — Height of the rectangle. fill — TRUE = The rectangle is filled.

FALSE = The rectangle is not filled.

The default value is FALSE.

fillColor

Pad color.

Possible value

The HTML color name or the ARGB value in format #AARRGGBB (AA=Alpha-Transparency, RR=Red, GG=Green, BB=Blue).

HTML color name	ARGB value
AliceBlue	#FFF0F8FF
AntiqueWhite	#FFFAEBD7
Aqua	#FF00FFFF
Aquamarine	#FF7FFFD4
Azure	#FFF0FFFF
Beige	#FFF5F5DC
Bisque	#FFFFE4C4
Black	#FF000000
BlanchedAlmond	#FFFFEBCD
Blue	#FF0000FF
BlueViolet	#FF8A2BE2
Brown	#FFA52A2A
BurlyWood	#FFDEB887
CadetBlue	#FF5F9EA0
Chartreuse	#FF7FFF00

Chocolate	#FFD2691E
Coral	#FFFF7F50
CornflowerBlue	#FF6495ED
Cornsilk	#FFFFF8DC
Crimson	#FFDC143C
Cyan	#FF00FFFF
DarkBlue	#FF00008B
DarkCyan	#FF008B8B
DarkGoldenrod	#FFB8860B
DarkGray	#FFA9A9A9
DarkGreen	#FF006400
DarkKhaki	#FFBDB76B
DarkMagenta	#FF8B008B
DarkOliveGreen	#FF556B2F
DarkOrange	#FFFF8C00
DarkOrchid	#FF9932CC
DarkRed	#FF8B0000
DarkSalmon	#FFE9967A
DarkSeaGreen	#FF8FBC8B
DarkSlateBlue	#FF483D8B

DarkSlateGray	#FF2F4F4F
DarkTurquoise	#FF00CED1
DarkViolet	#FF9400D3
DeepPink	#FFFF1493
DeepSkyBlue	#FF00BFFF
DimGray	#FF696969
DodgerBlue	#FF1E90FF
Firebrick	#FFB22222
FloralWhite	#FFFFFAF0
ForestGreen	#FF228B22
Fuchsia	#FFFF00FF
Gainsboro	#FFDCDCDC
GhostWhite	#FFF8F8FF
Gold	#FFFFD700
Goldenrod	#FFDAA520
Gray	#FF808080
Green	#FF008000
GreenYellow	#FFADFF2F
Honeydew	#FFF0FFF0
HotPink	#FFFF69B4

IndianRed	#FFCD5C5C
Indigo	#FF4B0082
Ivory	#FFFFFFF0
Khaki	#FFF0E68C
Lavender	#FFE6E6FA
LavenderBlush	#FFFFF0F5
LawnGreen	#FF7CFC00
LemonChiffon	#FFFFFACD
LightBlue	#FFADD8E6
LightCoral	#FFF08080
LightCyan	#FFE0FFFF
LightGoldenrodYellow	#FFFAFAD2
LightGray	#FFD3D3D3
LightGreen	#FF90EE90
LightPink	#FFFFB6C1
LightSalmon	#FFFFA07A
LightSeaGreen	#FF20B2AA
LightSkyBlue	#FF87CEFA
LightSlateGray	#FF778899
LightSteelBlue	#FFB0C4DE

LightYellow	#FFFFFFE0
Lime	#FF00FF00
LimeGreen	#FF32CD32
Linen	#FFFAF0E6
Magenta	#FFFF00FF
Maroon	#FF800000
MediumAquamarine	#FF66CDAA
MediumBlue	#FF0000CD
MediumOrchid	#FFBA55D3
MediumPurple	#FF9370DB
MediumSeaGreen	#FF3CB371
MediumSlateBlue	#FF7B68EE
MediumSpringGreen	#FF00FA9A
MediumTurquoise	#FF48D1CC
MediumVioletRed	#FFC71585
MidnightBlue	#FF191970
MintCream	#FFF5FFFA
MistyRose	#FFFFE4E1
Moccasin	#FFFFE4B5
NavajoWhite	#FFFFDEAD

Navy	#FF000080
OldLace	#FFFDF5E6
Olive	#FF808000
OliveDrab	#FF6B8E23
Orange	#FFFFA500
OrangeRed	#FFFF4500
Orchid	#FFDA70D6
PaleGoldenrod	#FFEEE8AA
PaleGreen	#FF98FB98
PaleTurquoise	#FFAFEEEE
PaleVioletRed	#FFDB7093
PapayaWhip	#FFFFEFD5
PeachPuff	#FFFFDAB9
Peru	#FFCD853F
Pink	#FFFFC0CB
Plum	#FFB0E0E6
Purple	#FF800080
Red	#FFFF0000
RosyBrown	#FFBC8F8F
RoyalBlue	#FF4169E1

SaddleBrown	#FF8B4513
Salmon	#FFFA8072
SandyBrown	#FFF4A460
SeaGreen	#FF2E8B57
SeaShell	#FFFFF5EE
Sienna	#FFA0522D
Silver	#FFC0C0C0
SkyBlue	#FF87CEEB
SlateBlue	#FF6A5ACD
SlateGray	#FF708090
Snow	#FFFFFAFA
SpringGreen	#FF00FF7F
SteelBlue	#FF4682B4
Tan	#FFD2B48C
Teal	#FF008080
Thistle	#FFD8BFD8
Tomato	#FFFF6347
Turquoise	#FF40E0D0
Violet	#FFEE82EE
Wheat	#FFF5DEB3

White	#FFFFFFFF
WhiteSmoke	#FFF5F5F5
Yellow	#FFFFFF00
YellowGreen	#FF9ACD32

lineWidth — Line width.

lineStyle — Line style.

Possible values

OdinLineStyle.Solid

OdinLineStyle.Dashed

OdinLineStyle.Dotted

lineColor — Line color.

Possible value is the HTML color name or the ARGB value in format #AARRGGBB (AA=Alpha-Transparency, RR=Red, GG=Green, BB=Blue).

HTML color name	ARGB value
AliceBlue	#FFF0F8FF
AntiqueWhite	#FFFAEBD7
Aqua	#FF00FFFF
Aquamarine	#FF7FFFD4
Azure	#FFF0FFFF

Beige	#FFF5F5DC
Bisque	#FFFFE4C4
Black	#FF000000
BlanchedAlmond	#FFFFEBCD
Blue	#FF0000FF
BlueViolet	#FF8A2BE2
Brown	#FFA52A2A
BurlyWood	#FFDEB887
CadetBlue	#FF5F9EA0
Chartreuse	#FF7FFF00
Chocolate	#FFD2691E
Coral	#FFFF7F50
CornflowerBlue	#FF6495ED
Cornsilk	#FFFFF8DC
Crimson	#FFDC143C
Cyan	#FF00FFFF
DarkBlue	#FF00008B
DarkCyan	#FF008B8B
DarkGoldenrod	#FFB8860B
DarkGray	#FFA9A9A9

DarkGreen	#FF006400
DarkKhaki	#FFBDB76B
DarkMagenta	#FF8B008B
DarkOliveGreen	#FF556B2F
DarkOrange	#FFFF8C00
DarkOrchid	#FF9932CC
DarkRed	#FF8B0000
DarkSalmon	#FFE9967A
DarkSeaGreen	#FF8FBC8B
DarkSlateBlue	#FF483D8B
DarkSlateGray	#FF2F4F4F
DarkTurquoise	#FF00CED1
DarkViolet	#FF9400D3
DeepPink	#FFFF1493
DeepSkyBlue	#FF00BFFF
DimGray	#FF696969
DodgerBlue	#FF1E90FF
Firebrick	#FFB22222
FloralWhite	#FFFFFAF0
ForestGreen	#FF228B22

Fuchsia	#FFFF00FF
Gainsboro	#FFDCDCDC
GhostWhite	#FFF8F8FF
Gold	#FFFFD700
Goldenrod	#FFDAA520
Gray	#FF808080
Green	#FF008000
GreenYellow	#FFADFF2F
Honeydew	#FFF0FFF0
HotPink	#FFFF69B4
IndianRed	#FFCD5C5C
Indigo	#FF4B0082
Ivory	#FFFFFFF0
Khaki	#FFF0E68C
Lavender	#FFE6E6FA
LavenderBlush	#FFFFF0F5
LawnGreen	#FF7CFC00
LemonChiffon	#FFFFFACD
LightBlue	#FFADD8E6
LightCoral	#FFF08080

LightCyan	#FFE0FFFF
LightGoldenrodYellow	#FFFAFAD2
LightGray	#FFD3D3D3
LightGreen	#FF90EE90
LightPink	#FFFFB6C1
LightSalmon	#FFFFA07A
LightSeaGreen	#FF20B2AA
LightSkyBlue	#FF87CEFA
LightSlateGray	#FF778899
LightSteelBlue	#FFB0C4DE
LightYellow	#FFFFFFE0
Lime	#FF00FF00
LimeGreen	#FF32CD32
Linen	#FFFAF0E6
Magenta	#FFFF00FF
Maroon	#FF800000
MediumAquamarine	#FF66CDAA
MediumBlue	#FF0000CD
MediumOrchid	#FFBA55D3
MediumPurple	#FF9370DB

MediumSeaGreen	#FF3CB371
MediumSlateBlue	#FF7B68EE
MediumSpringGreen	#FF00FA9A
MediumTurquoise	#FF48D1CC
MediumVioletRed	#FFC71585
MidnightBlue	#FF191970
MintCream	#FFF5FFFA
MistyRose	#FFFFE4E1
Moccasin	#FFFFE4B5
NavajoWhite	#FFFFDEAD
Navy	#FF000080
OldLace	#FFFDF5E6
Olive	#FF808000
OliveDrab	#FF6B8E23
Orange	#FFFFA500
OrangeRed	#FFFF4500
Orchid	#FFDA70D6
PaleGoldenrod	#FFEEE8AA
PaleGreen	#FF98FB98
PaleTurquoise	#FFAFEEEE

PaleVioletRed	#FFDB7093
PapayaWhip	#FFFFEFD5
PeachPuff	#FFFFDAB9
Peru	#FFCD853F
Pink	#FFFFC0CB
Plum	#FFB0E0E6
Purple	#FF800080
Red	#FFFF0000
RosyBrown	#FFBC8F8F
RoyalBlue	#FF4169E1
SaddleBrown	#FF8B4513
Salmon	#FFFA8072
SandyBrown	#FFF4A460
SeaGreen	#FF2E8B57
SeaShell	#FFFFF5EE
Sienna	#FFA0522D
Silver	#FFC0C0C0
SkyBlue	#FF87CEEB
SlateBlue	#FF6A5ACD
SlateGray	#FF708090

Snow	#FFFFFAFA
SpringGreen	#FF00FF7F
SteelBlue	#FF4682B4
Tan	#FFD2B48C
Teal	#FF008080
Thistle	#FFD8BFD8
Tomato	#FFFF6347
Turquoise	#FF40E0D0
Violet	#FFEE82EE
Wheat	#FFF5DEB3
White	#FFFFFFFF
WhiteSmoke	#FFF5F5F5
Yellow	#FFFFFF00
YellowGreen	#FF9ACD32

layer

Possible values

OdinPageLayer.Foreground = Drawing is done on top of the original document.

OdinPageLayer.Background = Drawing is done behind the original document.

Copyright<?rh-symbol_start name="Copyright" ?>©<?rh-symbol_end ?> 2008-2014 Hyland Software, Inc. and its affiliates.

SetFont

Defines the font specification for text output.

SetFont		
SYNTAX	SetFont(fontName, fontSize)	
ARGUMENTS	fontName	Name of the font, for example, Arial.
	fontSize	Size of the font, for example, 10.

SetInputBin

Sets the input tray for a page. The function overwrites the predefined tray for the first page or for the follow-up. depending on the case.

SetInputBin		
SYNTAX	SetInputBin(pageNumber, binName)	
ARGUMENTS	pageNumber	Page number
	binName	Name of the tray
EXAMPLE	;Set the input tray for the last page of the document streamDoc.SetInputBin(streamdoc.PageCount - 1, "testBin")	

TextOut

Writes a text on the current page.

CreateGuid	
SYNTAX	TextOut(x, y, text, bold, italic) TextOut(x, y, text, bold, italic, color, angle) TextOut(x, y, text, bold, italic, color, angle, layer)

ARGUMENTS

x	X-position of the starting point
y	Y-position of the starting point
text	Text to write
bold	TRUE = Formatting bold FALSE = Formatting not bold The default value is FALSE
italic	TRUE = Formatting italic FALSE = Formatting not italic The default value is FALSE
color	Color **Possible value** The HTML color name or the ARGB value in format #AARRGGBB (AA=Alpha-Transparency, RR=Red, GG=Green, BB=Blue).

HTML color name	ARGB value
AliceBlue	#FFF0F8FF
AntiqueWhite	#FFFAEBD7
Aqua	#FF00FFFF
Aquamarine	#FF7FFFD4
Azure	#FFF0FFFF
Beige	#FFF5F5DC
Bisque	#FFFFE4C4

Black	#FF000000
BlanchedAlmond	#FFFFEBCD
Blue	#FF0000FF
BlueViolet	#FF8A2BE2
Brown	#FFA52A2A
BurlyWood	#FFDEB887
CadetBlue	#FF5F9EA0
Chartreuse	#FF7FFF00
Chocolate	#FFD2691E
Coral	#FFFF7F50
CornflowerBlue	#FF6495ED
Cornsilk	#FFFFF8DC
Crimson	#FFDC143C
Cyan	#FF00FFFF
DarkBlue	#FF00008B
DarkCyan	#FF008B8B
DarkGoldenrod	#FFB8860B
DarkGray	#FFA9A9A9
DarkGreen	#FF006400
DarkKhaki	#FFBDB76B

DarkMagenta	#FF8B008B
DarkOliveGreen	#FF556B2F
DarkOrange	#FFFF8C00
DarkOrchid	#FF9932CC
DarkRed	#FF8B0000
DarkSalmon	#FFE9967A
DarkSeaGreen	#FF8FBC8B
DarkSlateBlue	#FF483D8B
DarkSlateGray	#FF2F4F4F
DarkTurquoise	#FF00CED1
DarkViolet	#FF9400D3
DeepPink	#FFFF1493
DeepSkyBlue	#FF00BFFF
DimGray	#FF696969
DodgerBlue	#FF1E90FF
Firebrick	#FFB22222
FloralWhite	#FFFFFAF0
ForestGreen	#FF228B22
Fuchsia	#FFFF00FF
Gainsboro	#FFDCDCDC

GhostWhite	#FFF8F8FF
Gold	#FFFFD700
Goldenrod	#FFDAA520
Gray	#FF808080
Green	#FF008000
GreenYellow	#FFADFF2F
Honeydew	#FFF0FFF0
HotPink	#FFFF69B4
IndianRed	#FFCD5C5C
Indigo	#FF4B0082
Ivory	#FFFFFFF0
Khaki	#FFF0E68C
Lavender	#FFE6E6FA
LavenderBlush	#FFFFF0F5
LawnGreen	#FF7CFC00
LemonChiffon	#FFFFFACD
LightBlue	#FFADD8E6
LightCoral	#FFF08080
LightCyan	#FFE0FFFF
LightGoldenrodYellow	#FFFAFAD2

LightGray	#FFD3D3D3
LightGreen	#FF90EE90
LightPink	#FFFFB6C1
LightSalmon	#FFFFA07A
LightSeaGreen	#FF20B2AA
LightSkyBlue	#FF87CEFA
LightSlateGray	#FF778899
LightSteelBlue	#FFB0C4DE
LightYellow	#FFFFFFE0
Lime	#FF00FF00
LimeGreen	#FF32CD32
Linen	#FFFAF0E6
Magenta	#FFFF00FF
Maroon	#FF800000
MediumAquamarine	#FF66CDAA
MediumBlue	#FF0000CD
MediumOrchid	#FFBA55D3
MediumPurple	#FF9370DB
MediumSeaGreen	#FF3CB371
MediumSlateBlue	#FF7B68EE

MediumSpringGreen	#FF00FA9A
MediumTurquoise	#FF48D1CC
MediumVioletRed	#FFC71585
MidnightBlue	#FF191970
MintCream	#FFF5FFFA
MistyRose	#FFFFE4E1
Moccasin	#FFFFE4B5
NavajoWhite	#FFFFDEAD
Navy	#FF000080
OldLace	#FFFDF5E6
Olive	#FF808000
OliveDrab	#FF6B8E23
Orange	#FFFFA500
OrangeRed	#FFFF4500
Orchid	#FFDA70D6
PaleGoldenrod	#FFEEE8AA
PaleGreen	#FF98FB98
PaleTurquoise	#FFAFEEEE
PaleVioletRed	#FFDB7093
PapayaWhip	#FFFFEFD5

PeachPuff	#FFFFDAB9
Peru	#FFCD853F
Pink	#FFFFC0CB
Plum	#FFB0E0E6
Purple	#FF800080
Red	#FFFF0000
RosyBrown	#FFBC8F8F
RoyalBlue	#FF4169E1
SaddleBrown	#FF8B4513
Salmon	#FFFA8072
SandyBrown	#FFF4A460
SeaGreen	#FF2E8B57
SeaShell	#FFFFF5EE
Sienna	#FFA0522D
Silver	#FFC0C0C0
SkyBlue	#FF87CEEB
SlateBlue	#FF6A5ACD
SlateGray	#FF708090
Snow	#FFFFFAFA
SpringGreen	#FF00FF7F

SteelBlue	#FF4682B4
Tan	#FFD2B48C
Teal	#FF008080
Thistle	#FFD8BFD8
Tomato	#FFFF6347
Turquoise	#FF40E0D0
Violet	#FFEE82EE
Wheat	#FFF5DEB3
White	#FFFFFFFF
WhiteSmoke	#FFF5F5F5
Yellow	#FFFFFF00
YellowGreen	#FF9ACD32

angle — Angle of rotation

layer — **Possible values**

OdinPageLayer.Foreground = Drawing is done on top of the original document.

OdinPageLayer.Background = Drawing is done behind the original document.

Context OdinFilter

About the Context OdinFilter

Use this context in filter scripts.

Functions of the FilterObject

About the Filter Object

The variable **filter** provides a filter object with the following functions.

DefinePostageDefinitionByName

DefinePostageDefinitionByOId

DefinePrinterByName

DefinePrinterByOId

EnumEnvelopes

Gen_Stack

GetParam

RunFilter

SetParam

DefinePostageDefinitionByName

Defines a postage definition by its name.

DefinePostageDefinitionByName		
SYNTAX	DefinePostageDefinitionByName(PostageDefinitionName)	
ARGUMENTS	PostageDefinitionName	Name of the postage definition.

DefinePostageDefinitionByOId

Defines a postage definition by its object ID.

DefinePostageDefinitionByOId		
SYNTAX	DefinePostageDefinitionByOId(postageOID)	
ARGUMENTS	postageOID	Object ID of the postage definition.

DefinePrinterByName

Defines a printer by its name and type.

DefinePrinterByName

SYNTAX DefinePrinterByName(printerName,type)

ARGUMENTS

printerName	Printer name.
printerType	Printer type.

Possible values	Description
ModusSuiteObjectType.Printer	Online printer.
ModusSuiteObjectType.EMailPrinter	Email printer.
ModusSuiteObjectType.OutsourcingPrinter	Outsourcing printer.

DefinePrinterByOId

Defines a printer by its object ID.

DefinePrinterByOId

SYNTAX DefinePrinterByOId(printerOID)

ARGUMENTS

printerOID	Object ID of the printer.

EnumEnvelopes

Invokes an instance of an envelope object to create envelopes.

Before an envelope object can be used, it must first be invoked by calling the Init function. If the envelope object is no longer required, it must be revoked by the FreeObject command in the filter script.

EnumEnvelopes	
SYNTAX	EnumEnvelopes()
RETURN	Instance of an envelope object.

Gen_Stack

Creates a stack and assigns the relevant job and envelope datasets to it.

The **UPDATE_SQL** filter object parameter must be set before this function is called. This parameter contains the update statement assigning job and envelope datasets to the new stack.

Gen_Stack		
SYNTAX	Gen_Stack(updateParameter, customStackColumns)	
ARGUMENTS	updateParameter	Contains name and value pairs, for example, Postage=100. If you set more than one pair, you must separate each pair by a comma. These pairs are used to replace placeholders in the SQL update statement with real values. Placeholders in an SQL update statement are identified by ':'
	customStackColumns	This parameter allows you to insert data into customer-specific columns in the stack table when a stack is created. If not required, you must define an empty string. As with the UpdateParameter, the string submitted here contains name and value pairs. For example, to fill the customer-specific columns ODS_INFO and ODS_EXT with data on insert, the values ODS_INFO=SpecialInfo,ODS_EXT=10 must be submitted. If the value contains quotation marks, equal signs, or commas, the name and value pair must be enclosed by quotation marks. **Example** ODS_CUSTOM1=123,"ODS_CUSTOM2=test,123=0815"

	Quotation marks have to de duplicated in a script. **Example** StackResult = Filter.Gen_Stack(UpdateParameter, "ODS_CUSTOM1=123,""ODS_ CUSTOM2=test,123=0815""")

RETURN The result here is a structure, not a basic data type, containing the following attributes:

Attribute	Definition
NewStackId	ID of the new stack dataset or an empty string, if no stack was created (no jobs or envelopes were assigned).
UpdatedRecordCount	The number of datasets updated by the Update-SQL. This refers to the number of job or envelope datasets assigned to the stack.

GetParam

Retrieves the value of a filter parameter.

GetParam

SYNTAX GetParam(parameterName)

ARGUMENTS

parameterName	Name of the filter parameter.

RETURN Parameter value.

RunFilter

Allows you to run another filter script from a filter script itself.

ReplaceFile

SYNTAX RunFilter(filterScriptName)

ARGUMENTS	filterScriptName	Name of the script to execute.

SetParam

Sets a filter parameter.

SetParam		
SYNTAX	SetParam(parameterName, parameterValue)	
ARGUMENTS	parameterName	Name of the filter parameter
	parameterValue	New value of the filter parameter

Overview of Possible Parameters for the Functions GetParam and SetParam

DBALIAS_NAME

Reads the name of the database alias defined in the settings of the OdinFilter component.

FILTERNAME

Allows you to define the value inserted into the field **ODS_Filtername** of the new stack. Is taken into account when creating a stack with **Gen_Stack**.

HOLDTIME

Allows you to overwrite the default print time value for a new stack.
Is taken into account when creating a stack with **Gen_Stack**.

MESSAGETEXT

Allows you to define the value inserted into the field **ODS_Messagetext** of the new stack. Is taken into account when creating a stack with **Gen_Stack**.

OMR_TYP

Allows you to define the value inserted into the field **ODS_OMR_TYPE** of the new stack. Is taken into account when creating a stack with **Gen_Stack**.

POSTAGEDEF_STR

Read postage class name of the postage definition set by using a variant of **DefinePostageDefinition**. Each name must be separated by commas.

PRIORITY

Allows you to overwrite the default priority value for a new stack.
Is taken into account when creating a stack with **Gen_Stack**.

STACKOBJECTSQL

Optional definition of a key in the Lookup Table **OdinStackObjectSql**.

The value of the key is an order by statement to be executed instead of the standard SQL when stack jobs or envelopes are loaded.

This information is stored in the field **ODJ_STACKOBJECTSQL** of the corresponding stack created and is evaluated when loading stack jobs or envelopes.

Example:

If the key **Test** is to be used by this stack to load its envelopes, the order by statements are retrieved from the lookup table **OdinStackObjectSql** in the keys **Test.Envelope_OrderBy** and **Test.Job_OrderBy**.

Structure of the Lookup Table OdinStackObjectSQL

2 values must be defined for each key, one for envelopes and one for jobs:

<Key>.Envelope_OrderBy=<ORDERBY-STATEMENT>

<Key>.Job_OrderBy=<ORDERBY-STATEMENT>

Sample Lookup Table

default.Envelope_OrderBy=ODE_SEQUENCE ASC

default.Job_OrderBy=ODJ_SEQUENCE ASC

Test.Envelope_OrderBy= ODE_REC_NAME DSC

Test.Job_OrderBy=ODJ_SORTING ASC

STACKTYPE

Allows you to overwrite the default value for stack type for a new stack.
Is taken into account when creating a stack with **Gen_Stack**.

STATE

Allows you to overwrite the default value for stack status for a new stack.
Is taken into account when creating a stack with **Gen_Stack**.

UPDATE_SQL

This parameter sets the SQL update statement to be used to assign jobs / envelopes to the new stack when the function **Gen_Stack** is executed.

Sample Script: Simple Filter Including Stack Generation

```
;***********************************************
; Filter Script Demo
; simple Filter including Stack generation
;***********************************************
;
filterName = "FilterOnline"
printerName = "PrinterOnline"
stackState = 0
;
;Get number of open Jobs
DatabaseAliasName = filter.GetParam("DBALIAS_NAME")
InitDataset("OdinJob", DatabaseAliasName)
;
StringListAdd("JobCountSql", "Select count(*) JobCount from Odin_job Where (ODJ_Stack_Id is null) and (Odj_Env_Id is null)")
SetSQL("OdinJob", "JobCountSql")
OpenSQL("OdinJob")
JobCount = GetField("OdinJob", "JobCount")
;
if (JobCount > 0)
  ;
  ;Open Jobs exist -> generate Stack
  Protocol("{0} Jobs for processing found", 10, JobCount)
  ;
  ; SQL to update open Jobs
  UpdateSql =
  "Update Odin_Job "+
```

```
  "Set ODJ_Stack_Id = :StackId, ODJ_Printer_OID = :PrinterOId, ODJ_Printer_Name = :PrinterName,
ODJ_Printer_Type = :PrinterType " +
  "Where (ODJ_Stack_Id is null) and (Odj_Env_Id is null) and (ODJ_State = 1)"
  ;
  filter.SetParam("UPDATE_SQL", UpdateSql)
  ;
  filter.SetParam("holdtime", "0")
  filter.SetParam("priority", "200")
  filter.SetParam("STACKTYPE", "0")
  filter.SetParam("OMR_TYP", "")
  filter.SetParam("STATE", stackState)
  filter.SetParam("filtername", filterName)
  ;
  filter.DefinePrinterByName(printerName, ModusSuiteObjectType.Printer)
  ;
  ;Alternative, if information has to be inserted into custom fields of the stack table
  ;my_result = filter.Gen_Stack("RecCity=NSU", "ODS_INFO=test,ODS_INT_INFO=10")
  my_result = filter.Gen_Stack("", "")
  ;
  stackId = my_result.NewStackId
  updCount = my_result.UpdatedRecordCount
  if (updCount > 0)
    ;
    Protocol("Stack {0} with {1} jobs created", 8, stackId, updCount)
    ;
  end-if
  ;
  Protocol("UpdatedRecordCount: " + my_result.UpdatedRecordCount, 10)
  Protocol("NewStackId: " + my_result.NewStackId, 10)
  ;
```

```
else
  ;
  Protocol("No Jobs for processing found", 10)
  ;
end-if
```

Functions of the Envelope Object

About the Envelope Object

The envelope object enables you to create envelope datasets.

AppendVJob

Inserts a virtual job behind the internal cursor position in the internal job table.

AppendVJob		
SYNTAX	AppendVJob(renditionFileName, pageCount, formName)	
ARGUMENTS	renditionFileName	Full path name statement to the XPS file you want to use as the document for the virtual job.
	pageCount	Number of document pages.
	formName	Name of the form.

CalcEnvelopePages

Calculates the number of pages for the current envelope (1 sheet has 2 pages).

Takes into account all documents of the job datasets and their respective enclosures. Enclosures in the postage definition are not taken into account.

CalcEnvelopePages	
SYNTAX	CalcEnvelopePages()
RETURN	Number of pages.
EXAMPLE	CalcEnvelopePages()

CalcEnvelopePapers

Calculates the number of sheets required for the current envelope (1 sheet has 2 pages).

Takes into account all documents of the job datasets and their respective enclosures. Enclosures in the postage definition are not taken into account.

CalcEnvelopePapers	
SYNTAX	CalcEnvelopePapers()
RETURN	Number of sheets.
EXAMPLE	CalcEnvelopePapers()

CalcEnvelopePostage

Calculates the postage class of the current envelope. Basis for calculating the postage class required is the weight of all documents in the job datasets and their enclosures as defined in the ODJ_Annex database field.

Enclosures in the postage definition are not taken into account.

CalcEnvelopePostage	
SYNTAX	CalcEnvelopePostage()
RETURN	Name of the postage class used.
EXAMPLE	CalcEnvelopePostage()

CalcEnvelopeWeight

Calculates the weight of the current envelope

Takes into account all documents of the job datasets and their respective enclosures (Odj_Annex).

Enclosures in the postage definition are not taken into account.

Syntax

CalcEnvelopeWeight()

Return value

Weight in grams

Commit

Ends the database transaction started with the StartTransaction command. All data modifications performed since the start of the transaction become a permanent part of the database.

Commit	
SYNTAX	Commit()
EXAMPLE	Commit()

Count

Calculates the number of jobs contained in the envelope.

Count	
SYNTAX	Count()
RETURN	Number of jobs.

CreateEnv

Creates a new envelope in the database and assigns the currently in the internal job table selected jobs to the envelope.

Takes into account any split markers that have been set.

CreateEnv	
SYNTAX	CreateEnv()
RETURN	The database ID of the created envelope record. If more than one envelope is created, the returned string contains a comma separated list of all IDs.

EndTimer

Stops the timer started previously with the StartTimer command and returns the elapsed time in milliseconds.

EndTimer

SYNTAX	EndTimer()
RETURN	Number of milliseconds elapsed.

Eof

Determines whether the cursor is at the end of the dataset in the internal job table.

Eof	
SYNTAX	Eof()
RETURN	TRUE = Eof reached. FALSE = Eof not reached.

First

Positions the cursor on the first record in the internal job table.

First	
SYNTAX	First()

GetField

Reads the content of a field in the current dataset in the internal job table.

GetField		
SYNTAX	GetField(fieldName)	
ARGUMENTS	fieldName	Name of the column.
RETURN	Field value.	

GotoJobTillWeight

Starts from the current cursor position in the internal job table and moves the cursor forward as long as the total weight of each individual job does not exceed the defined weight.

Setting a split marker at this position allows you to split an envelope into smaller envelopes.

GotoJobTillWeight		
SYNTAX	GotoJobTillWeight(tillWeight)	
ARGUMENTS	tillWeight	Maximum weight not to be exceeded by the sum of individual job weights.
RETURN	TRUE = The cursor has reached the end of the internal job table before the maximum weight was reached. FALSE = The cursor did not reach the end of the internal job table before the maximum weight was reached.	

GotoRecNum

Moves the cursor to a defined dataset in the internal job table.

GotoRecNum		
SYNTAX	GotoRecNum(recNumber)	
ARGUMENTS	recNumber	0-based index of the dataset where you want to position the cursor.

Init

Initializes the envelope object. This function must be the first function called after the envelope object and filter object have been set. Initialized envelope objects must be freed at the end of a filter script with the FreeObject function.

IncDate	
SYNTAX	Init(groupFields, jobSqlQuery, takeOverFields, takeOverValues)

ARGUMENTS	groupFields	Comma-separated column names of the job table defining the group change.
	jobSqlQuery	Select statement to determine the job records for logical envelope stuffing.
	takeOverFields	Additional fields in the ODIN_ENVELOPE table to be filled when an envelope dataset is created.
	takeOverValues	Information about where the values for the additional columns come from. It is possible to use column names from the job table (specified with **':'** before the name) as well as constant values.

InsertVJob

Inserts a virtual job before the internal cursor position in the internal job table.

InsertVJob		
SYNTAX	InsertVJob(renditionFileName, pageCount, formName)	
ARGUMENTS	renditionFileName	Full path name statement to the XPS file used as the document for the virtual job.
	pageCount	Number of document pages.
	formName	Name of the form.

Last

Positions the cursor on the last record in the internal job table.

Last	
SYNTAX	Last()
EXAMPLE	EnvEnum.Last()

Next

Positions the cursor on the next record in the internal job table.

Next	
SYNTAX	Next()

Prev

Positions the cursor on the previous record in the internal job table.

Prev	
SYNTAX	Prev()

ReadNextEnvelope

Calculates the datasets for the next envelope according to specified group change fields.

For more information, refer to the Init function groupFields parameter.

ReadNextEnvelope	
SYNTAX	ReadNextEnvelope()
RETURN	The number of jobs in this envelope or 0 if the end of the job table is reached.

Recnum

Reads the record index of the current cursor position in the internal job table.

Recnum	
SYNTAX	Recnum()
RETURN	Current record index. The return value is 0-based. The return value -1 means EOF.

Rollback

Ends the database transaction that was started with the command StartTransaction.

All data modifications performed since the start of the transaction are discarded.

This function should be called in the event of an error.

RollBack	
SYNTAX	RollBack()

SetField

Writes a value in the current dataset of the internal job table. The new value inserts into the database when the job is updated.

SetField		
SYNTAX	SetField(fieldName, value)	
ARGUMENTS	fieldName	Name of the column
	value	New value

Split

Sets a split marker at the current cursor position in the internal job table.

Allows you to split the datasets currently held in the internal job table into several envelopes.

Split	
SYNTAX	Split()

StartTimer

Launches the internal timer.

StartTimer	
SYNTAX	StartTimer()

StartTransaction

Starts a database transaction.

StartTransaction	
SYNTAX	StartTransaction()

Sample Script: Simple Enveloping Filter with Stack Generation

```
;************************************************
; Filter Script Demo
;Simple Enveloping Filter with Stack Generation
;************************************************
;
filterName = "FilterOutsourcing"
printerName = "PrinterOutsourcing"
postageDefinition = "PostageDefinition"
;
#region Postage/Printer
;
;set the Postage Definition
filter.DefinePostageDefinitionByName(postageDefinition)
;
;set the Printer assigned to the Jobs when the Envelope is created
filter.DefinePrinterByName(printerName, ModusSuiteObjectType.OutsourcingPrinter)
;
#endregion
```

```
;
#region SQL-Definition
;
;Group field change to define Envelope end
GroupFields = "ODJ_REC_NAME,ODJ_REC_AKZ,ODJ_REC_ZIP,ODJ_REC_CITY,ODJ_REC_
STREET"
;
;Select to load the Jobs for enveloping
;datasets are locked using the function "Updlock"
; you must type a space before and after " FROM "
; the where condition must at least contain the following minimum conditions:
;   ODJ_Stack_Id is NULL: Job is not assigned to a Stack
;   Odj_Env_Id is NULL: Job is not assigned to an Envelope
;   Odj_State = 1: Rendition created
JobSelect = "Select Odin_Job.* From Odin_Job with (Updlock) " +
"where (ODJ_Stack_Id is NULL) and (Odj_Env_Id is null) and (Odj_State = 1) " +
"order by ODJ_REC_NAME,ODJ_REC_AKZ,ODJ_REC_ZIP,ODJ_REC_CITY,ODJ_REC_STREET"
;
;As an alternative, SQL statements can be read from a LookupTable
;Example
;Sql-Statement should be read from the entry "JobSelect" in the LookupTable "odin"
;JobSelect = GetLookupValue("odin", "JobSelect")
;
;Content of columns from the Jobs to be inserted into the Envelope created
TakeOverFields = "ODE_REC_NAME,ODE_REC_AKZ,ODE_REC_ZIP,ODE_REC_CITY,ODE_REC_
STREET,ODE_DEVICE"
TakeOverValues = ":ODJ_REC_NAME,:ODJ_REC_AKZ,:ODJ_REC_ZIP,:ODJ_REC_CITY,:ODJ_REC_
STREET,Device x"
;
#endregion
;
```

```
;Create Envelope instance outside the try-block
EnvEnum = filter.EnumEnvelopes()
;
#region Envelope Stuffing
;
try
  ;initialize inside the block
  EnvEnum.Init(GroupFields, JobSelect, TakeOverFields, TakeOverValues)
  EnvEnum.StartTransaction()
  ;
  JobsInEnvelope = EnvEnum.ReadNextEnvelope()
  while (JobsInEnvelope > 0)
    ;
    EnvelopeId = EnvEnum.CreateEnv()
    JobsInEnvelope = EnvEnum.ReadNextEnvelope()
    ;
  end-while
  ;
  EnvEnum.Commit()
  ;
OnError
  ;
  EnvEnum.Rollback()
  Protocol("Exception: {0}", 10, Error.Message)
  Protocol("StackTrace: {0}", 10, Error.StackTrace)
  Raise(Error.Token, Error.Message)
  ;
finally
  ;
  ;free object to
```

```
  ;free up internal resources
  FreeObject("EnvEnum")
;
end-try
;
#endregion
;
#region create Stack for Envelope with no Stack assignment
;
;read the DbAlias of the Filter component
DatabaseAliasName = filter.GetParam("DBALIAS_NAME")
;
;Get number of open Envelopes
InitDataset("OdinEnv", DatabaseAliasName)
StringListAdd("EnvCountSql", "Select count(*) EnvCount from Odin_envelope Where (ODE_Stack_Id is
null)")
SetSQL("OdinEnv", "EnvCountSql")
OpenSQL("OdinEnv")
OpenEnvCount = GetField("OdinEnv", "EnvCount")
;
;create Stack if open Envelopes found
;A Stack with Envelopes is created for each Postage Class of the selected Postage Definition if open
Envelopes are found where the column ODE_POSTAGE
;
;contains the name of the Postage Class
if (OpenEnvCount > 0)
  ;
  filter.SetParam("filtername", filterName)
  filter.SetParam("holdtime", "10")
  filter.SetParam("priority", "200")
```

```
  filter.SetParam("STACKTYPE", "20")
  ;filter.SetParam("OMR_TYP", "C4_OMR")
  filter.SetParam("OMR_TYP", "")
  ;
  ;"StackId" is a protected keyword and is replaced by the ID of the Stack created during processing.
  ;"Postage" is a user-defined keyword. The value is defined in the UpdateParameter of the function Gen_
Stack.
  UpdateSql =
  "Update Odin_Envelope "+
  "Set ODE_Stack_Id = :StackId "+
  "Where (ODE_Stack_Id is null) and (ODE_POSTAGE = :Postage)"
  ;
  filter.SetParam("UPDATE_SQL", UpdateSql)
  ;
  ;returns the names of the Postage Classes defined in the loaded Postage Definition (comma separated)
  PostageNames = filter.GetParam("PostageDef_Str")
  ;
  while (PostageNames <> "")
    Postage = StrTok("PostageNames", ",")
    if (Postage = "")
      Postage = PostageNames
      PostageNames = ""
    end-if
    ;
    Filter.SetParam("FilterName", Postage)
    UpdateParameter = FormatStr("Postage=%s", Postage)
    StackResult = Filter.Gen_Stack(UpdateParameter, "")
    if (StackResult.NewStackId = "0")
      Protocol("No Stacks for Postage Class '{0}' created", 10, Postage)
    else
```

```
      Protocol("Stack for Postage Class '{0}' has the ID '{1}' and contains '{2}' Envelopes", 10, Postage,
StackResult.NewStackId, StackResult.UpdatedRecordCount)
    end-if
    ;
  end-while
;
else
  Protocol("No Envelopes for processing found", 10)
end-if
;
#endregion
```

Context OdinPrint

Properties of the Context OdinPrint

Parameter	Description
CurrentEnvelope	Current envelope, if found. Read-only property. **Functions** GetValue Returns the value of the given column.

GetValue

SYNTAX	GetValue(string columnName)
ARGUMENTS	columnName — Column name
RETURN	Column value If the data type of the database column is DATE, DATETIME or any valid date type, the date read generally returns in the format MM/DD/YYYY hh:mm:ss.

EXAMPLE	GetObject("DateVar","MLDate") … StringDateVar = CurrentEnvelope.GetValue("ODE_CREATE_AT") ;Convert date to the required format with the help of the MLDate object IntDateVar = DateVar.DateStringToInt64("MM/dd/yyyy hh:mm:ss", StringDateVar) v_LETTERDATE= DateVar.Int64ToDateString("MM.dd.yyyy hh:mm:ss", IntDateVar)

GetFieldNameEnumerator

Returns an enumerator.

GetFieldNameEnumerator

SYNTAX	GetFieldNameEnumerator()
RETURN	Enumerator with the following methods.

Method	Description
Current	Returns the current object.
MoveNext	Moves the position to the next object to be processed.
Reset	Resets the enumerator to the beginning of the first element.

CurrentJob

Current job, if found.

Read-only property.

Functions

GetValue

Returns the value of the given column.

GetValue

SYNTAX	GetValue(string columnName)
ARGUMENTS	columnName Column name
RETURN	Column value If the data type of the database column is DATE, DATETIME or any valid date type, the date read generally returns in the format MM/DD/YYYY hh:mm:ss.
EXAMPLE	GetObject("DateVar","MLDate") … StringDateVar = CurrentEnvelope.GetValue("ODE_CREATE_AT") ;Convert date to the required format with the help of the MLDate object IntDateVar = DateVar.DateStringToInt64("MM/dd/yyyy hh:mm:ss", StringDateVar) v_LETTERDATE= DateVar.Int64ToDateString("MM.dd.yyyy hh:mm:ss", IntDateVar)

GetFieldNameEnumerator

Returns an enumerator.

GetFieldNameEnumerator

SYNTAX	GetFieldNameEnumerator()
RETURN	Enumerator with the following methods.

Method	Description
Current	Returns the current object.
MoveNext	Moves the position to the next object to be processed.
Reset	Resets the enumerator to the beginning of the first element.

CurrentStack

Current stack.

Read-only property.

Functions

GetValue

Returns the value of the given column.

GetValue	
SYNTAX	GetValue(string columnName)
ARGUMENTS	columnName Column name
RETURN	Column value If the data type of the database column is DATE, DATETIME or any valid date type, the date read generally returns in the format MM/DD/YYYY hh:mm:ss.
EXAMPLE	GetObject("DateVar","MLDate") … StringDateVar = CurrentEnvelope.GetValue("ODE_CREATE_AT") ;Convert date to the required format with the help of the MLDate object IntDateVar = DateVar.DateStringToInt64("MM/dd/yyyy hh:mm:ss", StringDateVar) v_LETTERDATE= DateVar.Int64ToDateString("MM.dd.yyyy hh:mm:ss", IntDateVar)

GetFieldNameEnumerator

Returns an enumerator.

GetFieldNameEnumerator	
SYNTAX	GetFieldNameEnumerator()
RETU	Enumerator with the following methods.

RN

Method	Description
Current	Returns the current object.
MoveNext	Moves the position to the next object to be processed.
Reset	Resets the enumerator to the beginning of the first element.

Functions of the Context OdinPrint

HasData

Checks whether data exists for the given type.

HasData

SYNTAX HasData(OdinDataRowType dataType)

ARGUMENTS

dataType	**Possible values**
	OdinDataRowType.Process
	OdinDataRowType.Stack
	OdinDataRowType.Envelope
	OdinDataRowType.Job
	OdinDataRowType.VirtualJob

RETURN TRUE = The data row specified exists.

FALSE = The data row specified does not exist.

Context OdinRendition

About the Context OdinRendition

Use this context in scripts that convert document files of jobs to the Odin spool file format (XPS format).

Properties of the Context OdinRendition

Property	Description
CurrentJobRecord	Read or write access to the current job dataset.
CurrentProcessId	Specifies the ID of the currently processed Odin process.
ProcessRow	Read access to the process dataset.
JobCount	Returns the number of jobs of the current process.

Functions of the Context OdinRendition

GetFieldNameEnumerator

Returns an enumerator.

GetFieldNameEnumerator

SYNTAX GetFieldNameEnumerator()

RETURN Enumerator with the following functions.

Current	Returns the current object.
MoveNext()	Moves the position to the next object to process. SYNTAX MoveNext() RETURN TRUE = Successful FALSE = The end of the collection is passed.
Reset	Resets the enumerator to the beginning of the first element.

GetJobRow

Provides read or write access to the specified job dataset.

GetJobRow	
SYNTAX	GetJobRow(rowIndex)
ARGUMENTS	rowIndex — Index of the job row as integer.
RETURN	The jov row.
EXAMPLE	jobRow = GetJobRow(counter) ODJ_ID = jobRow.GetValue("ODJ_ID") ; read value jobRow.SetValue("ODJ_ARCHIVED", "Y") ; set value

GetValue

Returns the value of the given column.

GetValue	
SYNTAX	GetValue(columnName)
ARGUMENTS	columnName — Name of the column.
RETURN	Column value. If the data type of the database column is DATE, Datetime or any valid date type, the date read generally returns in the format MM/DD/YYYY hh:mm:ss.
EXAMPLE	GetObject("DateVar","MLDate") … StringDateVar = CurrentEnvelope.GetValue("ODE_CREATE_AT") ;Convert date to the required format with the help of the MLDate object IntDateVar = DateVar.DateStringToInt64("MM/dd/yyyy hh:mm:ss", StringDateVar) v_LETTERDATE= DateVar.Int64ToDateString("dd.MM.yyyy hh:mm:ss ", IntDateVar)

HasData

Checks whether data exists for the given type.

HasData

SYNTAX HasData(OdinDataRowType dataType)

ARGUMENTS

dataType	**Possible values**
	OdinDataRowType.Process
	OdinDataRowType.Stack
	OdinDataRowType.Envelope
	OdinDataRowType.Job
	OdinDataRowType.VirtualJob

RETURN TRUE = The data row specified exists.

FALSE = The data row specified does not exist.

SetValue

Writes a value in a database column of the current job dataset.

SetSQL

SYNTAX SetValue(columnName, newValue)

ARGUMENTS

columnName	Column name
newValue	Column value

EXAMPLE

```
Printer = CurrentJobRecord.GetValue("ODJ_PRINTER_OID")
if (Printer = "")
  CurrentJobRecord.SetValue("ODJ_PRINTER_OID", "test-pr-5")
end-if
```

Context OdinStream

About the Context OdinStream

Use this context in scripts to provide a stream object that can be accessed with **Stream**.

Properties of the Context OdinStream

Show All

Property	Description
Stream	Returns the IOdinStreamDocument interface. Script alias: stream **functions of the IOdinStream-Interface** AppendDocument GetProperty LoadDocument LoadEnvelopes LoadJobs SetProperty Sample Script LoadEnvelopes / LoadJobs
HasStream	TRUE = A stream object exists in the **Stream** context property. FALSE = No stream object exists in the **Stream** context property.

Functions of the IOdinStream-Interface

AppendDocument

Adds a document loaded with LoadDocument to a stream.

AppendDocument	
SYNTAX	AppendDocument(document, insertOMR, insertDVFrei)

ARGUMENTS	document	The document you want to add.
	insertOMR	Defines whether OMR codes are printed on the document. **Note** OMR codes are not printed on a stack cover page.
	insertDVFrei	Parameter reserved for later implementation. No current function.
EXAMPLE	document = stream.LoadDocument("d:\import\Empty_Page.xps", OdinDuplexSetting.Simplex, "firstbin","nextbin") stream.AppendDocument(document, true, false)	

GetProperty

Retrieves streaming values, for example page counter.

GetProperty

SYNTAX GetProperty(propertyName)

ARGUMENTS	propertyName	Name of the property.

RETURN Property value.

You can query the following properties. The identifiers in brackets are those defined in Modus 4 that remain valid to ensure backward compatibility. These are automatically converted to the corresponding new names.

General

Property	Description
EnvelopeId	Envelope ID of the envelope currently streamed.

JobId	Job ID of the job or document currently streamed.
OMRCode	Current value of the OMR code. Read-only property.
OMRReset	TRUE = A reset flag (all OMR marks set) is printed on all subsequent documents instead of the OMR code. The reset flag is printed until the property is set to FALSE. The OnStackEnd event enables to insert an additional document or an additional page after the stack, which contains a reset flag for the production inserter.
StackId	Stack ID of the stack to stream.
StreamFileName	Name of the stream file.
Sysdate	Current date.
Systime	Current time.
VirtualPageDescription (VPageDesc)	For a stack cover page = StackCoverPage. For an envelope cover page = EnvelopeCoverPage. Otherwise an empty string.

Stack

Property	Description
StackEnvelope	Envelope counter for the whole stack.
StackJob	Job or document counter for the whole stack.
StackPage	Page counter for the whole stack.
StackPaper	Sheet counter for the whole stack.
TotalStackEnvelopes (EnvCount)	Number of envelopes in a stack.

TotalStackJobs	Number of jobs or documents in a stack.
TotalStackPages (StackPage)	Number of pages in a stack.
TotalStackPapers	Number of sheets in a stack.

Envelope

Property	Description
EnvelopeEnd	TRUE = The current document is the last document of the envelope. FALSE = The current document is not the last document of the envelope.
EnvelopeJob	Job or document counter for the whole envelope.
EnvelopePage	Page counter for the whole envelope.
EnvelopePaper	Sheet counter for the whole envelope.
EnvelopeStart (EnvStart)	TRUE = The current document is the first document of the envelope. FALSE = The current document is not the first document of the envelope.
TotalEnvelopeJobs (JobCount)	Number of jobs or documents in an envelope.
TotalEnvelopePages	Number of pages in an envelope.
TotalEnvelopePapers	Number of sheets in an envelope.

Job / Document

Property	Definition
JobPage (PageCount)	Page counter for a document.
JobPaper (PageCountExtended)	Sheet counter for a document.
TotalJobPages	Number of pages in a document.
TotalJobPapers	Number of sheets in a document.

EXAMPLE

```
; Stack was processed by streaming...
; As an example, an additional page is inserted containing details of the number of pages/sheets, envelopes/jobs
;
TotalPages = stream.GetProperty("TotalStackPages")
TotalPapers = stream.GetProperty("TotalStackPapers")
TotalEnvelopes = stream.GetProperty("TotalStackEnvelopes")
TotalJobs = stream.GetProperty("TotalStackJobs")
;
; Load cover page for tray "coverpage"
document = stream.LoadDocument("c:\PerceptiveSoftware\modusSuite\data\EmptyPage.xps", OdinDuplexSetting.Simplex, "coverpage", "coverpage")
;
; print text as a test
document.SetFont("Tahoma", 16)
document.EditPage(0)
document.TextOut(20.5, 10.5, FormatStr("TotalPages/TotalPapers: %d/%d", TotalPages, TotalPapers), false, false)
document.TextOut(20.5, 17, "Number Envelopes/Jobs: %d/%d", TotalEnvelopes, TotalJobs), false, false)
document.Post()
;
; Print OMR-Reset character
```

```
Stream.SetProperty("OMRReset", true)
;
; Add document to stream
stream.AppendDocument(document, true, false)
;
; Free up the loaded document
FreeObject("document")
```

LoadDocument

Loads an XPS document. You can then add this document as an additional stack or envelope cover page or as a virtual document by using the AppendDocument function.

LoadDocument

SYNTAX	InitDataset(datasetName, aliasName, tableName)
ARGUMENTS	fileName — Name of the document to load. duplex — **Possible values** OdinDuplexSetting.Simplex: Simplex mode - no duplex OdinDuplexSetting.Horizontal: Horizontal duplex (long edge) OdinDuplexSetting.Vertical: Vertical duplex (short edge) firstBin — Tray for the first page. nextBin — Tray for the following page.
RETURN	Returns the IOdinStreamDocument interface with the functions required to perform drawing operations on pages.
EXAMPLE	document = stream.LoadDocument("d:\import\Empty_Page.xps", OdinDuplexSetting.Simplex, "firstbin","nextbin")

LoadEnvelopes

Loads all envelope datasets of a stack to the virtual ENVELOPE table for further processing.

LoadEnvelopes	
SYNTAX	LoadEnvelopes()
RETURN	Number of envelope datasets.

LoadJobs

Determines the envelope ID of the current envelope dataset, loads all corresponding envelope jobs, and makes them available in the virtual JOB table.

LoadJobs	
SYNTAX	LoadJobs()
RETURN	Number of job datasets.

SetProperty

Allows you to specify new settings for properties.

Note Altering values affects counters. Use the function with great care.

SetProperty		
SYNTAX	SetProperty(propertyName, value)	
ARGUMENTS	propertyName	Property name
	value	Property value

Sample Script LoadEnvelopes / LoadJobs

The functions of the VirtualDatabase, GetField() for example, provide access to stack, envelope and job data.

Example 1

ods_id = GetField("STACK", "ODS_ID")

ode_id = GetField("ENVELOPE", "ODE_ID")

```
odj_id = GetField("JOB", "ODJ_ID")
```

Example 2

```
; load stack envelopes
envelopeCount = stream.LoadEnvelopes()
First("ENVELOPE")
while (Eof("ENVELOPE") = false)
;
  ; Envelope ID
  EnvelopeId = GetField("ENVELOPE", "ODE_ID")
  ; Number of jobs in the envelope
  EnvelopeJobCount = GetField("ENVELOPE", "ODE_JOB_COUNT")
  ; load all jobs to the envelope
  JobsInEnvelope = stream.LoadJobs();
  ;
  ; Check the number of jobs defined in the envelope matches the number of actual jobs
  if (EnvelopeJobCount <> JobsInEnvelope)
    Protocol("Number of jobs is different (ODE_JOB_COUNT={0} / Job-Anzahl={1}", 10, EnvelopeJobCount,
JobsInEnvelope)
  end-if
  ;
  ; get all job IDs
  while (Eof("JOB") = false)
    JobId = GetField("Job", "ODJ_ID")
    ;
    Next("Job")
  end-while
  ;
  Next("ENVELOPE")
end-while
```

Functions of the Context OdinStream

HasData

Checks whether data exists for the given type.

HasData		
SYNTAX	HasData(OdinDataRowType dataType)	
ARGUMENTS	dataType	**Possible values** OdinDataRowType.Process OdinDataRowType.Stack OdinDataRowType.Envelope OdinDataRowType.Job OdinDataRowType.VirtualJob
RETURN	TRUE = The data row specified exists. FALSE = The data row specified does not exist.	

Context OdinStreamDocument

About the Context OdinStreamDocument

Use this context in scripts to provide a stream object that can be accessed with **streamDoc**.

Note All position information, as well as width and height, are specified in millimeters.

Properties of the IOdinStreamDocument-Interface

Show All

Property	Description
Stream	Returns the **IOdinStreamDocument** interface. Script alias: stream functions of the IOdinStream-Interface

	AppendDocument GetProperty LoadDocument LoadEnvelopes LoadJobs SetProperty Sample Script LoadEnvelopes / LoadJobs For more information, refer to Context OdinStream.
StreamDoc	Returns the **IOdinStreamDocument** interface of the currently loaded job or document. This document automatically inserts after the **OnDocument** script executes unless it is added by means of `AppendDocument`. This enables you to insert a virtual document before or after the current document by using `LoadDocument` and `AppendDocument`. Script alias: streamdoc For more information, refer to Functions of the IOdinStreamDocument-Interface. AddBookmark AddComment AddPJLCommand EditPage GetInputBin Image IsDuplex IsPortrait Line PageCount Post Rectangle SetAfpCopyGroup SetFont

SetInputBin TextOut

Functions of IOdinStreamDocument-Interface

AddBookmark

Adds a bookmark to a document. If the output format is PDF, a bookmark is created for every bookmark defined. The IDs must be unique for each bookmark.

AddBookmark		
SYNTAX	AddBookmark(string id, string parentId, string title) AddBookmark(string id, string parentId, string title, double x, double y)	
ARGUMENTS	id	Bookmark ID
	parentId	An empty string here means the bookmark is inserted at the highest level. Otherwise, the bookmark is inserted as a child of the bookmark specified.
	title	Text for the bookmark.
	x	X-position on the page.
	y	Y-position on the page.
EXAMPLE	document.AddBookmark("doc1_page1", "doc1", "Page 1") document.AddBookmark("doc1_page2", "doc1", "Page 2", 0, 0)	

AddComment

Writes a comment in the output file that means a NOP record in AFP.

AddComment

SYNTAX	AddComment(string text)
ARGUMENTS	text Comment text
EXAMPLE	; Add a NOP record to the first page streamdoc.EditPage(0) streamdoc.AddComment("my internal NOP-record for testing") streamdoc.Post()

Codepage for NOP comments

You can set the codepage for NOP comments in the AFP profile file **mffafp.pro** at the XML element `commentsdefaultcodepage`.

Examples

- <commentsdefaultcodepage iana="IBM850" />
- <commentsdefaultcodepage iana="windows-1252" />

AddPJLCommand

Writes a PJL command in a Postscript or PCL output file.

AddPJLCommand	
SYNTAX	AddPJLCommand(string value)
ARGUMENTS	value PJL-Command
EXAMPLE	; Add PJL-Command for staple and punch to the first page streamdoc.EditPage(0) streamdoc.AddPJLCommand("@PJL SET STAPLE=OFF") streamdoc.AddPJLCommand("@PJL SET PUNCH=OFF") streamdoc.Post()

EditPage

EditPage sets the given page to edit mode so that character commands can be executed on the page.

EditPage		
SYNTAX	EditPage(pageNumber)	
ARGUMENTS	pageNumber	Number of the page to be processed, beginning with 0.

GetInputBin

Returns the input bin of a page. If the input bin is not set, it returns the tray for the first page or for the subsequent tray, depending on the page number.

GetInputBin		
SYNTAX	GetInputBin(pageNumber)	
ARGUMENTS	pageNumber	Page number.
RETURN	Input bin.	

Image

Inserts an image file.

Image		
SYNTAX	Image(x, y, fileName) Image(x, y, imageFileName, layer)	
ARGUMENTS	x	X-position.
	y	Y-position.
	filename	File name.
	layer	**Possible Values**

	OdinPageLayer.Foreground = Drawing is done on top of the original document. OdinPageLayer.Background = Drawing is done behind the original document.
RETURN	TRUE = The user has the corresponding object rights. FALSE = The user does not have the corresponding object rights.

IsDuplex

Returns the duplex mode.

IsDuplex	
SYNTAX	IsDuplex()
RETURN	TRUE = Duplex FALSE = Simplex

IsPortrait

Returns the portrait mode.

IsPortrait		
SYNTAX	IsPortrait(pageNumber)	
ARGUMENTS	pageNumber	Page number.
RETURN	TRUE = Portrait FALSE = Landscape	

Line

Draws a line on the current page.

Line

SYNTAX

Line(x1, y1, x2, y2)

Line(x1, y1, x2, y2, lineWidth, lineStyle, color)

Line(x1, y1, x2, y2, lineWidth, lineStyle, color, layer)

ARGUMENTS

x1	X-position of the starting position of the line.
y1	Y-position of the starting position of the line.
x2	X-position of the end position of the line.
y2	Y-position of the end position of the line.
lineWidth	Line width.
lineStyle	**Possible values** OdinLineStyle.Solid OdinLineStyle.Dashed OdinLineStyle.Dotted
color	Color **Possible value** The HTML color name or the ARGB value in format #AARRGGBB (AA=Alpha-Transparency, RR=Red, GG=Green, BB=Blue).

HTML color name	ARGB value
AliceBlue	#FFF0F8FF
AntiqueWhite	#FFFAEBD7
Aqua	#FF00FFFF

Aquamarine	#FF7FFFD4
Azure	#FFF0FFFF
Beige	#FFF5F5DC
Bisque	#FFFFE4C4
Black	#FF000000
BlanchedAlmond	#FFFFEBCD
Blue	#FF0000FF
BlueViolet	#FF8A2BE2
Brown	#FFA52A2A
BurlyWood	#FFDEB887
CadetBlue	#FF5F9EA0
Chartreuse	#FF7FFF00
Chocolate	#FFD2691E
Coral	#FFFF7F50
CornflowerBlue	#FF6495ED
Cornsilk	#FFFFF8DC
Crimson	#FFDC143C
Cyan	#FF00FFFF
DarkBlue	#FF00008B
DarkCyan	#FF008B8B

DarkGoldenrod	#FFB8860B
DarkGray	#FFA9A9A9
DarkGreen	#FF006400
DarkKhaki	#FFBDB76B
DarkMagenta	#FF8B008B
DarkOliveGreen	#FF556B2F
DarkOrange	#FFFF8C00
DarkOrchid	#FF9932CC
DarkRed	#FF8B0000
DarkSalmon	#FFE9967A
DarkSeaGreen	#FF8FBC8B
DarkSlateBlue	#FF483D8B
DarkSlateGray	#FF2F4F4F
DarkTurquoise	#FF00CED1
DarkViolet	#FF9400D3
DeepPink	#FFFF1493
DeepSkyBlue	#FF00BFFF
DimGray	#FF696969
DodgerBlue	#FF1E90FF
Firebrick	#FFB22222

FloralWhite	#FFFFFAF0
ForestGreen	#FF228B22
Fuchsia	#FFFF00FF
Gainsboro	#FFDCDCDC
GhostWhite	#FFF8F8FF
Gold	#FFFFD700
Goldenrod	#FFDAA520
Gray	#FF808080
Green	#FF008000
GreenYellow	#FFADFF2F
Honeydew	#FFF0FFF0
HotPink	#FFFF69B4
IndianRed	#FFCD5C5C
Indigo	#FF4B0082
Ivory	#FFFFFFF0
Khaki	#FFF0E68C
Lavender	#FFE6E6FA
LavenderBlush	#FFFFF0F5
LawnGreen	#FF7CFC00
LemonChiffon	#FFFFFACD

LightBlue	#FFADD8E6
LightCoral	#FFF08080
LightCyan	#FFE0FFFF
LightGoldenrodYellow	#FFFAFAD2
LightGray	#FFD3D3D3
LightGreen	#FF90EE90
LightPink	#FFFFB6C1
LightSalmon	#FFFFA07A
LightSeaGreen	#FF20B2AA
LightSkyBlue	#FF87CEFA
LightSlateGray	#FF778899
LightSteelBlue	#FFB0C4DE
LightYellow	#FFFFFFE0
Lime	#FF00FF00
LimeGreen	#FF32CD32
Linen	#FFFAF0E6
Magenta	#FFFF00FF
Maroon	#FF800000
MediumAquamarine	#FF66CDAA
MediumBlue	#FF0000CD

MediumOrchid	#FFBA55D3
MediumPurple	#FF9370DB
MediumSeaGreen	#FF3CB371
MediumSlateBlue	#FF7B68EE
MediumSpringGreen	#FF00FA9A
MediumTurquoise	#FF48D1CC
MediumVioletRed	#FFC71585
MidnightBlue	#FF191970
MintCream	#FFF5FFFA
MistyRose	#FFFFE4E1
Moccasin	#FFFFE4B5
NavajoWhite	#FFFFDEAD
Navy	#FF000080
OldLace	#FFFDF5E6
Olive	#FF808000
OliveDrab	#FF6B8E23
Orange	#FFFFA500
OrangeRed	#FFFF4500
Orchid	#FFDA70D6
PaleGoldenrod	#FFEEE8AA

PaleGreen	#FF98FB98
PaleTurquoise	#FFAFEEEE
PaleVioletRed	#FFDB7093
PapayaWhip	#FFFFEFD5
PeachPuff	#FFFFDAB9
Peru	#FFCD853F
Pink	#FFFFC0CB
Plum	#FFB0E0E6
Purple	#FF800080
Red	#FFFF0000
RosyBrown	#FFBC8F8F
RoyalBlue	#FF4169E1
SaddleBrown	#FF8B4513
Salmon	#FFFA8072
SandyBrown	#FFF4A460
SeaGreen	#FF2E8B57
SeaShell	#FFFFF5EE
Sienna	#FFA0522D
Silver	#FFC0C0C0
SkyBlue	#FF87CEEB

SlateBlue	#FF6A5ACD
SlateGray	#FF708090
Snow	#FFFFFAFA
SpringGreen	#FF00FF7F
SteelBlue	#FF4682B4
Tan	#FFD2B48C
Teal	#FF008080
Thistle	#FFD8BFD8
Tomato	#FFFF6347
Turquoise	#FF40E0D0
Violet	#FFEE82EE
Wheat	#FFF5DEB3
White	#FFFFFFFF
WhiteSmoke	#FFF5F5F5
Yellow	#FFFFFF00
YellowGreen	#FF9ACD32

layer	**Possible values** OdinPageLayer.Foreground OdinPageLayer.Background

Pagecount

Returns the number of pages of the current document.

PageCount	
SYNTAX	PageCount()
RETURN	Number of pages.

Post

Writes character commands in the document layer.

Post	
SYNTAX	Post()

Rectangle

Draws a rectangle on the current page.

Rectangle	
SYNTAX	Rectangle(x, y, witdh, height, fill, fillcolor) Rectangle(x, y, width, height, lineWidth, lineStyle, lineColor, fill, fillColor) Rectangle(x, y, width, height, lineWidth, lineStyle, lineColor, fill, fillColor, layer)
ARGUMENTS	x — X-position of the top left corner of the rectangle. y — Y-position of the top left corner of the rectangle. If the string list is empty, all fields are read. The string list must exist. width — Width of the rectangle. height — Height of the rectangle. fill — TRUE = The rectangle is filled. FALSE = The rectangle is not filled.

The default value is FALSE.

fillColor

Pad color.

Possible value

The HTML color name or the ARGB value in format #AARRGGBB (AA=Alpha-Transparency, RR=Red, GG=Green, BB=Blue).

HTML color name	ARGB value
AliceBlue	#FFF0F8FF
AntiqueWhite	#FFFAEBD7
Aqua	#FF00FFFF
Aquamarine	#FF7FFFD4
Azure	#FFF0FFFF
Beige	#FFF5F5DC
Bisque	#FFFFE4C4
Black	#FF000000
BlanchedAlmond	#FFFFEBCD
Blue	#FF0000FF
BlueViolet	#FF8A2BE2
Brown	#FFA52A2A
BurlyWood	#FFDEB887
CadetBlue	#FF5F9EA0
Chartreuse	#FF7FFF00

Chocolate	#FFD2691E
Coral	#FFFF7F50
CornflowerBlue	#FF6495ED
Cornsilk	#FFFFF8DC
Crimson	#FFDC143C
Cyan	#FF00FFFF
DarkBlue	#FF00008B
DarkCyan	#FF008B8B
DarkGoldenrod	#FFB8860B
DarkGray	#FFA9A9A9
DarkGreen	#FF006400
DarkKhaki	#FFBDB76B
DarkMagenta	#FF8B008B
DarkOliveGreen	#FF556B2F
DarkOrange	#FFFF8C00
DarkOrchid	#FF9932CC
DarkRed	#FF8B0000
DarkSalmon	#FFE9967A
DarkSeaGreen	#FF8FBC8B
DarkSlateBlue	#FF483D8B

DarkSlateGray	#FF2F4F4F
DarkTurquoise	#FF00CED1
DarkViolet	#FF9400D3
DeepPink	#FFFF1493
DeepSkyBlue	#FF00BFFF
DimGray	#FF696969
DodgerBlue	#FF1E90FF
Firebrick	#FFB22222
FloralWhite	#FFFFFAF0
ForestGreen	#FF228B22
Fuchsia	#FFFF00FF
Gainsboro	#FFDCDCDC
GhostWhite	#FFF8F8FF
Gold	#FFFFD700
Goldenrod	#FFDAA520
Gray	#FF808080
Green	#FF008000
GreenYellow	#FFADFF2F
Honeydew	#FFF0FFF0
HotPink	#FFFF69B4

IndianRed	#FFCD5C5C
Indigo	#FF4B0082
Ivory	#FFFFFFF0
Khaki	#FFF0E68C
Lavender	#FFE6E6FA
LavenderBlush	#FFFFF0F5
LawnGreen	#FF7CFC00
LemonChiffon	#FFFFFACD
LightBlue	#FFADD8E6
LightCoral	#FFF08080
LightCyan	#FFE0FFFF
LightGoldenrodYellow	#FFFAFAD2
LightGray	#FFD3D3D3
LightGreen	#FF90EE90
LightPink	#FFFFB6C1
LightSalmon	#FFFFA07A
LightSeaGreen	#FF20B2AA
LightSkyBlue	#FF87CEFA
LightSlateGray	#FF778899
LightSteelBlue	#FFB0C4DE

LightYellow	#FFFFFFE0
Lime	#FF00FF00
LimeGreen	#FF32CD32
Linen	#FFFAF0E6
Magenta	#FFFF00FF
Maroon	#FF800000
MediumAquamarine	#FF66CDAA
MediumBlue	#FF0000CD
MediumOrchid	#FFBA55D3
MediumPurple	#FF9370DB
MediumSeaGreen	#FF3CB371
MediumSlateBlue	#FF7B68EE
MediumSpringGreen	#FF00FA9A
MediumTurquoise	#FF48D1CC
MediumVioletRed	#FFC71585
MidnightBlue	#FF191970
MintCream	#FFF5FFFA
MistyRose	#FFFFE4E1
Moccasin	#FFFFE4B5
NavajoWhite	#FFFFDEAD

Navy	#FF000080
OldLace	#FFFDF5E6
Olive	#FF808000
OliveDrab	#FF6B8E23
Orange	#FFFFA500
OrangeRed	#FFFF4500
Orchid	#FFDA70D6
PaleGoldenrod	#FFEEE8AA
PaleGreen	#FF98FB98
PaleTurquoise	#FFAFEEEE
PaleVioletRed	#FFDB7093
PapayaWhip	#FFFFEFD5
PeachPuff	#FFFFDAB9
Peru	#FFCD853F
Pink	#FFFFC0CB
Plum	#FFB0E0E6
Purple	#FF800080
Red	#FFFF0000
RosyBrown	#FFBC8F8F
RoyalBlue	#FF4169E1

SaddleBrown	#FF8B4513
Salmon	#FFFA8072
SandyBrown	#FFF4A460
SeaGreen	#FF2E8B57
SeaShell	#FFFFF5EE
Sienna	#FFA0522D
Silver	#FFC0C0C0
SkyBlue	#FF87CEEB
SlateBlue	#FF6A5ACD
SlateGray	#FF708090
Snow	#FFFFFAFA
SpringGreen	#FF00FF7F
SteelBlue	#FF4682B4
Tan	#FFD2B48C
Teal	#FF008080
Thistle	#FFD8BFD8
Tomato	#FFFF6347
Turquoise	#FF40E0D0
Violet	#FFEE82EE
Wheat	#FFF5DEB3

White	#FFFFFFFF
WhiteSmoke	#FFF5F5F5
Yellow	#FFFFFF00
YellowGreen	#FF9ACD32

lineWidth — Line width.

lineStyle — Line style.

Possible values

OdinLineStyle.Solid

OdinLineStyle.Dashed

OdinLineStyle.Dotted

lineColor — Line color.

Possible value is the HTML color name or the ARGB value in format #AARRGGBB (AA=Alpha-Transparency, RR=Red, GG=Green, BB=Blue).

HTML color name	ARGB value
AliceBlue	#FFF0F8FF
AntiqueWhite	#FFFAEBD7
Aqua	#FF00FFFF
Aquamarine	#FF7FFFD4
Azure	#FFF0FFFF

Beige	#FFF5F5DC
Bisque	#FFFFE4C4
Black	#FF000000
BlanchedAlmond	#FFFFEBCD
Blue	#FF0000FF
BlueViolet	#FF8A2BE2
Brown	#FFA52A2A
BurlyWood	#FFDEB887
CadetBlue	#FF5F9EA0
Chartreuse	#FF7FFF00
Chocolate	#FFD2691E
Coral	#FFFF7F50
CornflowerBlue	#FF6495ED
Cornsilk	#FFFFF8DC
Crimson	#FFDC143C
Cyan	#FF00FFFF
DarkBlue	#FF00008B
DarkCyan	#FF008B8B
DarkGoldenrod	#FFB8860B
DarkGray	#FFA9A9A9

DarkGreen	#FF006400
DarkKhaki	#FFBDB76B
DarkMagenta	#FF8B008B
DarkOliveGreen	#FF556B2F
DarkOrange	#FFFF8C00
DarkOrchid	#FF9932CC
DarkRed	#FF8B0000
DarkSalmon	#FFE9967A
DarkSeaGreen	#FF8FBC8B
DarkSlateBlue	#FF483D8B
DarkSlateGray	#FF2F4F4F
DarkTurquoise	#FF00CED1
DarkViolet	#FF9400D3
DeepPink	#FFFF1493
DeepSkyBlue	#FF00BFFF
DimGray	#FF696969
DodgerBlue	#FF1E90FF
Firebrick	#FFB22222
FloralWhite	#FFFFFAF0
ForestGreen	#FF228B22

Fuchsia	#FFFF00FF
Gainsboro	#FFDCDCDC
GhostWhite	#FFF8F8FF
Gold	#FFFFD700
Goldenrod	#FFDAA520
Gray	#FF808080
Green	#FF008000
GreenYellow	#FFADFF2F
Honeydew	#FFF0FFF0
HotPink	#FFFF69B4
IndianRed	#FFCD5C5C
Indigo	#FF4B0082
Ivory	#FFFFFFF0
Khaki	#FFF0E68C
Lavender	#FFE6E6FA
LavenderBlush	#FFFFF0F5
LawnGreen	#FF7CFC00
LemonChiffon	#FFFFFACD
LightBlue	#FFADD8E6
LightCoral	#FFF08080

LightCyan	#FFE0FFFF
LightGoldenrodYellow	#FFFAFAD2
LightGray	#FFD3D3D3
LightGreen	#FF90EE90
LightPink	#FFFFB6C1
LightSalmon	#FFFFA07A
LightSeaGreen	#FF20B2AA
LightSkyBlue	#FF87CEFA
LightSlateGray	#FF778899
LightSteelBlue	#FFB0C4DE
LightYellow	#FFFFFFE0
Lime	#FF00FF00
LimeGreen	#FF32CD32
Linen	#FFFAF0E6
Magenta	#FFFF00FF
Maroon	#FF800000
MediumAquamarine	#FF66CDAA
MediumBlue	#FF0000CD
MediumOrchid	#FFBA55D3
MediumPurple	#FF9370DB

MediumSeaGreen	#FF3CB371
MediumSlateBlue	#FF7B68EE
MediumSpringGreen	#FF00FA9A
MediumTurquoise	#FF48D1CC
MediumVioletRed	#FFC71585
MidnightBlue	#FF191970
MintCream	#FFF5FFFA
MistyRose	#FFFFE4E1
Moccasin	#FFFFE4B5
NavajoWhite	#FFFFDEAD
Navy	#FF000080
OldLace	#FFFDF5E6
Olive	#FF808000
OliveDrab	#FF6B8E23
Orange	#FFFFA500
OrangeRed	#FFFF4500
Orchid	#FFDA70D6
PaleGoldenrod	#FFEEE8AA
PaleGreen	#FF98FB98
PaleTurquoise	#FFAFEEEE

PaleVioletRed	#FFDB7093
PapayaWhip	#FFFFEFD5
PeachPuff	#FFFFDAB9
Peru	#FFCD853F
Pink	#FFFFC0CB
Plum	#FFB0E0E6
Purple	#FF800080
Red	#FFFF0000
RosyBrown	#FFBC8F8F
RoyalBlue	#FF4169E1
SaddleBrown	#FF8B4513
Salmon	#FFFA8072
SandyBrown	#FFF4A460
SeaGreen	#FF2E8B57
SeaShell	#FFFFF5EE
Sienna	#FFA0522D
Silver	#FFC0C0C0
SkyBlue	#FF87CEEB
SlateBlue	#FF6A5ACD
SlateGray	#FF708090

Snow	#FFFFFAFA
SpringGreen	#FF00FF7F
SteelBlue	#FF4682B4
Tan	#FFD2B48C
Teal	#FF008080
Thistle	#FFD8BFD8
Tomato	#FFFF6347
Turquoise	#FF40E0D0
Violet	#FFEE82EE
Wheat	#FFF5DEB3
White	#FFFFFFFF
WhiteSmoke	#FFF5F5F5
Yellow	#FFFFFF00
YellowGreen	#FF9ACD32

layer **Possible values**

OdinPageLayer.Foreground = Drawing is done on top of the original document.

OdinPageLayer.Background = Drawing is done behind the original document.

SetAfpCopyGroup

Set the copy group of a page.

SetAfpCopyGroup

SYNTAX	SetAfpCopyGroup(copyGroup)
ARGUMENTS	copyGroup — Copy group to set
EXAMPLE	; set the CopyGroup "CG1" on first page of each document streamdoc.EditPage(0) streamdoc.SetAfpCopyGroup("CG1") streamdoc.Post()

SetFont

Defines the font specification for text output.

SetFont

SYNTAX	SetFont(fontName, fontSize)
ARGUMENTS	fontName — Name of the font, for example, Arial. fontSize — Size of the font, for example, 10.

SetInputBin

Sets the input tray for a page. The function overwrites the predefined tray for the first page or for the follow-up as the case may be.

SetInputBin

SYNTAX	SetInputBin(pageNumber, binName)
ARGUMENTS	pageNumber — Page number. binName — Name of the tray.
EXAMPLE	;Set the input tray for the last page of the document streamDoc.SetInputBin(streamdoc.PageCount - 1, "testBin")

TextOut

Writes a text on the current page.

CreateGuid

SYNTAX

TextOut(x, y, text, bold, italic)

TextOut(x, y, text, bold, italic, color, angle)

TextOut(x, y, text, bold, italic, color, angle, layer)

ARGUMENTS

x	X-position of the starting point
y	Y-position of the starting point
text	Text to write
bold	TRUE = Formatting bold FALSE = Formatting not bold The default value is FALSE
italic	TRUE = Formatting italic FALSE = Formatting not italic The default value is FALSE
color	Color Possible value is the HTML color name or the ARGB value in format #AARRGGBB (AA=Alpha-Transparency, RR=Red, GG=Green, BB=Blue).

HTML color name	ARGB value
AliceBlue	#FFF0F8FF
AntiqueWhite	#FFFAEBD7

Aqua	#FF00FFFF
Aquamarine	#FF7FFFD4
Azure	#FFF0FFFF
Beige	#FFF5F5DC
Bisque	#FFFFE4C4
Black	#FF000000
BlanchedAlmond	#FFFFEBCD
Blue	#FF0000FF
BlueViolet	#FF8A2BE2
Brown	#FFA52A2A
BurlyWood	#FFDEB887
CadetBlue	#FF5F9EA0
Chartreuse	#FF7FFF00
Chocolate	#FFD2691E
Coral	#FFFF7F50
CornflowerBlue	#FF6495ED
Cornsilk	#FFFFF8DC
Crimson	#FFDC143C
Cyan	#FF00FFFF
DarkBlue	#FF00008B

DarkCyan	#FF008B8B
DarkGoldenrod	#FFB8860B
DarkGray	#FFA9A9A9
DarkGreen	#FF006400
DarkKhaki	#FFBDB76B
DarkMagenta	#FF8B008B
DarkOliveGreen	#FF556B2F
DarkOrange	#FFFF8C00
DarkOrchid	#FF9932CC
DarkRed	#FF8B0000
DarkSalmon	#FFE9967A
DarkSeaGreen	#FF8FBC8B
DarkSlateBlue	#FF483D8B
DarkSlateGray	#FF2F4F4F
DarkTurquoise	#FF00CED1
DarkViolet	#FF9400D3
DeepPink	#FFFF1493
DeepSkyBlue	#FF00BFFF
DimGray	#FF696969
DodgerBlue	#FF1E90FF

Firebrick	#FFB22222
FloralWhite	#FFFFFAF0
ForestGreen	#FF228B22
Fuchsia	#FFFF00FF
Gainsboro	#FFDCDCDC
GhostWhite	#FFF8F8FF
Gold	#FFFFD700
Goldenrod	#FFDAA520
Gray	#FF808080
Green	#FF008000
GreenYellow	#FFADFF2F
Honeydew	#FFF0FFF0
HotPink	#FFFF69B4
IndianRed	#FFCD5C5C
Indigo	#FF4B0082
Ivory	#FFFFFFF0
Khaki	#FFF0E68C
Lavender	#FFE6E6FA
LavenderBlush	#FFFFF0F5
LawnGreen	#FF7CFC00

LemonChiffon	#FFFFFACD
LightBlue	#FFADD8E6
LightCoral	#FFF08080
LightCyan	#FFE0FFFF
LightGoldenrodYellow	#FFFAFAD2
LightGray	#FFD3D3D3
LightGreen	#FF90EE90
LightPink	#FFFFB6C1
LightSalmon	#FFFFA07A
LightSeaGreen	#FF20B2AA
LightSkyBlue	#FF87CEFA
LightSlateGray	#FF778899
LightSteelBlue	#FFB0C4DE
LightYellow	#FFFFFFE0
Lime	#FF00FF00
LimeGreen	#FF32CD32
Linen	#FFFAF0E6
Magenta	#FFFF00FF
Maroon	#FF800000
MediumAquamarine	#FF66CDAA

MediumBlue	#FF0000CD
MediumOrchid	#FFBA55D3
MediumPurple	#FF9370DB
MediumSeaGreen	#FF3CB371
MediumSlateBlue	#FF7B68EE
MediumSpringGreen	#FF00FA9A
MediumTurquoise	#FF48D1CC
MediumVioletRed	#FFC71585
MidnightBlue	#FF191970
MintCream	#FFF5FFFA
MistyRose	#FFFFE4E1
Moccasin	#FFFFE4B5
NavajoWhite	#FFFFDEAD
Navy	#FF000080
OldLace	#FFFDF5E6
Olive	#FF808000
OliveDrab	#FF6B8E23
Orange	#FFFFA500
OrangeRed	#FFFF4500
Orchid	#FFDA70D6

PaleGoldenrod	#FFEEE8AA
PaleGreen	#FF98FB98
PaleTurquoise	#FFAFEEEE
PaleVioletRed	#FFDB7093
PapayaWhip	#FFFFEFD5
PeachPuff	#FFFFDAB9
Peru	#FFCD853F
Pink	#FFFFC0CB
Plum	#FFB0E0E6
Purple	#FF800080
Red	#FFFF0000
RosyBrown	#FFBC8F8F
RoyalBlue	#FF4169E1
SaddleBrown	#FF8B4513
Salmon	#FFFA8072
SandyBrown	#FFF4A460
SeaGreen	#FF2E8B57
SeaShell	#FFFFF5EE
Sienna	#FFA0522D
Silver	#FFC0C0C0

SkyBlue	#FF87CEEB
SlateBlue	#FF6A5ACD
SlateGray	#FF708090
Snow	#FFFFFAFA
SpringGreen	#FF00FF7F
SteelBlue	#FF4682B4
Tan	#FFD2B48C
Teal	#FF008080
Thistle	#FFD8BFD8
Tomato	#FFFF6347
Turquoise	#FF40E0D0
Violet	#FFEE82EE
Wheat	#FFF5DEB3
White	#FFFFFFFF
WhiteSmoke	#FFF5F5F5
Yellow	#FFFFFF00
YellowGreen	#FF9ACD32

angle	Angle of rotation

layer	**Possible values** OdinPageLayer.Foreground = Drawing is done on top of the original document. OdinPageLayer.Background = Drawing is done behind the original document.

Context OdinStreamPrint

Properties of the Context OdinStreamPrint

Show All

Parameter	Description
CurrentDocument	Current document, if found. Read-only property. **Functions** GetDocumentFileName Returns the name of the original document.

GetDocumentFileName	
SYNTAX	GetDocumentFileName()
RETURN	File name.

GetDuplex

Returns the duplex mode for this document.

GetDuplex	
SYNTAX	GetDuplex()
RETURN	**Enumerator with the following functions.** OdinDuplexSetting.Simplex = No duplex.

	OdinDuplexSetting.Horizontal = Duplex on the short edge. OdinDuplexSetting.Vertical = Duplex on the long edge.

GetFileName

Returns the rendition file name of the document.

GetFileName	
SYNTAX	GetFileName()
RETURN	Rendition file name.

GetFirstInputBin

Returns the tray for the first page of the document.

GetFirstInputBin	
SYNTAX	GetFirstInputBin()
RETURN	Name of the tray.

GetId

Returns the ID of the document (= Job-ID or StackCoverPage / EnvelopeCoverPage, as the case may be)

GetId	
SYNTAX	GetId()
RETURN	Document ID.

GetNextInputBin

Returns the follow on tray for the 2-n pages of a document.

GetNextInputBin

SYNTAX	GetNextInputBin()
RETURN	Next input bin.

GetPageCount

Returns the number of pages.

GetPageCount	
SYNTAX	GetPageCount()
RETURN	Number of pages

IsPortrait

Returns the portrait mode.

IsPortrait		
SYNTAX	IsPortrait(int pageNumber)	
ARGUMENTS	pageNumber	Page number
RETURN	TRUE = Portrait FALSE = Landscape	

SetDuplex

Specifies the duplex mode for this document.

SetDuplex		
SYNTAX	SetDuplex(duplexMode)	
ARGUMENTS	duplexMode	OdinDuplexSetting.Simplex = No duplex

	OdinDuplexSetting.Horizontal = Duplex on the short edge OdinDuplexSetting.Vertical = Duplex on the long edge

SetFirstInputBin

Sets the tray for the first page of the document.

SetFirstInputBin		
SYNTAX	SetFirstInputBin(string inputBin)	
ARGUMENTS	inputBin	Tray

SetNextInputBin

Set the follow on tray for the 2-n pages of a document.

SetNextInputBin		
SYNTAX	SetNextInputBin(string inputBin)	
ARGUMENTS	inputBin	Tray

HasDocument	TRUE = A document is available. FALSE = No document is available. Read-only property.
CurrentEnvelope	Current envelope, if found. Read-only property.

Functions

GetValue

Returns the value of the given column.

GetValue

SYNTAX	GetValue(string columnName)
ARGUMENTS	columnName — Column name
RETURN	Column value If the data type of the database column is DATE, DATETIME or any valid date type, the date read generally returns in the format MM/DD/YYYY hh:mm:ss.
EXAMPLE	GetObject("DateVar","MLDate") … StringDateVar = CurrentEnvelope.GetValue("ODE_CREATE_AT") ;Convert date to the required format with the help of the MLDate object IntDateVar = DateVar.DateStringToInt64("MM/dd/yyyy hh:mm:ss", StringDateVar) v_LETTERDATE= DateVar.Int64ToDateString("MM.dd.yyyy hh:mm:ss", IntDateVar)

GetFieldNameEnumerator

Returns an enumerator.

GetFieldNameEnumerator

SYNTAX	GetFieldNameEnumerator()
RETURN	Enumerator with the following methods.

Method	Description
Current	Returns the current object.

MoveNext	Moves the position to the next object to be processed.
Reset	Resets the enumerator to the beginning of the first element.

CurrentJob Current job, if found.

Read-only property.

Functions

GetValue

Returns the value of the given column.

GetValue

SYNTAX GetValue(string columnName)

ARGUMENTS

columnName	Column name

RETURN Column value

If the data type of the database column is DATE, DATETIME or any valid date type, the date read generally returns in the format MM/DD/YYYY hh:mm:ss.

EXAMPLE

```
GetObject("DateVar","MLDate")
…
StringDateVar = CurrentEnvelope.GetValue("ODE_CREATE_AT")
;Convert date to the required format with the help of the MLDate object
IntDateVar = DateVar.DateStringToInt64("MM/dd/yyyy hh:mm:ss", StringDateVar )
v_LETTERDATE= DateVar.Int64ToDateString("MM.dd.yyyy hh:mm:ss", IntDateVar)
```

GetFieldNameEnumerator

Returns an enumerator.

GetFieldNameEnumerator

SYNTAX GetFieldNameEnumerator()

RETURN Enumerator with the following methods.

Method	Description
Current	Returns the current object.
MoveNext	Moves the position to the next object to be processed.
Reset	Resets the enumerator to the beginning of the first element.

CurrentStack

Current stack

Read-only property.

Functions

GetValue

Returns the value of the given column.

GetValue

SYNTAX GetValue(string columnName)

ARGUMENTS

columnName	Column name

RETURN Column value

If the data type of the database column is DATE, DATETIME or any valid date type, the date read generally returns in the format MM/DD/YYYY hh:mm:ss.

EXAMPLE GetObject("DateVar","MLDate")

…

```
StringDateVar = CurrentEnvelope.GetValue("ODE_CREATE_AT")

;Convert date to the required format with the help of the MLDate object

IntDateVar = DateVar.DateStringToInt64("MM/dd/yyyy hh:mm:ss",
StringDateVar )

v_LETTERDATE= DateVar.Int64ToDateString("MM.dd.yyyy hh:mm:ss
", IntDateVar)
```

GetFieldNameEnumerator

Returns an enumerator.

GetFieldNameEnumerator

SYNTAX GetFieldNameEnumerator()

RETURN Enumerator with the following methods.

Method	Description
Current	Returns the current object.
MoveNext	Moves the position to the next object to be processed.
Reset	Resets the enumerator to the beginning of the first element.

Functions of the Context OdinStreamPrint

HasData

Checks whether data exists for the given type.

HasData

SYNTAX HasData(OdinDataRowType dataType)

ARGUMENTS	dataType	**Possible values** OdinDataRowType.Process OdinDataRowType.Stack OdinDataRowType.Envelope OdinDataRowType.Job OdinDataRowType.VirtualJob
RETURN	TRUE = The data row specified exists. FALSE = The data row specified does not exist.	

Context OnError

Properties of the Context OnError

Property	Description
LastErrorMessage	Last error message

Context OnNavigateObject

Properties of the Context OnNavigateObject

Property	Description
Block	Allows you to access the text block properties.
CancelIteration	TRUE = Cancel the navigation. FALSE = Do not cancel the navigation.
ComponentType	Object type **Possible values** TextSystemObjectType.DocumentReference TextSystemObjectType.Block

	TextSystemObjectType.Variable
DocumentReference	Allows you to access the document reference properties.
SkipChildren	TRUE = Skip child objects. FALSE = Do not skip child objects.
Variable	Allows you to access the variable properties.

Sample Script OnNavigateObject

The following script checks which objects were activated manually by a user and writes these in the variable **System.UsedBlocks**.

```
;Standard script to filter UsedBlocks according to standard modus4 logic
;
if (ComponentType = TextSystemObjectType.Block)
;
  ;This IF-query checks if the rule for the Content Composer 18.0 filter is met.
  ;Is the text block Kann (BlockType = 1)
  ;Is the text block visible (also takes account of parent text blocks (IsVisible)
  ;Was the block inserted? (IsInserted)
  if ((Block.BlockType = "1") and (Block.IsVisible = true) and (Block.IsInserted = true))
    ;
    if (System.UsedBlocks = "")
      System.UsedBlocks = Block.ObjectId
    else
      System.UsedBlocks = System.UsedBlocks + ", " + Block.ObjectId
    end-if
    ;
  end-if
  ;
  ;If the text block was not inserted, then its children do not need to be checked
```

```
  if ((Block.IsInserted = false) or (Block.IsVisible = false))
    SkipChildren = true
  end-if
  ;
end-if
```

Context OnProcessError

Properties of the Context OnProcessError

Show All

Property	Description
LastError Message	Last error message
ModusProcess	Provides the following properties and functions. **Properties** Objectname: Name of the process. SystemOId: ID of the system in which the process runs. **Functions** Cancel Cancels the process. Cancel SYNTAX Cancel() EXAMPLE `ModusProcess.Cancel()` GetComponent Enables access to the properties and methods of components. GetComponent

SYNTAX	GetComponent(string componentName)
ARGUMENTS	componentName — Name of the component
EXAMPLE	DocumentCollection = ModusProcess.GetComponent ("DocumentCollectionManager")

SendInfoMessage

Sends an information message to logged on remote control clients. A time stamp is inserted before each message.

SendInfoMessage

SYNTAX	SendInfoMessage(string message)
ARGUMENTS	message — The information message you want to send.
EXAMPLE	ModusProcess.SendInfoMessage("This is the message")

Stop

Ends running processes.

Stop

SYNTAX	Stop()
EXAMPLE	`ModusProcess.Stop()`

ResetError — Allows you to reset an error state in an **OnError** script.

TRUE = Prevents canceling a process in the event of an error.

FALSE = Allows canceling a process in the event of an error.

If the activities run in a timer, the current process reruns at the next timer interval. Otherwise, the process terminates.

ResumeOptions — **Possible values**

ProcessResumeOptions.NextActivity: The process resumes with the next activity.

ProcessResumeOptions.FromStart: The process starts from the beginning.

ProcessResumeOptions.FromStartWithIntervall: The process starts from the beginning at the next interval.

Notes

- Variable pools are not reset.
- Setting **ResumeOptions** only applies when the process runs in a timer.

Context Process

About the Context Process

Use this context for scripts in processes.

Properties of the Context Process

Show All

Property	Description
ModusProcess	Provides the following properties and functions. **Properties** Objectname: Name of the process. SystemOId: ID of the system in which the process runs. **Functions** Cancel Cancels the process.

Cancel	
SYNTAX	Cancel()
EXAMPLE	`ModusProcess.Cancel()`

GetComponent

Enables access to the properties and methods of components.

GetComponent	
SYNTAX	GetComponent(string componentName)
ARGUMENTS	componentName — Name of the component
EXAMPLE	DocumentCollection = ModusProcess.GetComponent ("DocumentCollectionManager")

SendInfoMessage

Sends an information message to logged on remote control clients. A time stamp is inserted before each message.

SendInfoMessage	
SYNTAX	SendInfoMessage(string message)
ARGUMENTS	message — The information message you want to send.
EXAMPLE	ModusProcess.SendInfoMessage("This is the message")

Stop

Ends running processes.

Stop	
SYNTAX	Stop()
EXAMPLE	`ModusProcess.Stop()`

Context ProcessTimer

About the Context ProcessTimer

Use this context for timer scripts.

Properties of the Context ProcessTimer

Show All

Property	Description
Execute Timer	TRUE = Execute timer logic. FALSE = Do not execute timer logic.
Interval	Interval in milliseconds the timer service triggers the process timer.
ModusProcess	Provides the following properties and functions. **Properties** Objectname: Name of the process SystemOId: ID of the system in which the process runs. **Functions** Cancel Cancels the process. Cancel SYNTAX Cancel() EXAMPLE `ModusProcess.Cancel()` GetComponent Enables access to the properties and methods of components. GetComponent

SYNTAX	GetComponent(string componentName)
ARGUMENTS	componentName Name of the component
EXAMPLE	DocumentCollection = ModusProcess.GetComponent ("DocumentCollectionManager")

SendInfoMessage

Sends an information message to logged on remote control clients. A time stamp is inserted before each message.

SendInfoMessage

SYNTAX	SendInfoMessage(string message)
ARGUMENTS	message The information message you want to send.
EXAMPLE	ModusProcess.SendInfoMessage("This is the message")

Stop

Ends running processes.

Stop

SYNTAX	Stop()
EXAMPLE	`ModusProcess.Stop()`

Context SelectionQueryScript

Properties of the Context SelectionQueryScript

Property	Description
Db	Database alias

Context TextComponent

About the Context TextComponent

Use this context for scripts assigned to text blocks.

Properties of the Context TextComponent

The name in brackets is the name of the property in Modus 4, which for reasons of maintaining downwards compatibility, may also be used.

Property	Description
BlockprotectionS (Blockprotection)	Corresponds to the property TextComponentProtection for a text block. For more information, refer to Text Block Properties.
BlockType (BlockType)	0 = Deactivated 1 = Required 2 = Activated 3 = Disabled
DataLink (Data Object)	Path to the data object. Read-only property.
Expandable	TRUE = The user can see the text block structure and is allowed to activate or deactivate its objects. FALSE = The user cannot see the text block structure.
ExpandableS	Y = The user can see the text block structure and is allowed to activate or deactivate its objects. N = The user cannot see the text block structure.
IsVisible	Determines by the property of the parent text block or document. TRUE = Visible FALSE = Invisible Read-only property.
TextBlockProtection	TRUE = TextBlockProtection is activated. FALSE = TextBlockProtection is not activated.

Title	Name of the text block the user sees in the Content Composer Client.
Visible	TRUE = Visible FALSE = Invisible
VisibleS	Y = Visible N = Invisible
XPathExpression	For more information, refer to Text Block Properties.
ErrorMessage	Error message. If a value is assigned, the generation aborts with the corresponding error message.
InsertObject	TRUE = Insert the object. FALSE = Do not insert the object.
InsertObjectS (InsertObject)	Y = Insert the object. N = Do not insert the object.
IsInserted	TRUE = Object was inserted. FALSE = Object was not inserted. Read-only property.
ObjectId (ObjectIndex)	Object ID Read-only property.
ObjectName	Object name Read-only property.
ParentBlock	Parent text block Read-only property.
ParentDocument	Parent document Read-only property.

Functions of the Context TextComponent

LoadExternalDocument

Loads an existing document and replaces the text block. You can specify the location of the document either as file path or as URL.

Use this function in a **BeforeInsert** script and use as a document a DOCX file created with Microsoft Word 2016 or higher.

The text block using the Word document cannot contain nested text blocks or variables.

LoadExternalDocument		
SYNTAX	LoadExternalDocument(fileSource)	
ARGUMENTS	fileSource	Name and path of the document. **Possible values** • Strings with preceding **http://** or **https://** substring, also in uppercase, are interpreted as URL. **Note** The function supports both Windows authentication and anonymous authentication. • Any other string is interpreted as a file path.
EXAMPLE	LoadExternalDocument("http://fileServer/Documents/How_To.docx") LoadExternalDocument("C:\Documents\How_To.docx")	

LoadExternalDocumentFromBytes

Loads an existing document from a byte array and replaces the text block.

Use this function in a **BeforeInsert** script and use as a document a DOCX file created with Microsoft Word 2016 or higher.

The text block using the Word document cannot contain nested text blocks or variables.

LoadExternalDocumentFromBytes	
SYNTAX	LoadExternalDocumentFromBytes(bytes)

ARGUMENTS	bytes A byte array.
EXAMPLE	LoadExternalDocumentFromBytes(System.IO.File.ReadAllBytes ("c:\Documents\Account.docx"))

Navigate

Enables you to navigate through the child objects of the current object.

Navigate	
SYNTAX	Navigate(onObjectScript, onlyActiveNodes, filterTypes)
ARGUMENTS	onObjectScript: Script that executes for each of the specified object types found. This script must have the context OnNavigateObject assigned. onlyActiveNodes: TRUE = Only navigates active objects. FALSE = Navigates all objects. filterTypes: Filter defining the objects to navigate. **Possible values** TextSystemObjectType.Variable TextSystemObjectType.Block TextSystemObjectType.DocumentReferences
EXAMPLE	Navigate("Scriptname", true, TextSystemObjectType.Block, TextSystemObjectType.Variable)

Context TextDocument

About the Context TextDocument

Use this context for scripts assigned to documents.

Properties of the Context TextDocument

The name in brackets is the name of the property in Modus 4, which may also be used for reasons of maintaining downward compatibility.

Property	Description
CustomIndex	Use this property in connection with dynamic bundles to pass on values to inserted documents. Read-only property.
MissingObjects	Comma-separated string containing the aliases of activated variable objects not found during document generation. An empty string is returned when no missing objects exist.
ErrorMessage	Error message. If a value is assigned, generation is aborted with the corresponding error message.
InsertObject	TRUE = Insert the object. FALSE = Do not insert the object.
InsertObjectS (InsertObject)	Y = Insert the object. N = Do not insert the object.
IsInserted	TRUE = Object was inserted. FALSE = Object was not inserted. Read-only property.
ObjectId (ObjectIndex)	Object ID Read-only property.
Objectname	Object name Read-only property.

Functions of the Context TextDocument

LoadExternalDocument

Loads a document and replaces the static document. You can specify the location of the document either as file path or as URL.

LoadExternalDocument

SYNTAX LoadExternalDocument(fileSource)

ARGUMENTS

fileSource	Name and path of the document. **Possible values** • Strings with preceding **http://** or **https://** substring, also in uppercase, are interpreted as URL. **Note** The function supports both Windows authentication and anonymous authentication. • Any other string is interpreted as a file path.

EXAMPLE LoadExternalDocument("http://fileServer/Documents/How_To.pdf")

LoadExternalDocument("C:\Documents\How_To.pdf")

LoadExternalDocumentFromBytes

Loads a document from a byte array and replaces the static document.

LoadExternalDocumentFromBytes

SYNTAX LoadExternalDocumentFromBytes(bytes)

ARGUMENTS

bytes	A byte array.

EXAMPLE LoadExternalDocumentFromBytes(System.IO.File.ReadAllBytes ("c:\Documents\Account.pdf"))

Navigate

Enables you to navigate through the child objects of the current object.

Navigate

SYNTAX Navigate(onObjectScript, onlyActiveNodes, filterTypes)

ARGUMENTS	onObjectScript	Script that executes for each of the specified object types found. This script must have the context OnNavigateObject assigned.
	onlyActiveNodes	TRUE = Only navigates active objects. FALSE = Navigates all objects.
	filterTypes	Filter defining the objects to navigate. **Possible values** TextSystemObjectType.Variable TextSystemObjectType.Block TextSystemObjectType.DocumentReferences
EXAMPLE	Navigate("Scriptname", true, TextSystemObjectType.Block, TextSystemObjectType.Variable)	

Context TextDocumentCollection

About the Context TextDocumentCollection

Use this context for scripts assigned to bundles.

This context provides the functions needed to insert documents and document copies into a bundle.

Properties of the Context TextDocumentCollection

The name in brackets is the name of the property in Modus 4, which may also be used for reasons of maintaining downward compatibility.

Property	Description
AllDocumentNames	Comma-separated list of the names of all original documents contained in a bundle. Read-only property.
AllDocuments	Comma-separated list of the child IDs of all original documents contained in a bundle.

	Read-only property.
AllowPrint	TRUE = Printing allowed. FALSE = Printing not allowed.
ArchiveData	Archive data. This property is only contained for reasons of compatibility.
BlockBundle	TRUE = Forward the bundle. FALSE = Do not forward the bundle.
Data ID	Returns the DataID assigned to the bundle object (root object in the bundle), or an empty string if no DataID is assigned.
DestinationGroup (DestUserGroup)	Name of the user group the bundle is forwarded to.
DestinationUser (DestUser)	Name of the user the bundle is forwarded to.
DocumentGroupIndex	BundleID Read-only property.
DocumentGroupName	Bundle name Read-only property.
DocumentNames	Comma-separated list of the names of all active documents. Read-only property.
Documents	Comma-separated list of the child IDs of all active documents. Read-only property.
HoldTime	Hold time for direct print stack.
Priority	Priority for direct print stack.
State	Bundle status Read-only property. **Possible values** DocumentCollectionState.None: Initial state

	DocumentCollectionState.DataProcessed: Data assigned and processed (dynamic bundle) DocumentCollectionState.Created: Documents were created. DocumentCollectionState.Printed: Documents were printed.
ErrorMessage	Error message. If a value is assigned, generation is aborted with the corresponding error message.
InsertObject	TRUE = Insert the object. FALSE = Do not insert the object.
InsertObjectS	Y = Insert the object. N = Do not insert the object.
IsInserted	TRUE = Object was inserted. FALSE = Object was not inserted. Read-only property.
ObjectId (ObjectIndex)	Object ID Read-only property.
Objectname	Object name Read-only property.

Functions of the Context TextDocumentCollection

AddDocumentByName

Adds a document to a bundle. The document is referenced by its name in the repository.

AddDocumentByName

SYN TAX
```
AddDocumentByName(string objectName, string childId,
TextInsertionRule insertRule, string beforeInsertName, string
afterInsertName, string conditionName, string defineSortName, string
defineGroupName, string defineArchiveName, int numberOfCopies,
string printerName, string formName, bool changingAllowed,
OdinPrintTime printTime, bool doArchive, string customIndex, string
```

```
documentName, string dataId, string password, bool isVisible, bool
isExpandable, DocumentType documentType, string title, string
customUIName)


AddDocumentByName(string objectName, string childId,
TextInsertionRule insertRule, string beforeInsertName, string
afterInsertName, string conditionName, string defineSortName, string
defineGroupName, string defineArchiveName, int numberOfCopies,
string printerName, string formName, bool changingAllowed,
OdinPrintTime printTime, bool doArchive, string customIndex, string
documentName, string dataId, string password, bool isVisible, bool
isExpandable, DocumentType documentType, string title, string
customUIName, OdinPrinterType printerType)


AddDocumentByName(string objectName, string childId,
TextInsertionRule insertRule, string beforeInsertName, string
afterInsertName, string conditionName, string defineSortName, string
defineGroupName, string defineArchiveName, int numberOfCopies,
string printerName, string formName, bool changingAllowed,
OdinPrintTime printTime, bool doArchive, string customIndex, string
documentName, string dataId, string password, bool isVisible, bool
isExpandable, DocumentType documentType, string title, string
customUIName, OdinPrinterType printerType, bool onBasePassAsSingle,
long OnBaseDocumentTypeId, string onBaseDIPFileConfigurationName,
string onBaseDocumentApiConfigurationName)
```

ARGUMENTS

objectName	Name of the document in the bundle.
childId	Freely definable document ID. This must be unique within the bundle.
insertRule	TextInsertionRule **Available types** • TextInsertionRule.Activated • TextInsertionRule.Deactivated • TextInsertionRule.Required • TextInsertionRule.Disabled

beforeInsertName	Name of the BeforeInsert script.
afterInsertName	Name of the AfterInsert script.
conditionName	Conditions
defineSortName	Name of the DefinitionSorting script.
defineGroupName	Name of the DefinitionGrouping script.
defineArchiveName	Name of the DefinitionArchiving script.
numberOfCopies	Number of copies.
printerName	Printers
formName	Form name.
changingAllowed	TRUE = The user can modify the document.. FALSE = The user cannot modify the document.
printTime	Printing time **Available types** • OdinPrintTime.Direct: Immediately • OdinPrintTime.Shifted: Shifted • OdinPrintTime.Unknown: Unknown • OdinPrintTime.Local: Local
doArchive	TRUE = Archive the document. FALSE = Do not archive the document.
customIndex	Freely definable index, for example, for subsequent queries.

documentName	Name of the document.
dataId	Selection
password	Password
isVisible	TRUE = Visible FALSE = Invisible
isExpandable	TRUE = Expandable FALSE = Not expandable
documentType	Document type **Available types** • DocumentType.Unknown **Note**: If this type is specified, Content Composer determines the correct document type. • DocumentType.DynamicWord • DocumentType.StaticWord • DocumentType.StaticPdf • DocumentType.StaticTiff • DocumentType.StaticXps
title	Title of the document in the bundle. If you set this parameter, it replaces the object name.
customUIName	CustomUI
printerType	Printer type **Available types** • OdinPrinterType.Online • OdinPrinterType.Email The default value is

	OdinPrinterType.Online.
onBasePassAsSingle	Possible values are `TRUE` or `FALSE`. For more information, refer to OnBase Archive - Transfer single file to archive.
OnBaseDocumentTypeId	OnBase Document Type ID (integer) For more information, refer to OnBase Archive - Document Type.
onBaseDIPFileConfigurationName	Name of the OnBase DIP File Configuration object to use. To not use an object, pass an empty string. For more information, refer to OnBase DIP File Configurations.
onBaseDocumentApiConfigurationName	Name of the OnBase Document API Configuration object to use. To not use an object, pass an empty string. For more information, refer to OnBase Document API Configurations.

EXAMPLE

```
AddDocumentByName("DocumentOriginal","ChildID_Original,TextInsertionRule.Activated, BeforeInsertDocument, AfterInsertDocument, True_Document, SortingDocument, GroupingDocument, ArchivingDocument, 1, Printer, Form, true, OdinPrintTime.Shifted, false, CustomIndex, StaticPdf, Partner_V2,Password, true, true, DocumentType.StaticPdf, Title_Original, CustomUIName)
```

AddDocumentByObjectId

Adds a document to a bundle. The document is referenced by its object ID.

AddDocumentByObjectId

SYNTAX

```
AddDocumentByObjectId(string objectName, string childId, TextInsertionRule insertRule, string beforeInsertOID, string afterInsertOID, string conditionOID, string defineSortOID, string defineGroupOID, string defineArchiveOID, int numberOfCopies, string printerOID, string formOID, bool changingAllowed, OdinPrintTime printTime, bool doArchive, string customIndex,
```

string documentOID, string dataId, string password, bool isVisible, bool isExpandable, DocumentType documentType, string title, string customUIOID)

ARGUMENTS

objectOID	Object Id of the document in the bundle.
childId	Freely definable document ID. This must be unique within the bundle.
insertRule	TextInsertionRule **Available types** TextInsertionRule.Activated TextInsertionRule.Deactivated TextInsertionRule.Required TextInsertionRule.Disabled
beforeInsertOID	Object ID of the BeforeInsert script.
afterInsertOID	Object ID of the AfterInsert script.
conditionOID	Object ID of the condition.
defineSortOID	Object ID of the DefinitionSorting script.
defineGroupOID	Object ID of the DefinitionGrouping script.
defineArchiveOID	Object ID of the DefinitionArchiving script.
numberOfCopies	Number of copies.
printerOID	Object ID of the printer.
formOID	Object ID of the form.
changingAllowed	TRUE = The user can modify the document. FALSE = The user cannot modify the document.
printTime	Printing time **Available types**

	OdinPrintTime.Direct: Immediately OdinPrintTime.Shifted : shifted OdinPrintTime.Unknown: Unknown OdinPrintTime.Local: Local
doArchive	TRUE = Archive the document. FALSE = Do not archive the document.
customIndex	Freely definable index, for example, for subsequent queries.
documentOID	Object ID of the document.
dataId	Selection
password	Password
isVisible	TRUE = Visible FALSE = Invisible
isExpandable	TRUE = Expandable FALSE = Not expandable
documentType	Document Type **Available types** DocumentType.Unknown DocumentType.DynamicWord DocumentType.StaticWord DocumentType.StaticPdf DocumentType.StaticTiff DocumentType.StaticXps
title	Title of the document in the bundle. If this parameter is set, it replaces the object name.
customUIName	Object ID of the CustomUI.

EXAMPLE	AddDocumentByObjectId("Document", "ChildID_Original2", TextInsertionRule.Activated, "15072011-226-sc-ts","15072011-227-sc-ts","10072011-186-cd-ts","15072011-230-sc-ts","15072011-229-sc-ts","15072011-228-sc-ts", 0, "04092011-43-pr-ts", "18112011-482-fo-ts", true, OdinPrintTime.Shifted, false, "CustomIndex_2", "14072011-204-td-ts", "Partner_V2", "Password", true, true, DocumentType.StaticPdf, "Titel_Original2", "06092011-67-cui-ts")

AddDocumentCopyByName

Adds a document copy to a bundle. The document is referenced by its name in the repository.

AddDocumentCopyByName

SYNTAX

AddDocumentCopyByName(string objectName, string childId, TextInsertionRule insertRule, string beforeInsertName, string afterInsertName, string conditionName, string defineSortName, string defineGroupName, string defineArchiveName, int numberOfCopies, string printerName, string formName, bool changingAllowed, OdinPrintTime printTime, bool doArchive, string customIndex, string copyLabel, string originalId, bool isVisible, string title)

AddDocumentCopyByName(string objectName, string childId, TextInsertionRule insertRule, string beforeInsertName, string afterInsertName, string conditionName, string defineSortName, string defineGroupName, string defineArchiveName, int numberOfCopies, string printerName, string formName, bool changingAllowed, OdinPrintTime printTime, bool doArchive, string customIndex, string copyLabel, string originalId, bool isVisible, string title, OdinPrinterType printerType)

AddDocumentCopyByName(string objectName, string childId, TextInsertionRule insertRule, string beforeInsertName, string afterInsertName, string conditionName, string defineSortName, string defineGroupName, string defineArchiveName, int numberOfCopies, string printerName, string formName, bool changingAllowed, OdinPrintTime printTime, bool doArchive, string customIndex, string copyLabel, string originalId, bool isVisible, string title, OdinPrinterType printerType, bool onBasePassAsSingle, long OnBaseDocumentTypeId, string onBaseDIPFileConfigurationName, string onBaseDocumentApiConfigurationName)

ARGUMENTS

objectName	Name of the document in the bundle.
childId	Freely definable document ID. This must be unique within the bundle.
insertRule	TextInsertionRule

	Available types TextInsertionRule.Activated TextInsertionRule.Deactivated TextInsertionRule.Required TextInsertionRule.Disabled
beforeInsertName	Name of the BeforeInsert script.
afterInsertName	Name of the AfterInsert script.
conditionName	Name of the condition.
defineSortName	Name of the DefinitionSorting script.
defineGroupName	Name of the DefinitionGrouping script.
defineArchiveName	Name of the DefinitionArchiving script.
numberOfCopies	Number of copies.
printerName	Printers
formName	Form name.
changingAllowed	TRUE = The user can modify the document. FALSE = The user cannot modify the document.
printTime	Printing time. **Available types** OdinPrintTime.Direct: Immediately OdinPrintTime.Shifted: Shifted OdinPrintTime.Unknown: Unknown OdinPrintTime.Local: Local
doArchive	TRUE = Archive the document.

	FALSE = Do not archive the document.
customIndex	Freely definable index, for example, for subsequent queries.
copyLabel	Text on document copy.
originalId	Child ID of the original.
isVisible	TRUE = Visible FALSE = Invisible
title	Title of the document in the bundle. If this parameter is set, it replaces the object name.
printerType	Printer type **Available types** OdinPrinterType.Online: Online printer OdinPrinterType.Email:Email-Printer If the parameter is not set, the printer type OdinPrinterType.Online is used.
onBasePassAsSingle	Possible values are `TRUE` or `FALSE`. For more information, refer to OnBase Archive - Transfer single file to archive.
OnBaseDocumentTypeId	OnBase Document Type ID (integer) For more information, refer to OnBase Archive - Document Type.
onBaseDIPFileConfigurationName	Name of the OnBase DIP File Configuration object to use. To not use an object, pass an empty string. For more information, refer to OnBase DIP

	File Configurations.
onBaseDocumentApiConfigurationName	Name of the OnBase Document API Configuration object to use. To not use an object, pass an empty string. For more information, refer to OnBase Document API Configurations.

EXAMPLE

```
AddDocumentCopyByName("Document_Copy", "ChildID_Copy", TextInsertionRule.Activated, "BeforeInsertDocument", "AfterInsertDocument", "True_Document","SortingDocument", "GroupingDocument", "ArchivingDocument", 1 ,"Printer", "Form", true, OdinPrintTime.Shifted, false, "CustomIndex_K", "CopyLabel", "ChildID_Original", true, "CopyTitle")
```

AddDocumentCopyByObjectID

Adds a document copy to a bundle. The document is referenced by its ID in the repository.

AddDocumentCopyByObjectId

SYNTAX

```
AddDocumentCopyByObjectId(string objectName, string childId, TextInsertionRule insertRule, string beforeInsertOId, string afterInsertOId, string conditionOId, string defineSortOId, string defineGroupOId, string defineArchiveOId, int numberOfCopies, string printerOId, string formOId, bool changingAllowed, OdinPrintTime printTime, bool doArchive, string customIndex, string copyLabel, string originalId, bool isVisible, string title)
```

ARGUMENTS

objectOid	Name of the document in the bundle.
childId	Freely definable document ID. This must be unique within the bundle.
insertRule	TextInsertionRule **Available types** TextInsertionRule.Activated TextInsertionRule.Deactivated TextInsertionRule.Required TextInsertionRule.Disabled

beforeInsertOid	Object ID of the BeforeInsert script.
afterInsertOid	Object ID of the AfterInsert script.
conditionOid	Object ID of the condition.
defineSortOid	Object ID of the DefinitionSorting script.
defineGroupOid	Object ID of the DefinitionGrouping script.
defineArchiveOid	Object ID of the DefinitionArchiving script.
numberOfCopies	Number of copies.
printerOid	Object ID of the printer.
formOid	Object ID of the form.
changingAllowed	TRUE = The user can modify the document. FALSE = The user cannot modify the document.
printTime	Printing time **Available types** OdinPrintTime.Direct: Immediately OdinPrintTime.Shifted: Shifted OdinPrintTime.Unknown: Unknown OdinPrintTime.Local: Local
doArchive	TRUE = Archive the document. FALSE = Do not archive the document.
customIndex	Freely definable index, for example for subsequent queries.
copyLabel	Text on copy.
originalId	Child Id of the original.
isVisible	TRUE = Visible

	FALSE = Invisible
title	Title of the document in the bundle. If this parameter is set, it replaces the object name.

EXAMPLE: AddDocumentCopyByObjectId("Document_Copy","ChildId_Copy",TextInsertionRule.Activated, "15072011-226-sc-ts", "15072011-227-sc-ts", "10072011-186-cd-ts", "15072011-230-sc-ts","15072011-229-sc-ts", "15072011-228-sc-ts", 1, "04092011-43-pr-ts", "18112011-482-fo-ts", true, OdinPrintTime.Shifted, false, "CustomIndexCopy", "CopyLabel", "ChildID_Original", true, "Titlecopy")

Navigate

Enables you to navigate through the child objects of the current object.

Navigate

SYNTAX: Navigate(onObjectScript, onlyActiveNodes, filterTypes)

ARGUMENTS

onObjectScript	Script that executes for each of the specified object types found. This script must have the context OnNavigateObject assigned.
onlyActiveNodes	TRUE = Only navigates active objects. FALSE = Navigates all objects.
filterTypes	Filter defining the objects to navigate. **Possible values** TextSystemObjectType.Variable TextSystemObjectType.Block TextSystemObjectType.DocumentReferences

EXAMPLE: Navigate("Scriptname", true, TextSystemObjectType.Block, TextSystemObjectType.Variable)

SetArchiveAttribute

Sets an attribute for the archiving process.

SetArchiveAttribute		
SYNTAX	SetArchiveAttribute(name, value)	
ARGUMENTS	name	Attribute name
	value	Attribute value
EXAMPLE	Refer to Identical archive attributes for all documents in a bundle.	

SetDataContext

Defines the current data context. Scripts that contain `SelVariable` statements now address this dataset.

If `dataid` returns null or an empty string, the data context is reset. This means that `SelVariable` statements result in an error. An exception is thrown if no data context exists for the specified ID.

SetDataContext		
SYNTAX	SetDataContext(dataId)	
ARGUMENTS	dataId	The ID of the data context to set

Context TextDocumentReference

About the Context TextDocumentReference

Use this context for scripts assigned to documents in a bundle.

Properties of the Context TextDocumentReference

The name in brackets is the name of the property in Modus 4, which may also be used for reasons of maintaining downward compatibility.

Property	Description

CustomIndex	Use this property in connection with dynamic bundles to pass on values to inserted documents.
DoArchive	TRUE = Archive FALSE = Do not archive
DocumentGroupIndex	BundleID Read-only property.
DocumentGroupName	Bundle name Read-only property.
Expandable	TRUE = Expandable FALSE = Not expandable
ExpandableS	Y = Expandable N = Not expandable
FormName	Reads the name of the form assigned or sets the form by specifying a name.
Grouping	Grouping information
InsertionRule	0 = Deactivated 1 = Required 2 = Activated 3 = Disabled You can use this property in **OnNavigateObject** scripts.
IsActivated	TRUE = The document reference is activated in the bundle. FALSE= The document reference is not activated in the bundle. Read-only property. **Note** IsActivated is true when the **TextInsertionRule** document property is set to Required or Activated.
IsCopy	TRUE = Document is a copy. FALSE = Document is an original. Read-only property.

MissingObjects	Comma-separated string containing the aliases of activated variable objects not found during document generation. An empty string is returned when no missing objects exist.
NumberOfCopies	Number of copies.
OnBaseDocumentTypeId	OnBase document type ID
OnBaseDocumentTypeName	OnBase document type name
PrinterName	Name of the assigned printer. Read-only property.
Sorting	Sorting information
Title	Name of the document the user sees in the Content Composer Client.
Transportpools	Pools whose variables are transferred to the Content Composer Client. Scripts executed in the context **TextDocumentReference** can set variables of **TransportPools** to transfer the variables of the specified pools to the Content Composer Client. The evaluation of variables set in **TransportPools** is carried out in the end processing of a document. The pools are defined by their names in a comma separated list. This can refer to pools that already exist, for example, ModusUser, and to pools created by the MonaLisa command CreatePool. **Example setting in DefinitionArchiving** TransportPools = "MyPool,ModusUser" The transfer of serialized pools is implemented as a sub element of the document structure in the element <archiving>. After end processing, the return structure of the document contains the sub element <archiving>. <object title="Partner_Application" type="D" name="Partner_ Application" index="02062008-823-tdr-itl1" reference="10DE6C09-

	79CF-4547-97AA-6302CF976185" state="1" enabled="1" print_asktime="0" form="TestForm" printer="" printer_type="" printtime="2" print_changeable="1" copycount="0" copy="0" password="composer" filename="Partner_Application_10DE6C09-79CF-4547-97AA-6302CF976185_O.docx"> <archiving> <pools> <pool name="MyPool"> <poolvar name="TESTVALUE" type="S">MyVariableInMyPool</poolvar> </pool> </pools> </archiving> </object>
Visible	TRUE = Visible FALSE = Invisible
VisibleS	Y = Visible N = Invisible
ErrorMessage	Error message. If a value is assigned, the generation aborts with the corresponding error message.
InsertObject	TRUE = Insert the object. FALSE = Do not insert the object.
InsertObjectS (InsertObject)	Y = Insert the object. N = Do not insert the object.
IsInserted	TRUE = Object was inserted. FALSE = Object was not inserted. Read-only property.
ObjectId (ObjectIndex)	Object ID Read-only property.

Objectname	Object name Read-only property.

Functions of the Context TextDocumentReference

DefinePrinterByName

Defines a printer by its name and type.

DefinePrinterByName

SYNTAX DefinePrinterByName(printerName)

ARGUMENTS

printerName	Printer name.
printerType	Printer type.

Possible values	Description
ModusSuiteObjectType.Printer	Online printer.
ModusSuiteObjectType.EMailPrinter	Email printer.
ModusSuiteObjectType.OutsourcingPrinter	Outsourcing printer.

formName	Form name.
allowChange	TRUE = The user can modify the printer. FALSE = The user cannot modify the printer.
printTime	Printing time.

Possible values	Description
OdinPrintTime.Direct	Immediately
OdinPrintTime.Shifted	Shifted
OdinPrintTime.Unknown	Unknown
OdinPrintTime.Local	Local

DefinePrinterByOId

Defines a printer by its object ID.

DefinePrinterByOId

SYNTAX DefinePrinterByOId(printerOId, formOId, allowChange, printTime)

ARGUMENTS

printerOID	Object ID of the printer.
formOID	Object ID of the form.
formName	Form name.
allowChange	TRUE = The user can modify the printer. FALSE = The user cannot modify the printer.
printTime	Printing time.

Possible values	Description

OdinPrintTime.Direct	Immediately
OdinPrintTime.Shifted	Shifted
OdinPrintTime.Unknown	Unknown
OdinPrintTime.Local	Local

SetArchiveAttribute

Sets an attribute for the archiving process.

SetArchiveAttribute	
SYNTAX	SetArchiveAttribute(name, value)
ARGUMENTS	name — Attribute name value — Attribute value
EXAMPLE	Refer to Identical archive attributes for all documents in a bundle.

Context TextVariable

About the Context TextVariable

Use this context for scripts assigned to text variables.

Generation of dynamic charts

Content Composer provides a property editor to predefine dynamic charts. The property editor assigns a specific ChartDefinitionXmlString to the ChartDefinition property of the according text block variable, that must be an image variable.

The TextVariable context provides functions and properties that enable you to define a dynamic chart.

These elements allow you to add data points to existing series (containing a collection of data points) to the predefined chart, which results in painting a chart for the according image variable when the document is generated.

Properties of the Context TextVariable

The name in brackets is the name of the property in Modus 4, which, for reasons of maintaining downward compatibility, may also be used.

Property	Description
AttributeName	Attribute name (database field or pool variable). Read-only property.
ChartDefinition	Contains the ChartDefinitionXmlString provided by the dynamic chart editor.
Content	Content of the variable.
DataLink	Path to the data object or the name of a variable pool. Read-only property.
ErrorMessage	Error message. If a value is assigned, the generation aborts with the corresponding error message.
FormattedContent	Formatted content of the variable. Read-only property.
HasChartDefinition	Checks whether a value is assigned to the ChartDefinition property. TRUE = A value is assigned to property ChartDefinition. Object **Chart** (see below) can be used. FALSE = No value is assigned to property ChartDefinition.
InsertObject	TRUE = Insert the object. FALSE = Do not insert the object.
InsertObjectS (InsertObject)	Y = Insert the object. N = Do not insert the object.
IsFreeInput	Evaluated when a value help is assigned. TRUE = The user can also specify a value manually. FALSE = The user can only select a value from the value help.
IsFreeInputS	Evaluated when a value help is assigned.

(InputEnabled)	Y = The user can also specify a value manually. N = The user can only select a value from the value help.
IsInputRequired	TRUE = The variable is required. FALSE = The variable is not required.
IsInputRequiredS (InputRequired)	Y = The variable is required. N = The variable is not required.
IsInserted	TRUE = The object was inserted. FALSE = The object was not inserted. Read-only property.
IsManual	TRUE = The user can input a value. FALSE = The user cannot input a value.
IsManualS (Input)	Y = The user can input a value. N = The user cannot input a value.
ObjectId (ObjectIndex)	Object ID Read-only property.
Objectname	Object name Read-only property.
ParentBlock	Parent text block Read-only property.
ParentDocument	Parent document Read-only property.
Prompt	Prompt text for manual variables.
XPathExpression	XPath Expression Read-only property.

Object	Description
Chart	Allows you to modify dynamic chart values within a script.

Properties

Property	Description
Chart.Title	Sets or gets the first title of the chart.
Chart.Xml	Returns the deserialized chart object as an XML string.

Functions

AddDataPoint

Adds a data point to the series.

AddDataPoint

SYNTAX AddDataPoint(string seriesName, double xValue, double yValue);

ARGUMENTS

seriesName	Case-sensitive name of the series.
xValue	X-value of the data point.
yValue	Y-value of the data point.

RETURN The new data point.

EXAMPLE dataPoint = Chart.AddDataPoint(

AddDataPointWithLabel

Adds a data point to the series with the provided label. This label overwrites the label of the series for this data point.

AddDataPointWithLabel

SYNTAX	AddDataPointWithLabel(string seriesName, string label, double xValue, double yValue)
ARGUMENTS	seriesName — Case-sensitive name of the series. label — Label of the data point. xValue — X-value of the data point. yValue — Y-value of the data point.
RETURN	Index of data point in the DataPointCollection of the series, or -1 if no point was added.
EXAMPLE	dataPoint = Chart.AddDataPoint("MySeriesName", 0, 10)

AddLegend

Adds a legend to a chart area.

AddLegend

SYNTAX	AddLegend(chartAreaName)
ARGUMENTS	chartAreaName — Name of the chart area assigned to the legend.
RETURN	The added legend.
EXAMPLE	legend = Chart.AddLegend(

AddToSeries

Adds the legend to a given series.

AddToSeries

SYNTAX AddToSeries(string seriesName)

ARGUMENTS

seriesName	Name of the series for which the legend is used.

EXAMPLE legend.AddToSeries("MySeriesName")

ChartAreaExists

Checks whether the chart area with the provided name exists.

ChartAreaExists

SYNTAX ChartAreaExists(string chartAreaName)

ARGUMENTS

chartAreaName	Case-sensitive name of the chart area to check.

RETURN TRUE = The chart area with the provided name exists.

FALSE = The chart area with the provided name does not exist.

EXAMPLE
```
if (Chart.ChartAreaExists(MyChartAreaName))
  ...
end-if
```

CreateDataPoint

Creates a data point and adds it to the specified series. You can use the returned object to specify the properties of the data point.

CreateDataPoint

SYNTAX CreateDataPoint(seriesName)

ARGUMENTS	seriesName	Name of the series to which the data point is created and added.
RETURN	The new data point.	
EXAMPLE	dataPoint = Chart.CreateDataPoint(MySeriesName)	

ClearLegends

Clears all legend definitions of the chart.

ClearLegends	
SYNTAX	ClearLegends()
EXAMPLE	Chart.ClearLegends()

SeriesExists

Checks whether the series with the provided name exists.

SeriesExists		
SYNTAX	SeriesExists(string seriesName)	
ARGUMENTS	seriesName	Case-sensitive name of the series to check.
RETURN	TRUE = The series with the provided name exists. FALSE = The series with the provided name does not exist.	
EXAMPLE	if (Chart.SeriesExists(MySeriesName)) ... end-if	

DataPoint	Allows you to modify the properties of the data point.

Properties

Property	Description
Datapoint.Exploded	Sets or gets the custom property **Exploded** for the data point. Is used, for example, in pie charts.
Datapoint.IsValueShownAsLabel	Sets or gets whether the value of the data point is displayed on the label. TRUE: Display the value. FALSE: Do not display the value.
Datapoint.Font	The font of the data point.
Datapoint.Label	The label for the data point.
Datapoint.XValue	The X-axis value of the data point.
Datapoint.YValue	The Y-axis value of the data point.

Functions

ClearYValues

Resets the Y-axis values of the data point.

ClearYValues	
SYNTAX	ClearYValues()
EXAMPLE	`datapoint.ClearYValues()`

SetCustomProperty

Sets a custom property for the data point.

SetColor	
SYNTAX	SetCustomProperty(string propertyName, string propertyValue)
ARGUMENTS	propertyName: See https://msdn.microsoft.com/en-us/library/dd456764.aspx propertyValue: See https://msdn.microsoft.com/en-us/library/dd456764.aspx
EXAMPLE	dataPoint.SetCustomProperty("LabelStyle", "Top")

SetColor

Defines the color of the data point.

SetColor	
SYNTAX	SetColor(int red, int green, int blue) SetColor(int red, int green, int blue, int alpha)
ARGUMENTS	red: Red portion of the color. green: Green portion of the color. blue: Blue portion of the color. alpha: Alpha value of the color.
EXAMPLE	dataPoint.SetColor(41,91,207)

SetLabelForeColor

Defines the foreground color of the label for the data point.

SetLabelForeColor

SYNTAX SetLabelForeColor(int red, int green, int blue)

ARGUMENTS

red	Red portion of the color.
green	Green portion of the color.
blue	Blue portion of the color.

EXAMPLE dataPoint.SetLabelForeColor(48,243,53)

SetXYStringValue

Sets a string value for the X-axis and a double value for the Y-axis.

SetYValues

SYNTAX SetXYStringValue(string xValue, double yValue)

ARGUMENTS

xValue	String that is displayed for the data point on the X-axis.
yValue	Value that is displayed for the data point on the Y-axis.

EXAMPLE dataPoint.SetXYStringValue("Feb", 60)

SetYValues

Defines the Y-axis values for the data point.

SetYValues

SYNTAX SetYValues(params object[] yValue)

ARGUMENTS	yValue	Multiple y-values to set for the data point.
EXAMPLE	dataPoint.SetYValues(180)	

Font

Allows you to modify the font properties.

Properties

Property	Description
Font.FontName	Sets or gets the name of the font to use. Example: Arial
Font.Size	Sets or gets the size of the font.

Legend

Allows you to modify the properties of the legend

Properties

Property	Description
Legend.DisplayInChartArea	Determines if the legend displays within the chart area or outside in the common area of the chart.
Legend.Font	Sets or gets the font of the legend.
Legend.Title	Sets the title of the legend.
Legend.TitleFont	The font of the title.

Functions

AddToSeries

Adds the legend to a given series.

AddToSeries		
SYNTAX	AddToSeries(string seriesName)	
ARGUMENTS	seriesName	Name of the series for which the legend will be used.
EXAMPLE	legend.AddToSeries(MySeriesName)	

SetForeColor

Defines the color of the legend.

SetForeColor		
SYNTAX	SetForeColor(int red, int green, int blue)	
ARGUMENTS	red	Red portion of the color.
	green	Green portion of the color.
	blue	Blue portion of the color.
EXAMPLE	legend.SetForeColor(239,52,211)	

SetPosition

Sets the position of the legend within the chart area.

SetPosition

SYNTAX	SetPosition(position)
ARGUMENTS	position — Position. **Possible values** • Top • Right • Bottom • Left
EXAMPLE	legend.SetPosition(Left)

Functions of the Context TextVariable

Navigate

Enables you to navigate through the child objects of the current object.

Navigate

SYNTAX	Navigate(onObjectScript, onlyActiveNodes, filterTypes)
ARGUMENTS	onObjectScript — Script that executes for each of the specified object types found. This script must have the context OnNavigateObject assigned. onlyActiveNodes — TRUE = Only navigates active objects. FALSE = Navigates all objects. filterTypes — Filter defining the objects to navigate. **Possible values**

	TextSystemObjectType.Variable TextSystemObjectType.Block TextSystemObjectType.DocumentReferences
EXAMPLE	Navigate("Scriptname", true, TextSystemObjectType.Block, TextSystemObjectType.Variable)

Sample Script: Generate a Chart

Assign the script to the "BeforeInsert" event of the text block variable that is defined as a picture variable.

The predefined chart is assuming to have a series named "MySeriesName" and a chart area named "MyChartAreaName".

```
if(HasChartDefinition)
if (Chart.ChartAreaExists("MyChartAreaName"))
if (Chart.SeriesExists("MySeriesName"))
  Chart.ClearLegends()
  legend = Chart.AddLegend("MyChartAreaName")
  legend.AddToSeries("MySeriesName")
  legend.DisplayInChartArea = false
  legend.SetForeColor(239,52,211)
  legend.Title = "The Legend"
  legend.SetPosition("Left")

  font = legend.Font
  font.Size = 11
  font.FontName = "Algerian"
  legend.Font = font

  font = legend.TitleFont
  font.Size = 17
```

```
  font.FontName = "Algerian"
  legend.TitleFont = font

  dataPoint = Chart.CreateDataPoint("MySeriesName")
  dataPoint.XValue = 0
  dataPoint.YValue = 90
  dataPoint.Label = "90°"
  dataPoint.SetColor(41,91,207)

  dataPoint = Chart.CreateDataPoint("MySeriesName")
  dataPoint.XValue = 0
  dataPoint.SetYValues(180)
  dataPoint.Label = "180°"
  dataPoint.SetColor(52,192,239)
  dataPoint.SetLabelForeColor(48,243,53)
  dataPoint.Exploded = true

  dataPoint = Chart.AddDataPoint("MySeriesName", 0, 10)
  dataPoint.Label = "10°"
  dataPoint.SetColor(152,192,239)
  dataPoint.SetLabelForeColor(48,243,70)
  ; After all modifications have been done, assign the context object's XML to the textblock variable property
Content in order to get the chart generated when generating the document.
  Content = Chart.Xml
end-if
end-if
end-if
```

Context XsdSetXmlData

About the Context XsdSetXmlData

Use this context for scripts assigned to an XSD selection object.

The script is executed if the DataSelectionManager component retrieves XSD selection data.

Properties of the Context XsdSetXmlData

Property	Description
XmlData	Contains the business data of an XSD selection object. The script assigns an XML string.

Sample Script XsdSetXmlData

```
;selection parameter are available in the systempool

id = System.CustomerId

;do something to acquire the xml from an external DB, file share etc.

;...

;finally assign the xml string to the context property "XmlData"

XmlData = "<customer><id>"+ id
+"</id><surname>John</surname><lastname>Doe</lastname></customer>"
```

Script Order

The following figure shows the order in which scripts and conditions execute in a bundle

<table>
<tr><td>

_____</td><td colspan="2">Bundle
AfterDataRetrieval
---------->Query manual variables
of all active objects
OnGetManualVariable
BeforeCreatingDocuments
InitializeDocumentCreation</td></tr>
<tr><td></td><td></td><td>DocumentReference
BeforeInsert
Condition</td></tr>
</table>

<table>
<tr><td rowspan="9"></td><td></td><td></td><td colspan="3">Document
Load
Condition</td></tr>
<tr><td></td><td></td><td></td><td colspan="2">Text Block
BeforeInsert
Condition</td></tr>
<tr><td rowspan="4"></td><td rowspan="4"></td><td rowspan="4"></td><td rowspan="4"></td><td>Variable
BeforeInsert
Condition
AfterInsert

Child Text Block
BeforeInsert
Condition
AfterInsert

Variable
BeforeInsert
Condition
AfterInsert</td></tr>
<tr><td></td><td></td><td></td><td colspan="2">AfterInsert</td></tr>
<tr><td></td><td></td><td colspan="3">Unload
---------->DocX created</td></tr>
<tr><td></td><td colspan="4">DefinitionSorting
DefinitionGrouping
DefinitionArchiving
AfterInsert</td></tr>
<tr><td>_____

 </td><td colspan="4">AfterCreateDocuments
_____
BeforePrintingAndArchiving
InitializePrintAndArchive
BeforePrintingDocument</td></tr>
</table>

Alias Tables

About Alias Tables

Alias tables enable you to store assignments for variable text blocks or variable documents.

Creating an Alias Table

Complete the following steps to create an alias table for a variable text block or an variable document.

1. In the bottom pane of the **Navigator**, right-click and click **New** > **Alias Table**.
2. In the **Create New Alias Table** dialog box, type a name for the alias table and click **OK**.
3. On the **Lookup alias table** tab, click the **New Key** button.
4. In the **Create New Key** dialog box, type the value contained in the corresponding variable for the key and click **OK**.
   Example
   A variable contains a country code. Depending on the value of this variable, a different text block is inserted into a document.
   Create a key for every single possible value of the variable, for example DE, GB, NL, FR.
5. In the **Editor** window, select a key.
6. On the **Lookup alias table** tab, click the **New Reference** button.
7. In the **Select an object** dialog, select the text block or the document you want to insert when the variable contains the value of the key.

For more information, refer to Define a variable text block and Define a variable document.

Lookup Tables

About Lookup Tables

Lookup tables enable you to specify value assignments to replace variable contents during document creation.

Creating a Lookup Table

To create a lookup table, complete the following steps.

1. In the bottom pane of the **Navigator**, right-click and click **New** > **Lookup Table**.
2. In the **Create New Lookup Table** dialog box, type a name for the lookup table and click **OK**.
3. In the **Editor** window, type the value pairs in the following format
   `Variable content=value to be inserted`

Note Type each value pair on a separate line.

Example

A variable contains a salutation code. Create the following value pairs for the variable.

001=Mrs.

002=Mr.

003=Company

If this lookup table is assigned to a text block variable that contains the value **003,** the content of the variable is replaced with the content **Company**.

Transfer Jobs

About Transfer Jobs

Transfer jobs are a collection of objects you want to transfer to another system.

Use the **System Administration** to define the available target systems.

Transfer of Versioned Objects

- You can transfer both work versions and release versions to other systems.
- To allow the import of work versions, open the **System Administration** and set the **Import Work Versions** property of the appropriate system to `True`.
- On transfer, the object in the target system receives the same version number as the object in the source system.

Transfer of Release Versions

- If the version number of the object in the source system is equal or lower than in the target system, the transfer does not execute for this object.
- If the version number of the object in the source system is higher than in the target system, the transfer executes for this object.
- If the object in the target system is logically deleted, the following rules apply.

Transfer of Work Versions

- If the version number of the object in the source system is lower than in the target system, the transfer does not execute for this object.
- If the version number of the object in the source system is equal to the version number in the target system, the object undeletes in the target system.
- If the version number of the object in the source system is higher than in the target system, the transfer executes for this object. The object undeletes in the target system.

- If the version number of the object in the source system is lower than in the target system, the transfer does not execute for this object.
- If the version number of the object in the source system is equal to the version number in the target system, the following rules apply.
  - If the timestamp (last modified at) in the source system is newer than in the target system, the transfer executes for this object.
    Important The transfer is not revocable for this object!
  - If the timestamp (last modified at) in the source system is older than in the target system, the transfer does not execute for this object.
- If the version number of the object in the source system is higher than in the target system, the transfer executes for this object.
- If the object in the target system is logically deleted, the following rules apply.
  - If the version number of the object in the source system is lower than in the target system, the transfer does not execute for this object.
  - If the version number of the object in the source system is equal to the version number in the target system, the object undeletes in the target system. The following rules apply.
    - If the timestamp (last modified at) in the source system is newer than in the target system, the transfer executes for this object.
      Important The transfer is not revocable for this object!
    - If the timestamp (last modified at) in the source system is older than in the target system, the transfer does not execute for this object.
  - If the version number of the object in the source system is higher than in the target system, the transfer executes for this object. The object undeletes in the target system.

Transfer History

For more information, refer to Transfer History.

Revoking a Transfer

For more information, refer to Revoke a transfer.

Creating a Transfer Job

To create a transfer job, complete the following steps:

1. In the bottom pane of the **Navigator**, right-click and click **New** > **Transfer Job**.
2. In the **Create New Transfer Job** dialog box, type a name for the transfer job and click **OK**.
3. From the **Navigator** pane, drag the required objects to the left-hand pane of the **Editor** window.
4. Select the objects that might contain child objects that are not yet included in the transfer job.
5. On the **Object Transfer** tab, in the **Edit** group**,** click the **Show required objects** button.
6. In the **Object 'XY' references the following objects** dialog box, select the objects that you want to add to the transfer job.
7. In the right-hand pane of the **Editor** window, select the required target systems.
8. Optional. To flag objects as write-protected in the target system, in the right-hand pane of the **Editor** window, select the **Set Write Protection** check box.
9. To start the transfer, on the **Object Transfer** tab, under **Execute**, click the **Execute** button.
   Note If version control is not licensed, existing objects in the target system will be overwritten.

Transfer Status

After executing a transfer, in the **Editor** window, the **Status** column displays a transfer status icon for each object.

Icon	Description
	Transfer successfully executed.
	Transfer to at least one system not executed. For more information refer to the transfer log file. Path and name of the log file display in the **Output** window.
	Transfer failed. Check the error message in the **Output** pane.

Creating a Scheduled Transfer Job

To create a scheduled transfer job, complete the following steps.

Prerequisites

You must have a Versioning license to create scheduled transfer jobs.

The user assigned to the **modus_core** Windows service must have user role access to the intended target system of the scheduled transfer.

1. In the bottom pane of the **Navigator**, right-click and click **New** > **Transfer Job**.
2. In the **Create New Transfer Job** dialog box, type a name for the transfer job and click **OK**.
3. From the **Navigator** pane, drag the required objects to the left-hand pane of the **Editor** window.
4. Select the objects that might contain child objects that are not yet included in the transfer job. Then, on the **Object Transfer** tab, in the **Edit** group**,** click the **Show required objects** button.
5. In the **Object 'XY' references the following objects** dialog box, select the objects that you want to add to the transfer job.
6. In the right-hand pane of the **Editor** window, select the required target systems.
7. Optional. To flag objects as write-protected in the target system, in the right-hand pane of the **Editor** window, select the **Set Write Protection** check box.
8. On the **Object Transfer** tab, in the **Execute** group**,** click the **Schedule** button.
9. In the **Schedule Transfer** dialog box, select the deferred execution date and time for the scheduled

transfer.
Note The date can be specified to the nearest hour.

10. Click **OK**.

Notes

- At the specified time, the transfer process starts the transfer job.
- An Administration Function allows you to edit or delete scheduled transfers.

Exporting Objects

If you need to transfer individual objects or complete systems to another computer, you can export them to a file system and import them into the target computer.

To export objects, complete the following steps.

1. In the bottom pane of the **Navigator**, right-click and click **New** > **Transfer Job**.
2. In the **Create New Transfer Job** dialog box, type a name for the transfer job and click **OK**.
3. From the **Navigator** pane, drag the required objects to the left-hand pane of the **Editor** window.
4. Select the objects that might contain child objects that are not yet included in the transfer job. Then, on the **Object Transfer** tab, in the **Edit** group**,** click the **Show required objects** button.
5. In the **Object 'XY' references the following objects** dialog box, select the objects that you want to add to the transfer job.
6. To start the export, on the **Object Transfer** tab, under **Execute**, click the **Export** button.
7. In the **Export to File System** dialog box, type a name for the subdirectory where you want to export the objects.
   Note This directory appears directly below the RWS work directory. Its name is a combination of the computer and user name as well as the date and time.

For information about importing objects, refer to Repository Import.

Transfer Job Editor Tools

Icon	Name	Description
	Execute	Starts the transfer.

	Export	Starts the export to the file system.
	Schedule	Creates a scheduled deferred transfer.
	Cancel	Cancels a running transfer.
	Add	Opens a dialog where you can select objects you want to add to the transfer.
	Remove	Removes the selected object from the transfer job.
	Show required objects	Shows the objects required by the selected objects.
	Create Release Version	Creates release versions for all selected objects in the transfer. If no objects are selected, release versions are created for all work versions in the transfer.
	Refresh	Refreshes the transfer objects.

Value Helps

About Value Helps

Value helps assist users when typing input into manual variables.

Static Value Helps

About Static Value Helps

Static value helps provide equal input options on each occurrence of a manual variable.

Creating a Text Box with Input Mask Value Help

This value help type allows you to specify an input mask to define how to insert input into manual variables.

To create a text box with input mask value help, complete the following steps.

1. In the bottom pane of the **Navigator**, right-click and click **New** > **Value Help**.
2. In the **Create New Value Help** dialog box, type a name for the value help and click **OK**.
3. In the **Value Help Types** dialog box, select **Text Box with Input Mask** and click **OK**.
4. In the **Input Mask** box, type the input mask using the required **Formatting Characters**.

Character	Meaning in the Input Mask
!	Specifies that optional characters are displayed as leading spaces in the input mask. If ***!*** is not specified, optional characters are displayed as trailing spaces in the mask.
>	Converts all following characters to uppercase until the end of the mask is reached or the character < is detected.
<	Converts all following characters to lowercase until the end of the mask is reached or the character > is detected.
<>	If both signs are used together, no case sensitivity check is made. The data specified is formatted in exactly the same way a user types it.
\	Specifies that the subsequent character is displayed as a placeholder in the input mask.
L I	Specifies that only letters are valid entries (A-Z, a-z). An entry is not required.
A a	Specifies that only alphanumeric characters are valid (A-Z, a-z, 0-9). An entry is not required.
C c	Any character is allowed. An entry is not required.
0 9	Specifies that only numeric characters are valid. An entry is not required.
#	Specifies that only a numeric character or a plus or minus sign is allowed. An entry is not required.
:	A colon separates hours, minutes, and seconds in defining a time. If a different character is specified in the Windows regional settings, that

	character displays instead of a colon.
/	A slash separates the date entries day, month, and year from each other. If a different character is specified in the Windows regional settings, that character displays instead of the slash.
;	A semicolon separates the three mask parameters from each other.
_	The underline character _ automatically inserts a space in the text. The cursors skips over these characters when data is typed.

You can use any characters not listed in the above table as placeholders for input.

These characters are inserted automatically and skipped by the cursor on data entry.

The special mask characters listed in the table above can also be used as placeholders as long as the "\" character is specified first.

5. Optional. In the **Mask Test** box, type a value to test how the input mask affects the specified variables.
6. Optional. In the **Description** box**,** type a description.
7. On the **Value Help** tab, click the **Test Value Help** button.
8. In the **Test Value Help** dialog box, test the input mask.

Creating a Drop-Down Value Help

This value help type allows you to define a list of values that display in a drop-down menu during manual variable input.

To create a drop-down value help, complete the following steps.

1. In the bottom pane of the **Navigator**, right-click and click **New** > **Value Help**.
2. In the **Create New Value Help** dialog box, type a name for the value help and click **OK**.
3. In the **Value Help Types** dialog box, select **Drop-down**.
4. In the **Entries in the selection list** box**,** type the list of required values with each value on a separate line.
   Note You can also define values in the form of **[ModusUser.AttributeName]**.
5. Optional. To display the list in alphabetic order, select the **Sort List** check box.
6. Optional. In the **Description** box**,** type a description.
7. On the **Value Help** tab, click the **Test Value Help**

button.

8. In the Test **Value Help** dialog box, test the input mask.

Creating a Popup Calendar Value Help

This value help type provides a calendar as a way to select a date.

To create a popup calendar value help, complete the following steps.

1. In the bottom pane of the **Navigator**, right-click and click **New** > **Value Help**.
2. In the **Create New Value Help** dialog box, type a name for the value help and click **OK**.
3. In the **Value Help Types** dialog box, select **Popup Calendar**.
4. In the **Predefined Formats** list, select the preferred date format from the following options.
   Long Date in the form of Monday, 5.November 2011
   Short Date in the form of 05.11.2011
   Custom User-defined date format

   In the **Self-defined format** box, type the preferred date format.

   You can use any of the following definitions to set up a custom date format.

Definition	Result
dd	Day, numeric two digits
ddd	Name of the weekday (abbreviated)
dddd	Name of the weekday (full)
MM	Month, numeric two digits
MMM	Name of the month (abbreviated)
MMMM	Name of the month (full)
yy	Year, numeric two digits
yyyy	Year, numeric four digits
hh	Hours

mm	Minutes
ss	Seconds

Example

dddd, MMMM dd, yyyy hh:mm:ss

Result

Tuesday, November 05, 2013 10:58:59

5. Optional. In the **Description** box**,** type a description.
6. On the **Value Help** tab, click the **Test Value Help** button.
7. In the Test **Value Help** dialog box, test the input mask.

Creating a Numeric Input Field Value Help

Use this value help type to define the value range of a numeric variable.

To create a numeric input field value help, complete the following steps.

1. In the bottom pane of the **Navigator**, right-click and click **New** > **Value Help**.
2. In the **Create New Value Help** dialog box, type a name for the value help and click **OK**.
3. In the **Value Help Types** dialog box, select **Numeric Input Field**.
4. In the **Minimum Value** box, specify the minimum value for the variable.
5. In the **Maximum Value** box, specify the maximum value for the variable.
6. In the **Number of decimal places** box, specify the number of decimal places.
7. In the **Increment**box, specify the value of the individual increment steps.
8. Optional. In the **Description** box**,** type a description.
9. On the **Value Help** tab, click the **Test Value Help** button.
10. In the **Test Value Help** dialog box, test the input mask.

Creating a Multiline Text Box Value Help

Use this value help type to type multiple lines of text when specifying the variable value.

To create a multiline text box value help, complete the following steps.

1. In the bottom pane of the **Navigator**, right-click and click **New** > **Value Help**.

2. In the **Create New Value Help** dialog box, type a name for the value help and click **OK**.
3. In the **Value Help Types** dialog box, select **Multiline Textbox** and click **OK**.
4. Optional. To restrict the text length, select the **Restrict text field input** check box and specify the value in the **Maximum Text Length** box.
5. Optional. Type an explanatory text in the **Description** field.
6. On the **Value Help** tab, click the **Test Value Help** button.
7. In the **Test Value Help** dialog box, test the input mask.

Dynamic Value Helps

About Dynamic Value Helps

Dynamic value helps extend static value helps by allowing value definition at runtime. This feature prevents the presentation of inapplicable input options to the end user.

The processing of dynamic value help is carried out by scripts executed in the ManualVariable context.

When manual variables are queried, the OnGetManualVariable event executes for each variable.

Word User Interfaces

About Word User Interfaces

Word User Interfaces enable you to restrict user editing functions when finishing documents in Word.

To do this, you need to assign the Word User Interface to a document in a bundle.

Creating a Word User Interface

To create a Word user interface, complete the following steps.

1. In the bottom pane of the **Navigator**, right-click and click **New** > **Word User Interface**.
2. In the **Create New Word User Interface** dialog box, type a name for the Word user interface and click **OK**.
3. To insert a template, on the **Word User Interface Tools** tab, click the **Load XML Template** button.
4. Modify the template according to your specific requirements.
   For more information, refer to the Microsoft documentation.
5. Optional. To format the code, on the **Word User Interface Tools** tab, click the **Format XML**

button.

6. To check the code, on the **Word User Interface Tools** tab, click the **Check XML** button.
7. Optional. In the **Object Inspector** pane, type a free text description for the **Description** property.

Assigning a Word User Interface to a Document

To assign a Word user interface to a document, complete the following steps.

1. Open the bundle that contains the respective document.
2. In the **Editor** window, click the document.
3. In the **Object Inspector** pane, click the **Word User Interface** property and click the ellipsis button next to the property.
4. In the **Select an object** dialog box, select a Word user interface and click the **Select**button.

Word Templates

About Word Templates

To work with Word templates, the user must possess the Word Template (74) object right.

Content Composer supports the import of Word templates, allowing you to load a Normal.dotm into the system.

You can use Word templates to specify the Normal.dotm you want to use in a particular system. You are able to edit the Normal.dotm as an object in Studio and to export the file for external use.

You can assign a specific Word template containing a Normal.dotm to each system. Without an assignment, Content Composer uses the template specified in the Content Composer configuration. For more information, refer to Assign a Word Template.

You can transfer or export Word templates into any other system. After being assigned, these Word templates are ready for use in the target system. For more information, refer to Assign Word Template.

A Normal.dotm template contains styles you can use in a Word document. Each new document you create in Word will use this template until you select a different, specific template file.

Content Composer utilizes Normal.dotm in various situations. When organizing document creation, take care to provide the appropriate Normal.dotm during the steps of document processing.

- **Creating new objects and modifying existing objects in Content Composer Studio**. When you create documents with a Normal.dotm different from the one used for text administration, document styles differ accordingly.
- **Generating documents**. You have to use identical Normal.dotm templates for creation and final generation of documents.

- **Re-working generated documents in the Client**. You have to use identical Normal.dotm templates on the client computer used for re-working and on the server used for creation.

Note We recommend to use only self-defined styles of a Normal.dotm template in Content Composer.

The use of standard styles may cause issues when you work with multiple languages or when standard styles are changed.

- Microsoft Word does not assign a consistent ID for standard styles used in additional languages.
- Microsoft Word does not reset standard styles after the cancellation of additional definitions.

Creating a Word Template

To create a Word template, complete the following steps.

1. In the bottom pane of the **Navigator**, right-click and click **New** > **Word Template**.
2. In the **Create New Word Template** dialog box, type a name for the Word template and click **OK**.
3. On the **Editor Tools** tab, click the **Import Word Template** button.
4. In the **Open** dialog box, select a Word document template.
5. Optional. In the **Object Inspector** pane, type a free text description for the **Description** property.
6. In **Content Composer Studio**, save the Word template.
7. Assign the Word template to a system.

Modifying a Word Template

To modify a Word template, complete the following steps.

1. Open a Word template.
2. On the **Editor Tools** tab, click the **Show Word Document** button.
3. Modify the document styles.
   Note The Word document should remain empty.
4. In **Content Composer Studio**, save the Word template.

Note When a text administrator saves changes to a word template assigned to a specific system, these changes are immediately visible to this text administrator.

As another text administrator working with the studio on a different machine in the same system, you initially will not recognize the changes made to the word template.

Complete one of these steps to be able to see the changes.

- Restart the Studio.
- Switch to another system and then switch back to the previously used system.

Exporting a Word Template

Content Composer supports the export of Word templates, allowing you to save a template object of the system as a Normal.dotm file.

To export a Word template, complete the following steps.

1. Open a Word template.
2. On the **Editor Tools** tab, click the **Export Word Template** button.
3. In the **Save as** dialog box, select a directory, type a name for the file in the **File name** box, and then click **Save**.

Processes

About Processes

The process object enables you to make simple graphical representation of processing sequences.

The process designer provides you with a graphical user interface to create, save, and edit models of your work processes.

A distinction is made between "Standard Processes" that control processes and sequences within Content Composer Studio and "MWS Processes" that control processes and sequences in the Content Composer Client.

Creating a Process

To create a process, complete the following steps.

1. In the bottom pane of the **Navigator**, right-click and click **New** > **Process**.
2. In the **Create New Process** dialog box, type a name for the process and click **OK**.
3. In the **Process Selection** dialog box, select one of the following options.
   - Standard Process
   - MWS Process
4. Optional. In the **Object Inspector** pane, edit the properties of the process.
5. From the **Toolbox**, drag the preferred modules to the **Editor** window.
6. In the **Editor** window, position and connect the process designer modules as preferred.
7. From the **Toolbox**, drag the preferred components to the **Editor** window.
8. In the **Process Designer Tools** tab, click the **Compile**

button.

9. Observe success or failure messages in the **Output** pane.
10. Start the process using the **Process Designer Tools** buttons.

Properties of a Standard Process

Property	Description
Description	Free text description.
Context	The context of a standard process is always **Process**.
OnError	Script that executes in case of an error. Context: OnProcessError
Objectname	Name of the process.

Properties of an MWS Process

Property	Description
Description	Free text description.
Context	The context of a MWS process is always **MwsProcess**.
OnGetInfoScript	Script called by the `Process_GetInfo`. function The script sets information for activities.
OnNewProcessScript	Script that executes when a new process is created.
OnProcessEndScript	Script that executes when a process is closed using `Process_Close`.
OnSetOptionsScript	Script that executes when XML options are set for activities. This always refers to external information that has nothing to do with the MWS process itself.
OnStatusChangeScript	Script that executes when the structure of the business component is modified.
ProcessRootDirectory	Used to specify the directory where MWS process data is stored. The specification can also contain variables, for example

	"$server\processes". The temporary directory is used if this property is empty.
OnError	Script that executes in case of an error. Context: OnProcessError
Objectname	Name of the process.

Process Designer Tools

The toolbar provides you with a number of functions for editing processes.

Icon	Name	Description
	Align	Adjusts the horizontal and vertical position of blocks.
	Cut	Deletes the selected object and copies it to the clipboard.
	Copy	Copies the selected object.
	Paste	Inserts an object from the clipboard.
	Undo	Undoes the last action.
	Repeat	Repeats the last action.
	Compile	Compiles the process.
	Execute	Runs the process.
	Stop	Stops the process.

	Pause	Pauses the process.
	Cancel	Cancels the process.
	Zoom In	Zooms in the editor window.
	Zoom out	Zooms out the editor window.
	Normal Size	Returns the view of the editor window to normal size.
	Adjust	Fits the size of the view of the editor window to make all process blocks visible.

Process Designer Modules

Areas of a Process Module

	Port Connection point to other modules	

<table>
<tr>
<td>

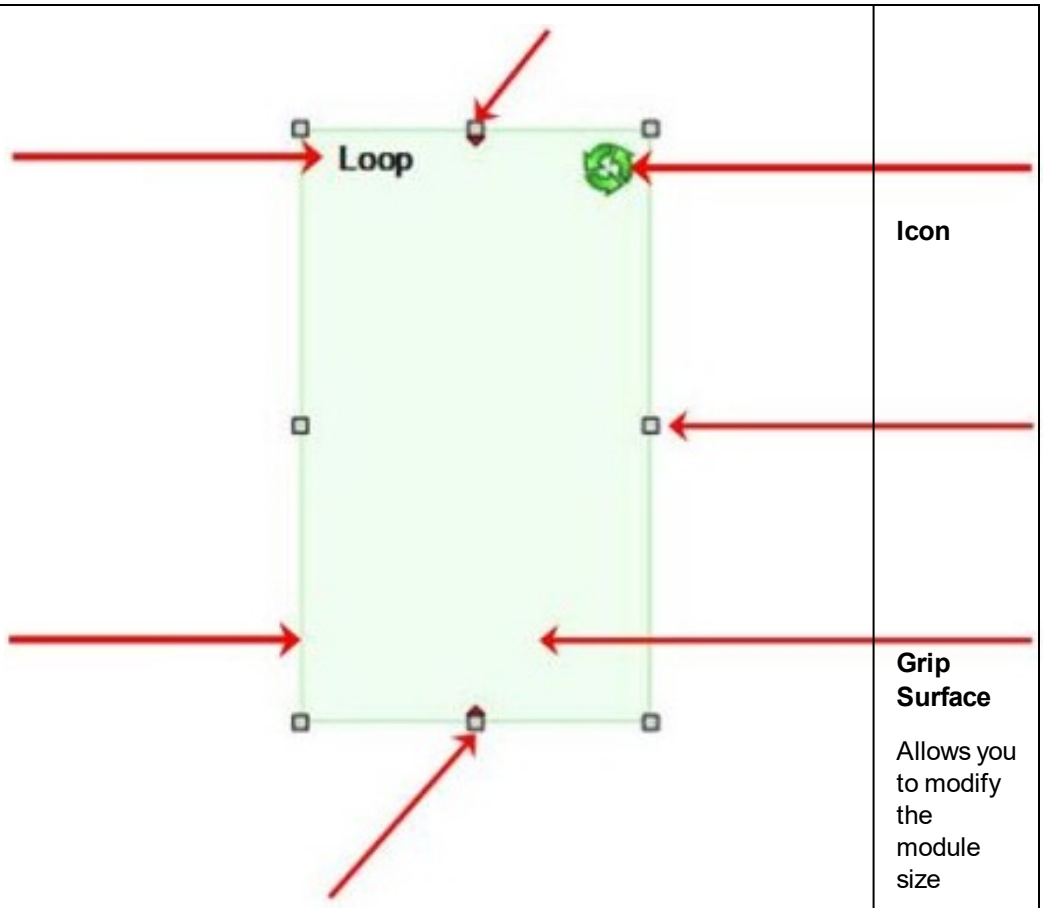
Name

Freely definable module name

Can automatically display the name of the attached script

Border

</td>
<td>



</td>
<td>

Icon

Grip Surface

Allows you to modify the module size

Container

Used to hold other modules

(Only for container-modules)

</td>
</tr>
<tr>
<td></td>
<td>

Port

Connection point to other modules

</td>
<td></td>
</tr>
</table>

Check

A check module can contain one ore more sequentially checked conditions.

Properties of a check module

Property name	Description
Check Type	First: One of the conditions defined must return `True` for processing to continue. Any: Any of the conditions defined must return `True` for processing to continue.

Condition

Conditions group in a check module. This defines the order in which conditions are checked.

Properties of a condition module

Property name	Description
Condition	Condition object.
Condition Type	Type / Effect Complex: This condition type corresponds to an IF-query. WhenNone: Choose this condition type if none of the complex type conditions in the check module apply and you need the check module to react to this. You can then assign a script to the condition to pass on an error message. WhenAny: Choose this condition type if one of the complex type conditions in the check module apply and you need the check module to react to this. WhenAll: Choose this condition type if all of the complex type conditions in the check module apply and you need the check module to react to this.
Description	Optional explanatory text.

IF-Check

An **IF-Check** comprises of a **Check** module that contains a **Condition** module.

IF-ELSE-Check

An IF-ELSE-Check comprises of a **Check** module containing two **Condition** modules.

The first condition is set by default to the condition type **Complex** and corresponds to the If-query.

The second condition is set by default to the condition type **WhenNone** and corresponds to the Else-query.

Comment

Use comment modules to add explanatory text to the process.

Loop

A loop module enables you to repeat a sequence of steps up to the point the condition is no longer valid.

Properties of a loop module

Property name	Description
Condition	Condition object
Description	Optional explanatory text

Script Execution

A script execution module functions as a placeholder for a script and makes starting a script from a process possible.

Note If the script name exceeds a certain length, it truncates for clarity. Right-click the script and click **Show Complete Name** to display the full name.

Properties of a script execution module

Property name	Description
Script	Script object
Description	Optional explanatory text

Universal Timer

A universal timer module can contain any kind of **Process Designer** modules and enables you to run processes within a specific time frame.

Properties of a universal timer module

Property name	Description
Script	Script object
OnError	Script that executes in case of an error.
Description	Optional explanatory text
Enabled	TRUE: the timer is active FALSE: the timer is not active
Interval	Interval between two timer events in milliseconds
Synchronous	TRUE: synchronous timer operating mode FALSE: asynchronous timer operating mode
UniversalTimer	To create at timer, complete the following steps. 1. Double-click the **UniversalTimer** property. 2. Click the ellipsis button to the right of the **UniversalTimer** property. 3. In the **Settings** dialog box, specify the timer settings.

Position and Connect Process Designer Modules

About Positioning and Connecting Process Modules

- You can drag process modules from the Toolbox to the **Editor** window.
- You can drag existing scripts directly from the Navigator to the **Editor** window.
- You can drop a process module as soon as you move it towards another process module. This is signaled by the existing process module by changing the color of its frame.

Examples: Add Modules to a Process

Adding a Loop to a Process

To add a loop, complete the following step.

- From the **Toolbox**, drag a loop module to the **Editor** window.

Inserting a Script Execution in the Loop

To insert a script execution in the loop, complete the following steps.

1. From the **Toolbox**, drag a script execution module to the loop in the **Editor** window and drop the script execution exactly onto the light green area of the loop.

2. In the **Object Inspector** pane, click the **Script** property and click the ellipsis button next to the property.
3. In the **Select an object** dialog box, select the preferred script and click **Select**.

Inserting Another Script Execution in the Loop

To insert another script execution in the loop, complete the following steps.

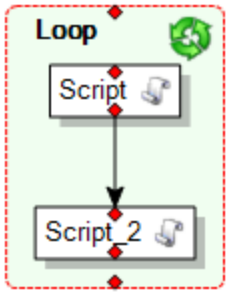
1. From the **Toolbox**, drag a script execution module to the loop in the **Editor** window. Depending on whether you want to position the script execution before or after the first script, move it towards the top edge or bottom edge of the script, then drop the script execution when the border of the first script changes to red dashes.

2. In the **Object Inspector** pane, click the **Script** property and click the ellipsis button next to the property.
3. In the **Select an object** dialog box, select the preferred script and click **Select**.

Inserting a Script Execution as Loop Successor

To insert a script as loop predecessor or successor, complete the following steps.

1. From the **Toolbox**, drag a script execution and move it towards the top edge or the bottom edge of the loop until the border changes its appearance to red dashes, then drop it.
   Note This change in appearance signals that the loop accepts the script as a predecessor or successor.

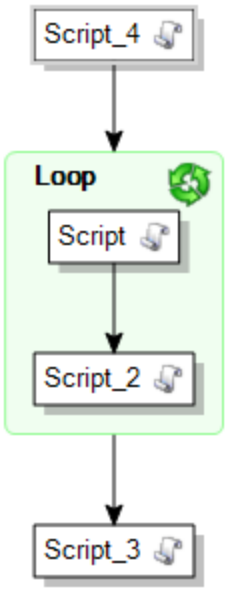


2. In the **Object Inspector** pane, click the **Script** property and click the ellipsis button next to the property.
3. In the **Select an object** dialog box, select the preferred script and click **Select**.

Notes

- Predecessor and successor means that the script is on the same hierarchy level as the loop.
- A predecessor is signaled by an arrow running from the bottom edge of the script to the top edge of the loop.
- A successor is shown by an arrow running from the bottom edge of the loop to the top edge of the script.

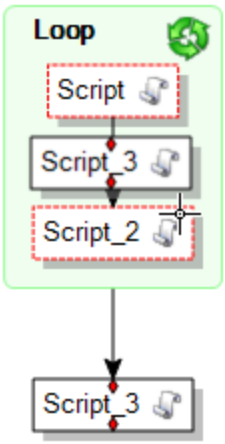
Example



Moving a Script Between two Already Linked Scripts

To move Script_3 from its logical position as loop successor and insert it between scripts 1 and 2, perform the following step.

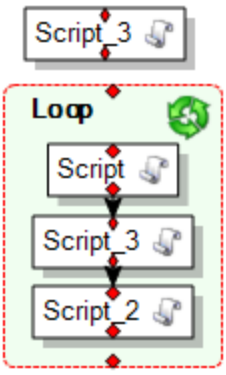
- Press CTRL and drag Script_3 to a position between Script_1 and Script_2 until the borders of both scripts change their appearance to red dashes, then drop Script_3.



Moving a Script to a Position Outside the Loop

To use Script_3 as a predecessor of the loop, complete the following step.

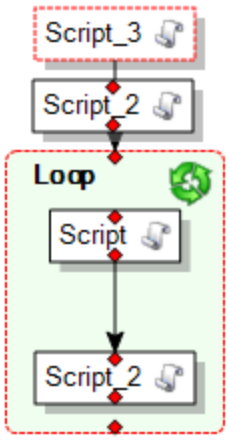
- Press CTRL and drag Script_3 over the top edge of the loop until the border changes to red dashes, then drop Script_3.



Inserting a Script Between a Script and a Loop

To insert Script_2 between Script_3 and the loop, complete the following step.

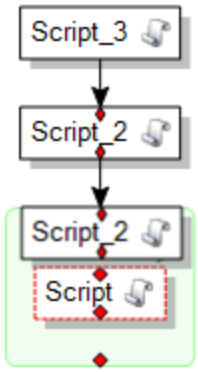
- Press CTRL and drag Script_2 to a position between the loop and Script_3 until the borders of both these two objects change to red dashes, the drop Script_2.



Inserting a Script Back in a Loop

To insert Script_2 back in the loop before Script, complete the following step.

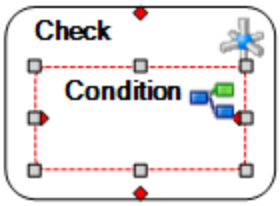
- Press CTRL and drag Script_2 over the top edge of Script inside the loop until the border of Script changes its appearance to red dashes, then drop Script_2.



Arranging Conditions in Succession

To arrange conditions in succession, complete the following steps.

1. From the **Toolbox**, drag a condition module to the **Editor** window.
2. In the **Object Inspector** pane, edit the properties of the condition module.
3. From the **Toolbox**, drag another condition module to the border of the first condition module so that the border changes its appearance to red dashes and a small PLUS sign displays, then drop the condition to position it after the first condition.

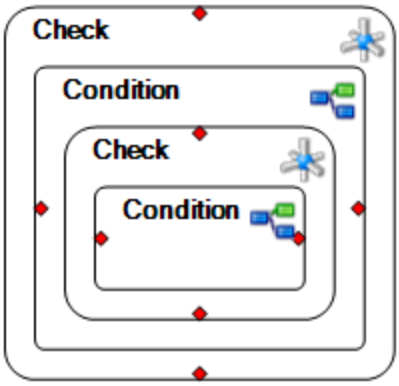


4. In the **Object Inspector** pane, edit the properties of the second condition module.

Nest Conditions

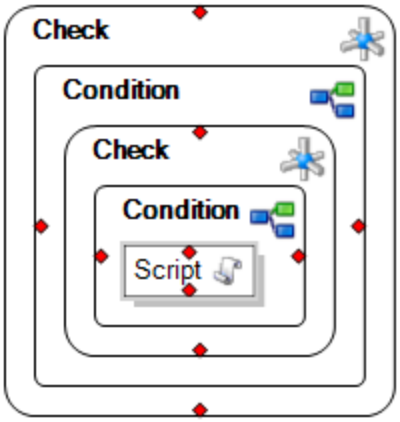
To nest conditions, complete the following steps.

1. From the **Toolbox**, drag a condition module to the **Editor** window.
2. In the **Object Inspector** pane, edit the properties of the condition module.
3. From the **Toolbox**, drag another condition module inside the border of the first condition module.
   Note The border stays as a black line.



4. Optional. Drag other objects, for example a script, into the condition module.

Note These objects execute if the condition is valid.



Process Designer Components

About Process Designer Components

Process Designer components provide a process with the necessary properties and functions to perform the specific processing tasks for which the process is designed.

CsvConverter

About CsvConverter

The CsvConverter component enables you to convert a comma separated (CSV) file to an XML file. This component supports both comma separated files and files that contain delimiter characters, such as, ';', '|', and '$'.

Properties of CsvConverter

The following table displays the properties of CSV converter.

Property	Description
Output directory	The directory where the generated XML file is saved.
Configuration	An XML string that specifies the settings to read the input file and create the XML output file. You can perform the following actions on the editor available. • **New**. Click the **New** button to get a default configuration template. • **Open**. Click the **Open** button to load an existing configuration file in the editor.

CsvConverter configuration

The following section describes the XML structure to specify the conversion from CSV to XML.

The XML has the following basic structure.

```
<csv2xml>
  <configuration>
    <fielddelimiters>…</fielddelimiters>
    <sourcefileencoding>…</sourcefileencoding>
    <textqualifier>…</textqualifier>
    <sourcefilecontainsheaderrow>…</sourcefilecontainsheaderrow>
    <lineseparator>…</lineseparator>
    <xmloutput>
    …
    <xmloutput>
  </configuration>
</csv2xml>
```

The following table displays the XML elements required for CsvConverter configuration.

XML element	Description
Csv2XML	This is the XML root element.
Configuration	This element contains several XML elements to specify how to interpret the CSV input file and the definition of the XML output structure. The attribute information is listed below. • **Name**. The unique name of a configuration element. Currently, only one configuration element is supported so the attribute value is always “default.”
fielddelimeters	Specifies the field delimiter characters that are used in the CSV input file to separate the data fields. • It is possible to specify 0-n delimiters, with each consisting of one character.

	• If no delimiter is specified, the complete data row is considered as a single column. • The delimiter character is specified with the ASCII value. The common delimiters and their ASCII values are listed below. ",". "," ";". ";" TAB. " " "|" . "|" ".". "." ":" . ":"
sourcefileencoding	Specifies the encoding used to read the CSV input file, for example, “windows-1250”. For more information, refer to the encoding parameters in the Append topic .
textqualifier	An optional character that encloses values in your text file. When the converter encounters the text qualifier character, the text that follows that character and precedes the next occurrence of that character is imported as one value, even if the text contains a delimiter character.
sourcefilecontainsheaderrow	Specifies if the first line of the CSV input file contains the names of the data columns or values. Possible values are as follows. • 'y'. First line contains the names of the data column. There is no data. • 'n'. First line contains data.
lineseparator	Defines the characters used in the input file to signal the end of a text line. The characters are specified with the ASCII value. This is an optional element. The default value is the standard windows line break "CRLF:
&#xA". Common ASCII values are as follows. • **Carriage return (CR)**. &#xD • **Line feed (LF)**. &#xA
xmloutput	This element contains XML elements that define the structure of

the generated XML output file. Below is a list of the attributes.

- name. The unique name of a configuration element. Currently, only one xmloutput element is supported so the attribute value is always “default.”
- jobelementpath and multiselmaxcontentcount

These two attributes specify that many XML output files are created from one input file and the name of the XML root element.

If the value of attribute "jobelementpath" is an empty string or missing, every data record contained in the input file is written into a single XML output file (one XML output file contains one data record). If the value of attribute "jobelementpath" contains a string having the format "/" + [root element name] + "/" + [element name]

multiple data records contained in the input file are written to a single XML output file. The value of [root element name] specifies the name of the XML root element.

The value of [element name] specifies the name of element that is the child of the root element and contains one data record.

The value of attribute "multiselmaxcontentcount" specifies the maximum number of data records that are written into one XML output file. If the input file contains more data records than the number specified with “multiselmaxcontentcount,” more than one XML output file is created.

If the value of “multiselmaxcontentcount” is zero, all data records are written into one XML output file.

The following elements are available to define the XML output structure.

- **xmlelement**. Defines an XML element of the output XML file. The attributes are listed below.
  - **name**. This is the name of the XML element in the output file.
  - **isdefault (bool)**. Possible values are ‘y’ (`true`) or ‘n’ (`false`). If `true`, this XML element gets all columns contained in the input file that are not explicitly assigned to any other XML element. Only one element can be a default element.

- **xmlattribute**. Defines the mapping of column value from the input file to an XML attribute in the output file. The “xmlattribute” element is a child element of a “xmlelement.” Regular expressions are supported. The attributes are listed below.
  - **name**. This is the name of the XML element in the output file.
  - **csvcolumnname**. This is the actual name of the column in the input file.

- **relation**. Defines the keys of a master-detail (child) relationship in case the input line contains structured data having 1-n relationships. The “relation” element is contained in the child “xmlelement” of the relation. The attributes are listed below.
  - **parentattribute**. Name of the primary key field in the parent (master) element.
  - **childattribute**. Name of the foreign key field in the child (detail) element.

Simple Example

This is an example for an input file with simple data rows that do not contain master-detail relationships.

Input file

```
invoice|amount|tax|date
0815|10|15.9|01/05/2015
```

Csv Converter configuration

```
<?xml version="1.0" encoding="utf-8"?>
<csv2xml>
 <configuration name="default">
  <fielddelimiters>|</fielddelimiters>
  <sourcefileencoding>windows-1250</sourcefileencoding>
  <textqualifier></textqualifier>
  <sourcefilecontainsheaderrow>y</sourcefilecontainsheaderrow>
  <lineseparator>&#xD;&#xA;</lineseparator>
  <xmloutput name="default" jobelementpath="/jobs/job" multiselmaxcontentcount="0">
   <xmlelement name="invoice" isdefault="true">
    <xmlattribute name="invoicenumber" csvcolumnname="invoice"/>
    <xmlattribute name="netamount" csvcolumnname="amount"/>
    <xmlattribute name="grossamount" csvcolumnname="tax"/>
    <xmlattribute name="date" csvcolumnname="date"/>
   </xmlelement>
  </xmloutput>
 </configuration>
</csv2xml>
```

Generated XML output file

```
<?xml version="1.0" encoding="utf-8"?>
<jobs>
 <job>
  <invoice invoicenumber="0815" netamount="10" grossamount="15.9" date="01/05/2015" />
 </job>
<job>
```

Advanced Example

This is an example for an input file where the following relationships are stored in each data row.

Customer

  Order (1-n)

   OrderLineItem (1-n)

So in one data row there can be stored data for one customer.

Data for 1-n orders of this customer.

For each order 1-n order line items.

Input file

```
CustomerName;CustomerId;OrderNumber1;OrderDate1;LineItemOrderNumber1;LineItemProduct1;LineItemPrice1;LineItemOrderNumber2;LineItemProduct2;LineItemPrice2;OrderNumber2;OrderDate2;LineItemOrderNumber1;LineItemProduct1;LineItemPrice1;LineItemOrderNumber2;LineItemProduct2;LineItemPrice2

John;18032015;O-2000;2015-01-10;O-2000;Audi;30.000;O-2000;Mercedes;40.999;O-2001;2015-01-11;O-2001;Porsche;130.000;O-2001;BMW;35.999
```

CsvConverter configuration

```
<?xml version="1.0" encoding="utf-8"?>
<csv2xml>
<configuration name="default">
  <fielddelimiters>;</fielddelimiters>
  <sourcefileencoding>windows-1250</sourcefileencoding>
  <textqualifier>"</textqualifier>
  <sourcefilecontainsheaderrow>y</sourcefilecontainsheaderrow>
  <lineseparator>&#xD;&#xA;</lineseparator>
   <xmloutput name="default" >
     <xmlelement name="Customer" isdefault="y" >
```

```
      <xmlelement name="Order" isdefault="n" >
       <xmlattribute name="OrderNumber" csvcolumnname="OrderNumber\d{1,}"/>
       <xmlattribute name="OrderDate" csvcolumnname="OrderDate\d{1,}"/>
         <xmlelement name="OrderLineItem" isdefault="n">
           <xmlattribute name="LineItemOrderNumber" csvcolumnname="LineItemOrderNumber\d{1,}"/>
           <xmlattribute name="LineItemProduct" csvcolumnname="LineItemProduct\d{1,}"/>
           <xmlattribute name="LineItemPrice" csvcolumnname="LineItemPrice\d{1,}"/>
           <relation>
             <parentattribute name="OrderNumber"/>
             <childattribute name="LineItemOrderNumber"/>
           </relation>
         </xmlelement>
       </xmlelement>
     </xmlelement>
   </xmloutput>
</configuration>
</csv2xml>

Generated XML output file
<?xml version="1.0" encoding="utf-8"?>
 <Customer CustomerName="John" CustomerId="18032015">
  <Order OrderNumber="O-2000" OrderDate="2015-01-10">
  <OrderLineItem LineItemProduct="Audi" LineItemPrice="30.000" />
  <OrderLineItem LineItemProduct="Mercedes" LineItemPrice="40.999" />
 </Order>
 <Order OrderNumber="O-2001" OrderDate="2015-01-11">
  <OrderLineItem LineItemProduct="Porsche" LineItemPrice="130.000" />
  <OrderLineItem LineItemProduct="BMW" LineItemPrice="35.999" />
 </Order>
</Customer>
```

Known Limitations

Following are the known limitations.

- Line breaks in column values are not supported.
- Invalid encoding format in the configuration file is not supported, but it can generate the XML file with respect to its nearest matching encoding style.
- If any column names starts or contains numbers or special characters, those characters are represented in the XML file with their corresponding Hexadecimal value.
- In text qualifier handling, the application treats delimiter characters as delimiter when the number of double quotes are even.
- Two consecutive double quotes are replaced by one double quotes, three consecutive double quotes are replaced by two consecutive double quotes and so on.
- If the CSV source files contains data with master-detail relationship, it must have a header line containing the column names.
- Composite keys for relations are not supported - a key can consists only of one column.

CsvConverter Functions

Init

Initializes the CsvConverter component. This function must be the first function called before using any other functions of CsvConverter component. This method validates the configuration passed for the conversion.

Init	
SYNTAX	Init()
EXAMPLE	csvConverter = ModusProcess.GetComponent("CsvConverter") csvConverter.Init()

IsInitialized

This function checks whether the CsvConverter component is initialized.

IsInitialized	
SYNTAX	IsInitialized
RETURN	TRUE: The component is initialized. FALSE:The component is not initialized.

EXAMPLE	csvConverter = ModusProcess.GetComponent("CsvConverter") csvConverter.Init() status = csvConverter.IsInitialized

Run

This function converts the valid input source CSV or TXT file to XML file according to configuration set during conversion.

Run		
SYNTAX	Run(inputFileName)	
ARGUMENTS	inputFileName	Name of the CSV or TXT file to convert.
EXAMPLE	csvConverter = ModusProcess.GetComponent("CsvConverter") csvConverter.Init() csvConverter.Run("C:\inputFile.csv")	

Sample Script CsvConverter

```
csvConverter = ModusProcess.GetComponent("CsvConverter")
csvConverter.Init()
; You can specify the output directory in the following cases.
;   - At design time in the Content Composer Studio editor
;   - In the Monalisa script (if needed)
;     Note The value set in the script overrides the setting made at design time.
csvConverter.OutputDirectory = "d:\temp\CVSConverter\out"
GetObject("DOS", "MLDos")
; path to search for inputfiles
path = "d:\temp\CSVConverter\"
; pattern to search for
searchPattern = "*.csv"
```

```
; get all matching filenames
FileList = DOS.GetFiles(path, searchPattern)
InputFile = StrTok("FileList", ",")
ExecuteLoop = ((InputFile <> "") AND (DOS.FileExists(InputFile)) AND (csvConverter.IsInitialized))
while (ExecuteLoop)
try
Protocol("start conversion of file '{0}'", 5, InputFile)
csvConverter.Run(InputFile)
OnError()
;report error and continue with next file
Protocol("error trying to convert file '{0}': {1} ", 0, InputFile, Error.Message)
End-Try
InputFile = StrTok("FileList", ",")
ExecuteLoop = ((InputFile <> "") AND (DOS.FileExists(InputFile)))

;end while-loop if Content Composer process was asked to stop running
ExecuteLoop = ExecuteLoop and ((ModusProcess.State <>
ModusSuite.Runtime.Types.StateType.StopPending) and (ModusProcess.State <>
ModusSuite.Runtime.Types.StateType.CancelPending))
end-while
```

DataSelectionManager

About DataSelectionManager

You must have an Xdata license installed to use the **DataSelectionManager** component.

This component enables you to execute data retrieval defined by an Xdata selection in a process.

Properties of the DataSelectionManager

Property	Description
DebugMode	TRUE = Run the selection in the debugger. FALSE = Do not run the selection in the debugger.

LastError	Returns the error message of the last executed data collection as a string. An empty string is returned when no error exists.
NavigatorXml	If the function GetDataExtended is called with the parameter writeNavigatorXml = True, you can access the navigation XML here. For more information, refer to GetDataExtended.
SelectionReference	Defines the selection the component executes. Selection parameters pass on with the help of system pool variables. For more information, refer to the sample script.
SystemOId	If a system OID is set, the selection is either taken from the corresponding system or from the same system as the process itself.
ElapsedMilliseconds	Returns the duration of last executed data collection in elapsed milliseconds as long.
XmlData	Returns the XML of the last executed data collection as a string.

DataSelectionManager Functions

GetData

Executes data retrieval.

GetData		
SYNTAX	GetData(raiseError)	
ARGUMENTS	raiseError	TRUE = An error is thrown if a reading script tries to read a data object with SelVariable that does not exist. FALSE = No error is thrown if a reading script tries to read a data object with SelVariable that does not exist.
RETURN	<> 0 if an error occurred.	

GetDataExtended

Executes data retrieval.

GetDataExtended

SYNTAX GetDataExtended(raiseError, writeNavigatorXml)

ARGUMENTS

raiseError	TRUE = An error is thrown if a reading script tries to read a data object with SelVariable that does not exist. FALSE = No error is thrown if a reading script tries to read a data object with SelVariable that does not exist.
writeNavigatorXml	TRUE = The navigation schema is written to the NavigatorXml property when data retrieval executes. FALSE= No navigation schema is written.

RETURN <> 0 if an error occurred.

LoadSelection

Sets the selection reference used by the next data retrieval.

LoadSelection

SYNTAX LoadSelection(selectionReference)

ARGUMENTS

selectionReference	Selection reference.

LoadSelectionByName

Loads a selection specified by name and optionally by type.

LoadSelectionByName

SYNTAX LoadSelectionByName(selectionName, objectType)

ARGUMENTS

selectionName	Name of the selection.

	objectType	**Possible values** • ComposerSuiteObjectType.Selection (default value if the parameter is not specified) • ComposerSuiteObjectType.XsdSelection This parameter is optional. The default value is ComposerSuiteObjectType.Selection.
EXAMPLE	LoadSelectionByName("Invoice") LoadSelectionByName("Invoice", ComposerSuiteObjectType.XsdSelection)	

LoadSelectionByOId

Loads a selection specified by object ID and optionally by type.

LoadSelectionByOId

SYNTAX LoadSelectionByOId(selectionOId, objectType)

ARGUMENTS

selectionOId	Object ID of the selection.
objectType	**Possible values** • ComposerSuiteObjectType.Selection (default value if the parameter is not specified) • ComposerSuiteObjectType.XsdSelection This parameter is optional. The default value is ComposerSuiteObjectType.Selection.

EXAMPLE LoadSelectionByOId("15072011-226-sc-ts")

LoadSelectionByOId("15072011-226-sc-ts", ComposerSuiteObjectType.XsdSelection)

WriteXml

Saves the XML from the last data collection to a file.

TrimStart	
SYNTAX	WriteXml(fileName, prettyXml)
ARGUMENTS	fileName — Name of the file the XML saves prettyXml — Defines whether the XML saves in a readable (indented) format
RETURN	<> 0 if an error occurred.
EXAMPLE	Selection = ModusProcess.GetComponent("DataSelectionManager") Selection.GetData(true) Selection.WriteXml("c:\sel.txt", true)

Sample Script DataSelectionManager

```
SelManager = ModusProcess.GetComponent("DataSelectionManager")

;Insert selection parameter in the System pool
 System.CustomerId = "0815"

;Execute selection
 Res = SelManager.GetData(False)

;Save xml result into a file
 OutputFileName =  "c:\temp\" + System.CustomerId + ".txt"
 SelManager.WriteXml(OutputFileName, true)

; Assign xml result to a local variable
 myStringVariable = SelManager.XmlData
```

DocumentCollectionManager

About DocumentCollectionManager

This component enables you to generate documents from a bundle and to forward them to other downstream systems in a process.

Properties of the DocumentCollectionManager

Property	Description

BundleSystemOId	Reads or sets the system OID of the system from which the bundle loads.
DocumentCollection	Specifies the bundle you want to create.
MissingObjects	Comma-separated string containing the objects not found during document generation. An empty string is returned when no missing objects exist.
InsertTextComponentIndicators	TRUE = Insert text block indicators. FALSE = Do not insert text block indicators.
InsertCustomUI	TRUE = Assign a CustomUI object to an inserted document. FALSE = Do not assign a CustomUI object.
IsDocumentCollectionLoaded	TRUE = A bundle was loaded. FALSE= No bundle was loaded.
DestinationDirectory	Target directory for document generation.

DocumentCollectionManager Functions

AssignData

Assigns data to a data reference.

AssignData

SYNTAX AssignData(dataId, xmlData)

ARGUMENTS

dataId	Data ID
xmlData	XML

EXAMPLE

```
AssignData(System.DataId, System.LastXml)
```

CreateDocuments

Executes document generation.

CreateDocuments	
SYNTAX	CreateDocuments()

DefinePrinterForBundleByName

Defines a printer and print time for all references of the bundle.

DefinePrinterForBundleByName

SYNTAX

DefinePrinterForBundleByName(printerName, printerType, printTime)

DefinePrinterForBundleByName(printerName, printerType, printTime, formName)

ARGUMENTS

printerName	Printer name.
printerType	Printer type.

Possible values	Description
ComposerSuiteObjectType.Printer	Online printer.
ComposerSuiteObjectType.EMailPrinter	Email printer.
ComposerSuiteObjectType.OutsourcingPrinter	Outsourcing printer.

printTime — Printing time.

Possible values	Description
OdinPrintTime.Direct	Immediately
OdinPrintTime.Shifted	Shifted

OdinPrintTime.Unknown	Unknown
OdinPrintTime.Local	Local

formName	Form name.

GetDataReferenceEnumerator

Returns an enumerator containing all text data references.

GetDataReferenceEnumerator

SYNTAX GetDataReferenceEnumerator()

RETURN Enumerator with the following properties and functions.

Objectname	Object name.
IsAssigned	TRUE = Data is already assigned to the data reference. FALSE = No data is assigned to the data reference.
IsActiveDataSelection	TRUE = Active selection (data is actively retrieved). FALSE = Not an active selection.
IsPassiveDataSelection	TRUE = Passive selection (data is not actively retrieved). FALSE = Not a passive selection.
Assign(xml)	Assigns data to the data reference. xml: XML string.
GetXml()	Returns the data XML of this data reference or an empty string if no data is assigned.

GetManualVariables

Returns an XML string containing the manual variables of all original documents.

GetManualVariables	
SYNTAX	GetManualVariables()
RETURN	XML string.

GetSelectionParameters

Returns an XML string containing the selection parameters of all data references.

GetSelectionParameters	
SYNTAX	GetSelectionParameters()
RETURN	XML string containing the selection parameters.

LoadByAlias

Loads a bundle by an alias as defined in an alias table.

LoadByAlias		
SYNTAX	LoadByAlias(aliasTableName, alias)	
ARGUMENTS	aliasTableName	Name of the alias table.
	alias	Alias identifier.

LoadById

Loads a bundle by its object ID from the same system the process resides in.

LoadById

SYNTAX	LoadById(objectId)
ARGUMENTS	objectId — Object ID.

LoadByName

Loads a bundle by its name from the same system the process resides in.

LoadByName

SYNTAX	LoadByName(objectName)
ARGUMENTS	objectId — Object name.

PrintAndArchive

Prints and archives documents or forwards them to the downstream output system.

PrintAndArchive

SYNTAX

PrintAndArchive(processId, processType, processTitle)

PrintAndArchive(processId, processType, processTitle, bundlePrintSequence)

ARGUMENTS

processId	Process ID.
processType	Process type.
processTitle	Process title.
bundlePrintSequence	**Possible values** BundlePrintSequence.BundleSequence = Default setting if the parameter is not defined. Originals and copies pass on to Odin in the same order as they are positioned in the bundle. BundlePrintSequence.OriginalsFirst = First passes all originals on to Odin and then the copies.

RETURN	Composer process info XML (Import XML for Odin) as XML string.

ProcessData

Executes the AfterDataRetrieval script.

ProcessData	
SYNTAX	ProcessData()

ResetDocuments

Resets the status of all original documents to **not inserted**.

ResetDocuments	
SYNTAX	ResetDocuments()

SetManualVariables

Sets manual variables using an XML string.

SetManualVariables		
SYNTAX	SetManualVariables(manualVariables)	
ARGUMENTS	manualVariables	XML string

SetSelectionParameters

Sets selection parameters using an XML string.

SetSelectionParameters	
SYNTAX	SetSelectionParameters(selectionParameters)

ARGUMENTS	selectionParameters	XML string

Sample Script DocumentCollectionManager

```
batchManager = ModusProcess.GetComponent("ModusBatchManager")
DocumentCollection = ModusProcess.GetComponent("DocumentCollectionManager")
;
;Load bundle by specifying the name
bundleName = batchManager.DocumentCollection
documentCollection.BundleSystemOId = batchManager.SystemOId
documentCollection.LoadByName(bundleName)
;
documentCollection.DestinationDirectory = batchManager.Directory
```

FlatFileConverter

About FlatFileConverter

The FlatFileConverter component can convert text files with fixed record length to an XML file.

Properties of FlatFileConverter

The following table describes the properties of FlatFileConverter.

Property	Description
Configuration	An XML string that specifies the settings to read the input file and create the XML output file. The editor available to edit this XML offers the following toolbar buttons. To create or modify the configuration, complete the following steps. • **New**. Click **New** to display a default configuration template that the

	user can modify. • **Open**. Click **Open** to display a dialog box to load an existing configuration file into the editor.
ConfigurationName	It is possible to define more than one “parserdefinition” element in the configuration XML file each having a unique name. The name of the “parserdefintion” is selected with this property.
Output directory	The directory path where the generated XML file is created.

FlatFileConverter Events

Event	Description
BeforeSaveXML:	This event is executed before saving the created XML to a file. You can skip to write the XML file in the attached script by setting the contextvariable “SaveXmlToFile” to `FALSE`. For more information, you can refer the Properties of context FlatFileConverter. topic.

FlatFileConverter Configuration

The following section describes the XML structure to specify the conversion from a flat file to XML structure.

The parser definition has the following basic XML structure:

```
<parserdefinitions
  <parserdefinition
    <traceconfig
    </traceconfig
    <CustomRootNode
    </CustomRootNode
  </parserdefinition
</parserdefinitions
```

The following table displays the XML elements required for FlatFileConverter configuration.

XML element	Description
parserdefinitions	This is the XML root element.
parserdefinition	This element contains several XML elements that specify how to interpret the input file and define the XML output structure. You can define 1-n parserdefinition elements, where each definition must have a unique name.
CustomRootNode	CustomRootNode is the most important child element. This is the XML root element of the desired output XML structure.

Attribute	Description
name	The unique name of a parserdefinition element. • If only one element exists, it must be “default”. • If multiple parserdefinition elements are defined, the name of the one to use for processing is set with the flat file component attribute '
doneextension	This optional attribute specifies the file extension for input files that are successfully processed. If the attribute value is an empty string or is missing, the default value “fin” is used.
id	This identifier specifies the **ATTRIBUTE**-element of the **CustomRootNode** that contains a unique ID. The converter handles all lines having the same ID as one data record. For example, IDs for Customer ID or Contract ID are as follows. • **ignoreemptyvalues**. This attribute specifies if the validity should be checked also for empty values of the node. **Possible values** • **Y:**. Test for valid values are not executed in case an empty string was extracted for this node from input file. This is the default value.

- **N**: The check if the node contains a valid value is executed.

- **sourcecodepage.** This specifies the encoding used to read the input file, for example **windows-1250**. If the attribute value is not specified, the default code page of the used windows system is used.

  You can use the following values as encoding parameters.

  - Empty string. In this case, the default encoding (ANSI code page of the operating system) is used.
  - Code page, for example, "1252"
  - Name, for example, "windows-1252"
  - Friendly name, for example, "Western European" (Windows)

  Possible encoding parameters

Code page	name	Display name
37	IBM037	IBM EBCDIC (USA-Canada)
437	IBM437	OEM USA
500	IBM500	IBM EBCDIC (International)
708	ASMO-708	Arabic (ASMO 708)
720	DOS-720	Arabic (DOS)
737	ibm737	Greek (DOS)
775	ibm775	Baltic (DOS)
850	ibm850	Western European (DOS)
852	ibm852	Eastern European (DOS)
855	IBM855	OEM Cyrillic

857	ibm857	Turkish (DOS)
858	IBM00858	OEM Multilingual Latin I
860	IBM860	Portuguese (DOS)
861	ibm861	Icelandic (DOS)
862	DOS-862	Hebrew (DOS)
863	IBM863	French, Canada (DOS)
864	IBM864	Arabic (864)
865	IBM865	Nordic (DOS)
866	cp866	Cyrillic (DOS)
869	ibm869	Greek, modern (DOS)
870	IBM870	IBM EBCDIC (Multilingual Latin-2)
874	windows-874	Thai (Windows)
875	cp875	IBM EBCDIC (Greek, modern)
932	shift_jis	Japanese (Shift-JIS)
936	gb2312	Simplified Chinese (GB2312)
949	ks_c_5601-1987	Korean
950	big5	Chinese (Traditional) (Big5)
1026	IBM1026	IBM EBCDIC (Turkish, Latin-5)
1047	IBM01047	IBM Latin-1
1140	IBM01140	IBM EBCDIC (USA-Canada-European)
1141	IBM01141	IBM EBCDIC (Germany-European)

1142	IBM01142	IBM EBCDIC (Denmark-Norway-European)
1143	IBM01143	IBM EBCDIC (Finland-Sweden-European)
1144	IBM01144	IBM EBCDIC (Italian-European)
1145	IBM01145	IBM EBCDIC (Spain-European)
1146	IBM01146	IBM EBCDIC (Great Britain-European)
1147	IBM01147	IBM EBCDIC (France-European)
1148	IBM01148	IBM EBCDIC (International-European)
1149	IBM01149	IBM EBCDIC (Icelandic-European)
1200	utf-16	Unicode
1201	unicodeFFFE	Unicode (Big Endian)
1250	windows-1250	Central European (Windows)
1251	windows-1251	Cyrillic (Windows)
1252	Windows-1252	Western European (Windows)
1253	windows-1253	Greek (Windows)
1254	windows-1254	Turkish (Windows)
1255	windows-1255	Hebrew (Windows)
1256	windows-1256	Arabic (Windows)
1257	windows-1257	Baltic (Windows)
1258	windows-1258	Vietnamese (Windows)

1361	Johab	Korean (Johab)
10000	macintosh	Western European (Mac)
10001	x-mac-japanese	Japanese (Mac)
10002	x-mac-chinesetrad	Chinese traditional (Mac)
10003	x-mac-korean	Korean (Mac)
10004	x mac-arabic	Arabic (Mac)
10005	x-mac-hebrew	Hebrew (Mac)
10006	x-mac-greek	Greek (Mac)
10007	x-mac-cyrillic	Cyrillic (Mac)
10008	x-mac-chinesesimp	Chinese simplified (Mac)
10010	x-mac-romanian	Romanian (Mac)
10017	x-mac-ukrainian	Ukrainian (Mac)
10021	x-mac-thai	Thai (Mac)
10029	x-mac-ce	Central European (Mac)
10079	x-mac-icelandic	Icelandic (Mac)
10081	x-mac-turkish	Turkish (Mac)
10082	x-mac-croatian	Croatian (Mac)
12000	utf-32	Unicode (UTF-32)
12001	utf-32BE	Unicode (UTF-32-Big Endian)
20000	x-Chinese-CNS	Chinese traditional (CNS)
20001	x-cp20001	TCA Taiwan
20002	x-Chinese-Eten	Chinese traditional (Eten)
20003	x-cp20003	IBM5550 Taiwan
20004	x-cp20004	TeleText Taiwan
20005	x-cp20005	Wang Taiwan
20105	x-IA5	Western European

		(IA5)
20106	x-IA5-German	German (IA5)
20107	x-IA5-Swedish	Swedish (IA5)
20108	x-IA5-Norwegian	Norwegian (IA5)
20127	us-ascii	US-ASCII
20261	x-cp20261	T.61
20269	x-cp20269	ISO-6937
20273	IBM273	IBM EBCDIC (Germany)
20277	IBM277	IBM EBCDIC (Denmark-Norway)
20278	IBM278	IBM EBCDIC (Finland-Sweden)
20280	IBM280	IBM EBCDIC (Italy)
20284	IBM284	IBM EBCDIC (Spain)
20285	IBM285	IBM EBCDIC (Great Britain)
20290	IBM290	IBM EBCDIC (Japanese Katakana)
20297	IBM297	IBM EBCDIC (France)
20420	IBM420	IBM EBCDIC (Arabic)
20423	IBM423	IBM EBCDIC (Greek)
20424	IBM424	IBM EBCDIC (Hebrew)
20833	x-EBCDIC-KoreanExtended	IBM EBCDIC (Korean, extended)
20838	IBM-Thai	IBM EBCDIC (Thai)
20866	koi8-r	Cyrillic (KOI8-R)
20871	IBM871	IBM EBCDIC (Icelandic)

20880	IBM880	IBM EBCDIC (Cyrillic, Russian)
20905	IBM905	IBM EBCDIC (Turkish)
20924	IBM00924	IBM Latin-1
20932	EUC-JP	Japanese (JIS 0208-1990 and 0212-1990)
20936	x-cp20936	GB2312-80 Chinese (simple)
20949	x-cp20949	Korean Wansung
21025	cp1025	IBM EBCDIC (Cyrillic, Serbian-Bulgarian)
21866	koi8-u	Cyrillic (KOI8-U)
28591	iso-8859-1	Western European (ISO)
28592	iso-8859-2	Central European (ISO)
28593	iso-8859-3	Latin 3 (ISO)
28594	iso-8859-4	Baltic (ISO)
28595	iso-8859-5	Cyrillic (ISO)
28596	iso-8859-6	Arabic (ISO)
28597	iso-8859-7	Greek (ISO)
28598	iso-8859-8	Hebrew (ISO-Visual)
28599	iso-8859-9	Turkish (ISO)
28603	iso-8859-13	Estonian (ISO)
28605	iso-8859-15	Latin 9 (ISO)
29001	x-Europa	Europe
38598	iso-8859-8-i	Hebrew (ISO-Logical)
50220	iso-2022-jp	Japanese (JIS)
50221	csISO2022JP	Japanese (JIS, 1

		Byte Kana allowed)
50222	iso-2022-jp	Japanese (JIS, 1 Byte Kana allowed - SO/SI)
50225	iso-2022-kr	Korean (ISO)
50227	x-cp50227	ISO-2022 Chinese (simplified)
51932	euc-jp	Japanese (EUC)
51936	EUC-CN	Chinese simplified (EUC)
51949	euc-kr	Korean (EUC)
52936	hz-gb-2312	Chinese simplified (HZ)
54936	GB18030	GB18030 Chines simplified
57002	x-iscii-de	ISCII Devanagari
57003	x-iscii-be	ISCII Bengali
57004	x-iscii-ta	ISCII Tamil
57005	x-iscii-te	ISCII Telugu
57006	x-iscii as	ISCII Assamese
57007	x-iscii-or	ISCII Oriya
57008	x-iscii-ka	ISCII Kannada
57009	x-iscii-ma	ISCII Malayalam
57010	x-iscii-gu	ISCII Gujarati
57011	x-iscii-pa	ISCII Punjabi
65000	utf-7	Unicode (UTF-7)
65001	utf-8	Unicode (UTF-8)

jobelementpath and multiselmaxcontent count	These two attributes specify if one or many XML output files are created from one input file and the name of the XML root element. • If the value of attribute **jobelementpath** is an empty string or missing, every data record

contained in the input file is written into a single XML output file (one XML output file contains one data record).

- The value of attribute **multiselmaxcontentcount** is ignored in this case.
- If the value of attribute **jobelementpath** contains a string that has the format "/" + [root element name] + "/" + element name], then multiple data records contained in the input file are written to a single XML output file.
- The value of [root element name] specifies the name of the XML root element. The value of [element name] specifies the name of element that is the child of the root element and contains one data record.
- The value of the attribute **multiselmaxcontentcount** specifies the maximum number of data records that are written into one XML output file.
- If the input file contains more data records than the number specified with **multiselmaxcontentcount**, more than one XML output file is created.
- If the value of **multiselmaxcontentcount** is zero, all data records are written into one XML output file.

Defining Data Conversion

The parserdefinition contains a hierarchical structure of XML elements that specifies how to extract data from the input file and the desired XML structure of the XML output.

The root element of this structure is the CustomRootNode.

The structure of this XML file corresponds roughly to the desired XML result, however, there may be elements that are used only for internal purposes and will not be included in the result later. The following example displays the different scenarios. The name of the CustomRootNode in this example is “Customer”.

```
<parserdefinition name="default" >
  <Customer type="ELEMENT" source="LINES" keypos="1" keylength="8" >
```

```
    <CustomerId type="ATTRIBUTE" pos="1" length="8" />
  </Customer>
</parserdefinition>
```

The following XML output is generated when the input file is processed.

```
<Customer CustomerId="18032015" />
```

So the desired XML element names and the XML structure are specified in the parser definition by using these names and structure. The name of a node corresponds to the identifier under which it is inserted. The following attributes are available to specify, read, validate, modify, and write output XML.

- **type**. This specifies the type of XML node which is created in the output XML. The following are the possible values.
  - **ELEMENT**. An XML element is created.
  - **ATTRIBUTE**. An XML attribute is created.
  - **CHARDATA**. The value is written as the content of the parent node.
- **source,sourcenode**. This specifies the kind of data source in the input file. The default setting is the value STRING. The following are the possible values for the source.
  - **LINES**. Multiple lines are read from the input file.
  - **LINE** . One line from the input file is assigned.
  - **STRING**. A string from within a line or from within another string is read.
  - **PARENT**. The data of the parent node is taken.

The following table shows the valid values for a parentnode's source for a given value.

value	parentnode
LINES	LINE
LINE	LINES
STRING	LINES, LINE, STRING, PARENT
PARENT	STRING

The root element of the desired output XML structure must be defined with the following structure.

type="ELEMENT" and source="LINES".

If for a node the string determination is defined by using the attribute "sourcenode", this node is used as the source for the string. The value for "sourcenode" is the name of the desired node. First, the specified node is searched at the level of the current node. If this fails, all parent nodes are searched until the node is found. If this fails too, an error occurs.

- **pos, length**. These attributes are used to specify the position and length to read data from the input line. To start from the first character the value one for pos.
- **insert**. This attribute specifies if the defined node is created in the output XML. The following are the possible values.
  - **“Y”**. Node is created in XML file.
  - **“N”**. Node is not created.
- **validvalues, ecinvalidvalue, ignoreemptyvalues**. For a node with source=”STRING” it is possible to define simple validity checks and conditions. With the attribute "validvalues", a comma separated valid character string can be specified for a node. If the value assigned to the node is not equal to any of these specified valid values, an error occurs.
  - The attribute **ecinvalidvalue** specifies an errorcode used for tracing.
  - The attribute **ignoreemptyvalues** defines, whether the validity should be checked for empty values of the node.
  - The flag **ignoreemptyvalues** defaults to true. It can be specified globally as an attribute of the parserdefinition node and also on a inner node.
- **conditions**. For nodes of source=”STRING”, it is possible to define conditions so that inserting a node is dependent of the values of other nodes. Therefore, one or more conditions are specified in the form of conditions="node1=A, node2!=B". All of these conditions must be met, before the node is inserted. Values can be compared with equal (=) and not equal (!=) operator.
- **maxoccurs**. This attribute can be used for nodes with source=”LINES” or nodes having type="ELEMENT" and source="STRING". For these nodes this attribute specifies how many times this node is inserted into the output XML. Specify the value “UNBOUND” for a 1-n relation.

Process at runtime

The following sections describe how the defined nodes are processed during conversion.

Process LINE and LINES

For nodes with source LINES and LINE, the additional attributes **keypos**, **keylength** and **key** are expected. For source LINES, specifying the attribute **key** is optional.

Without giving any **key** all lines that are identical from the point **keypos** with the length **keylength** are assigned to this node.

For the attribute **key**, one or more comma separated values can be specified. Only those lines having corresponding **key** from the position **keypos** with length **keylength** are assigned to the node.

A node with source=”LINES” can be inserted only once in the result.

Optionally, an additional **startkey** with **startkeypos** and **startkeylength** and a similar **stopkey** can be defined.

These keys can be used in case it is not possible to recognize the definition of the key when a data record is finished. You can specify both or only one of the start or stop key.

Data acquisition for LINE

From the parent node with source=”LINES”, the line from the point **keypos** with length **keylength** a value of **key** is assigned.

A line can occur multiple times and thus this node is inserted more than once in the XML result. In this case, the node has to be defined as a 1-n relationship by setting attribute **maxoccurs**="UNBOUNDED". This relationship is also considered when creating the XML schema.

Data acquisition STRING

In this case, a string is simply read from a parent string or line. For this purpose, the details supplied with **pos** and **length** are used. When reading from a LINES node, it is always read from the first row.

If for a node the string determination is defined by using the attribute **sourcenode**, this node is used as the source for the string. The value for **sourcenode** is the name of the desired node. First, the specified node is searched at the level of the current node. If this fails, all parent nodes are searched until the node is found. If this fails too, an error occurs.

Nodes having type="ELEMENT" and source="STRING" can also be processed more than once. If at such a node the attribute **count** is specified, the node is added repeatedly up to this value. After each insertion, the position to determine the data is increased internally by the value of the attribute **length**. Specifying **maxoccurs** (either UNBOUNDED or exact value) here also indicates the 1-n relationship.

If the value of **count** is outside 0-**maxoccurs**, an error occurs. The attribute **ecinvalidvalue** allows you to specify an error code for tracing. For a similar node, the optional attribute **indexattribute** is valid. This optional attribute defines an attribute for this node and at runtime the current counter of the multi-node is written. The **indexattribute** should exist as a subnode in the definition, so that it is taken into account, when generating the XML schema. The value of count can be changed at runtime. Of course, this only makes sense if **count** is read from another node (see below) instead of being a constant value.

With the attribute **countaddend** a summand or addend (negative or positive) for the value of **count** is specified. The attribute values of **count**, **pos** and **length** must be numeric. But they also can begin with an @ sign. After the @ sign the name of a node with source=”STRING” must be specified. At runtime, that node’s value is taken as value for the according attribute.

The node is determined the same way as described for **sourcenode** (see above).

Validity checks

For nodes of source=”STRING”, it is possible to define simple validity checks and conditions. With the attribute **validvalues** comma separated valid character strings can be specified for a node. If the value assigned to the node is not equal to any of these specified valid values, an error occurs.

Using the attribute **conditions** you can define, whether inserting a node is dependent of the values of other nodes. Therefore, one or more conditions are specified in the form of conditions="node1=A, node2!=B".

- All of these conditions must be met, before the node is inserted.
- Values can be compared with equal (=) and not equal (!=) operator
- The attribute **ecinvalidvalue** allows you to specify an errorcode used for tracing.
- The attribute **ignoreemptyvalues** defines, whether the validity should be checked for the empty values of the node.
- The flag **ignoreemptyvalues** defaults to `true` and can be set globally as an attribute of the **parserdefinition** node.

Process data

For nodes of source=”STRING”, you can modify the input data before it is written to the output XML. This is done by using the attribute functions. Its value may contain a comma separated list of function-calls, which are executed in the order they were specified. Currently, the following function is currently available.

- **InsertLine**. Replaces the character "#" with the string "\line".

Insert data

To write data to the XML files the nodes of type=”ATTRIBUTE” or “CHARDATA” are used. Leading or trailing blank spaces are deleted previously. For nodes of type=”CHARDATA” the attribute 'linelength' can be used to specify a number. This results in inserting the linelength count characters with the string "\line".

Example

This is an example for an input file where the following relationships are stored in each data row.

Customer

Order (1-n)

OrderLineItem (1-n)

Input data

18032015Doe O-20002015-01-10Audi 30.000

18032015Doe O-20002015-01-10Mercedes 40.999

18032015Doe O-20012015-01-11Porsche 130.000

18032015Doe O-20012015-01-11BMW 35.999

The input file has the following structure.

field name	beginning position	length

CustomerID	1	8
CustomerName	9	15
orderNumber	24	6
orderDate	30	10
LineitemProduct	40	16
LineItemPrice	57	9

- CustomerId is used as an ID to get all lines belonging to one customer.
- OrderNumber is used to collect all lines belonging to one order. As the order number is not constant, it is chosen to use first two characters “O-”, identifying an order, as key to get all OrderLineitems belonging to one order (see OrderLineItem).

Configuration

```
<parserdefinitions>
 <parserdefinition name="default" id="CustomerId" ignoreemptyvalues="Y" >
   <Customer type="ELEMENT" source="LINES" keypos="1" keylength="8" >
   <CustomerId type="ATTRIBUTE" pos="1" length="8" />
   <CustomerName type="ATTRIBUTE" pos="9" length="15" />
    <Order type="ELEMENT" source="LINES" keypos="24" keylength="6" maxoccurs="UNBOUNDED" >
      <OrderNumber type="ATTRIBUTE" pos="24" length="6" />
      <OrderDate type="ATTRIBUTE" pos="30" length="10" />
      <OrderLineItem type="ELEMENT"  source="LINE" key="O-" keypos="24" keylength="2"
                        maxoccurs="UNBOUNDED" >
        <LineItemProduct type="ATTRIBUTE" pos="40" length="16"  />
        <LineItemPrice type="ATTRIBUTE" pos="57" length="9"  />
       </OrderLineItem>
     </Order>
   </Customer>
 </parserdefinition>
</parserdefinitions>
```

Generated output XML file

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Customer CustomerId="18032015" CustomerName="Doe">
 <Order OrderNumber="O-2000" OrderDate="2015-01-10">
  <OrderLineItem LineItemProduct="Audi" LineItemPrice="30.000" />
  <OrderLineItem LineItemProduct="Mercedes" LineItemPrice="40.999" />
 </Order>
 <Order OrderNumber="O-2001" OrderDate="2015-01-11">
  <OrderLineItem LineItemProduct="Porsche" LineItemPrice="130.000" />
  <OrderLineItem LineItemProduct="BMW" LineItemPrice="35.999" />
 </Order>
</Customer>
```

- **traceconfig**. This XML element is optional. It contains settings to customize the error message.
- **traceline**. The element 'traceline' describes the structure of an error message that is written to the Content Composer logfile using the TraceSource “Xdata”. The node errorlist describes error codes and related error messages. The definition of the attribute 'unknownerrorcode' is important. The error code listed here is used for unknown errors. The error codes used in certain situations are described in the Process at runtime topic and can be defined directly in a parsernode definition.

FlatFileConverter Functions

Init

This function initializes the component. This function must be the first function called before using any other functions of this component. This method validates the configuration used for the conversion.

Init	
SYNTAX	Init()
EXAMPLE	FlatFileConverter = ModusProcess.GetComponent("FlatFileConverter ") FlatFileConverter.Init()

IsInitialized

This function checks whether the FlatFileConverter component is initialized.

IsInitialized	
SYNTAX	IsInitialized
RETURN	TRUE. The component is initialized. FALSE. The component is not initialized
EXAMPLE	FlatFileConverter = ModusProcess.GetComponent("FlatFileConverter") FlatFileConverter.Init() ok = FlatFileConverter.IsInitialized

Run

This function converts the input file to an XML file according to the configuration set during conversion.

Run		
SYNTAX	Run(inputFileName)	
ARGUMENTS	inputFileName	Name of the input file to convert.
EXAMPLE	FlatFileConverter = ModusProcess.GetComponent("FlatFileConverter") FlatFileConverter.Init() FlatFileConverter.Run(“c:\inputFile.txt”)	

SampleScriptFlatFileConverter

```
Run(inputFileName)
inputFileName. Name of the input file to convert
converter = ModusProcess.GetComponent("FlatFileConverter")
converter.Init()
SourceFile = "d:\test\FixedPosition_Example_MasterDetail_Input.txt"
converter.Run(SourceFile)
```

ModusBatchManager

About ModusBatchManager

The ModusBatchManager component enables you to generate documents in batch processes.

Properties of the ModusBatchManager

Property	Description
BeginJobScript	Start script for the preparation of document creation.
DB Alias	Database alias for the virtual database within the component.
EndJobScript	Script for the end processing of document creation.
ProcessRootDirectory	Directory in which the generated bundles are saved.

ModusBatchManager Functions

DoWork

Executes the BatchManager start script.

DoWork		
SYNTAX	DoWork() DoWork(string xml)	
ARGUMENTS	xml	XML data assigned to the BatchManager context.
RETURN	TRUE = Successful FALSE = Unsuccessful	
EXAMPLE	batchManager.DoWork()	

FinishWork

Executes the BatchManager end script.

FinishWork	
SYNTAX	FinishWork() FinishWork(xml)
ARGUMENTS	xml XML data assigned to the BatchManager context.
EXAMPLE	batchManager.FinishWork()

MwsActivityStatus

These components are only available in MWS processes and are responsible for storing information on process activities (states).

An MwsActivityStatus is identified by a unique object name.

Scripts able to use the Context MwsProcess can access MWSActivity with the command **GetActivityStatus**.

MwsAssistantManager

About MwsAssistantManager

The MwsAssistantManager component enables you to delete finished MWS processes.

Properties of the MwsAssistantManager

Property	Description
Condition	Condition defining the processes to be deleted in the following form. MWS_STATUS=2 In this example, all processes with the value 2 in the MWS_STATUS field are deleted.
DB Alias	Database alias enabling use of the virtual database inside this component.

MwsAssistantManager Functions

DeleteExpiredSessions

Use this function to delete expired MWS sessions.

The functions deletes all sessions older than the number of days specified in the parameter `OlderThanDaysCount`.

DeleteExpiredSessions

SYNTAX	DeleteExpiredSessions(OlderThanDaysCount)
ARGUMENTS	OlderThanDaysCount Number of days.
RETURN	Number of deleted sessions.
EXAMPLE	mwsAssistant = ModusProcess.GetComponent("MWSAssistantManager") deletedSessions = mwsAssistant.DeleteExpiredSessions(30)

DeleteFinishedProcesses

Deletes the processes selected by the condition.

DeleteFinishedProcesses

SYNTAX	DeleteFinishedProcesses()
RETURN	Number of deleted processes.
EXAMPLE	mwsAssistant = ModusProcess.GetComponent("MWSAssistantManager") deletedRecords = mwsAssistant.DeleteFinishedProcesses() if (deletedRecords > 0) Protocol("===== MWSAssistant deleted {0} process(es) =====", 8, deletedRecords) end-if

Sample Script MWSAssistantManager

```
mwsAssistant = ModusProcess.GetComponent("MWSAssistantManager")
deletedRecords = mwsAssistant.DeleteFinishedProcesses()
if (deletedRecords > 0)
  Protocol("===== MWSAssistant deleted {0} process(es) =====", 8, deletedRecords)
end-if
```

OdinArchive

About OdinArchive

The OdinArchive component enables you to archive a stack.

Properties of OdinArchive

Property	Description
DB Alias	Odin database on which the component executes its actions.
NextStackType	Specifies the next stack type set after successful execution. The following table displays all possible stack types. Any additionally required stack types must be defined in the file **ModusSuite.Common.PropertyEditors.dll.config**.

Default stack types

Stack type	Name	Description
0	Print	Stack is ready to be printed to an online printer or email printer (online print).
10	PortoOptimization	Stack is ready for IT franking. This function is currently not supported.
20	Streaming	Stack is ready for streaming.
30	Convert	Stack is ready to be converted to the appropriate output format.
40	StreamPrint	Stack (stream) is ready for batch printing to an outsourcing printer.
50	Archive	Stack is ready for archiving.
99	Finish	Stack processing is finished.

OnArc	Script that executes when the ArchiveJob function is called.

hive	The script only executes when the job is to be archived (ODJ_TO_ARCHIVE = "Y"). The script must check whether the job has already been archived (ODJ_ARCHIVED = "Y/N"). **Available context** Context OdinArchive
Stack Select	SQL select to identify the next stack to process. Either specify the select statement directly or refer to a lookup table (for example $OdinSQL.PrintStackSelect$).

OdinArchive Events

Event	Meaning
OnAfterStack	This event is executed when the **End()** function is called after a stack has been processed. The current stack dataset can be accessed in the context property **CurrentStack**. **Available context:** Context OdinArchive
OnArchive	This event provides access to stack, envelope or job data. In the assigned script, for example, the rendition file can be converted to an archive format (for example Tiff or PDF) by using the **OdinFileConvert** component and then written to a database.
OnBeforeStack	This event is executed when the **Start()** function is called before a stack has been processed. The current stack dataset can be accessed in the context property **CurrentStack**. **Available context:** Context OdinArchive

OdinArchive Functions

ArchiveJob

Archives the given job or the of an envelope. The job and the envelope are set in the context OdinArchive and are thus available to the OnArchive event.

ArchiveJob

SYNTAX
ArchiveJob(job)
ArchiveJob(envelope, job)

ARGUMENTS

envelope	The envelope you want to archive.
job	The job you want to archive.

EXAMPLE

```
OdinArchive.ArchiveJob(job)
OdinArchive.ArchiveJob(envelope, job)
```

End

Ends archiving.

End

SYNTAX End()

GetCurrentStackRow

Show All

After calling Start() and before calling End() the function returns an object whose interface provides read access to the current stack dataset.

GetCurrentStackRow

SYNTAX GetCurrentStackRow()

TAX

RETURN

Object interface that provides read access to the current stack dataset.

The following functions are available.

GetValue

Returns the value of the given column.

GetValue	
SYNTAX	GetValue(columnName)
ARGUMENTS	columnName Column name
RETURN	The column value. If the data type of the database column is DATE, Datetime or any valid date type, the date read generally returns in the format MM/DD/YYYY hh:mm:ss.
EXAMPLE	GetObject("DateVar","MLDate") StringDateVar = CurrentJobRecord.GetValue("ODJ_create_at") ;Convert date to the required format with the help of the MLDate object IntDateVar = DateVar.DateStringToInt64("MM/dd/yyyy hh:mm:ss", StringDateVar) v_LETTERDATE= DateVar.Int64ToDateString("MM.dd.yyyy hh:mm:ss ", IntDateVar)

GetFieldNameEnumerator

Returns an enumerator.

GetFieldNameEnumerator	
SYNTAX	GetFieldNameEnumerator()

RETURN	Enumerator with the following functions.
Current	Returns the current object.
MoveNext()	Moves the position to the next object to process.
SYNTAX	MoveNext()
RETURN	TRUE = Successful FALSE = The end of the collection is passed.
Reset	Resets the enumerator back to the beginning of the first element.

GetEnvelopeEnumerator

Returns an envelope enumerator. Use the GetValue() function to retrieve envelope data.

GetEnvelopeEnumerator

SYNTAX	GetEnvelopeEnumerator()
RETURN	Enumerator with the following functions.
Current	Returns the current envelope.
MoveNext()	Moves the position to the next envelope to process.

SYNTAX	MoveNext()
RETURN	TRUE = Successful FALSE = The end of the collection is passed.

EXAMPLE	envEnumerator = OdinPrint.GetEnvelopeEnumerator()
	nextElement = envEnumerator.MoveNext()
	while (nextElement)
	envelope = envEnumerator.Current;
	…
	nextElement = envEnumerator.MoveNext()

Reset	Resets the enumerator to the beginning of the first element.

GetJobEnumerator

Returns a job enumerator. Use the GetValue() function to retrieve job data.

GetJobEnumerator

SYNTAX

GetJobEnumerator()

GetJobEnumerator(envelope)

ARGUMENTS

envelope	The envelope whose jobs you want to access.

RETURN

Enumerator with the following functions.

Current	Returns the current job.

MoveNext()	SYNTAX	MoveNext()
	RETURN	TRUE = successful FALSE = The end of the collection is passed.

	EXAMPLE	jobEnumerator = OdinPrint.GetJobEnumerator()
		nextElement = envEnumerator.MoveNext()
		while (nextElement)
		;
		job = jobEnumerator.Current;
		;
		...
		nextElement = jobEnumerator.MoveNext()

Reset	Resets the enumerator to the beginning of the first element.	

GetNextStack

Determines the next stack to convert according to the corresponding SQL, which is stored in the StackSelect property, and returns its ID.

GetNextStack	
SYNTAX	GetNextStack()
RETURN	ID of the stack found, or an empty string if none found.

HasEnvelopes

Checks whether the stack loaded by the Start function contains envelopes.

HasEnvelopes

SYNTAX	HasEnvelopes()
RETURN	TRUE = The stack contains envelopes. FALSE = The stack contains no envelopes.

Start

Initializes archiving of a stack.

Start		
SYNTAX	Start(stackId)	
ARGUMENTS	stackId	Stack ID

Sample Script OdinArchive

```
; To convert the rendition file by the OdinFileConvert component in the OnArchive script, the component must be stored in the system pool.
; SystemPool.FileConvert = ModusProcess.GetComponent("OdinFileConvert")
; Get archive components
OdinArchive = ModusProcess.GetComponent("OdinArchive")
; determine next stack
stackID = OdinArchive.GetNextStack()
if (stackID <> "")
  ; Yes, then initialize stack archiving
  OdinArchive.Start(stackID)
  ; Does the stack contain envelopes ?
  if (OdinArchive.HasEnvelopes)
    ; Outsourcing Stack:
    ; Yes, get an envelope enumerator
    envEnumerator = OdinArchive.GetEnvelopeEnumerator()
```

```
  ; run through all envelopes and archive jobs
  nextEnv = envEnumerator.MoveNext()
  while (nextEnv)
    ; get current envelope from enumerator
    envelope = envEnumerator.Current
    ; as a test get envelope ID
    envelopId = envelope.GetValue("ODE_ID")
    ; get an job enumerator for the envelope.
    jobEnumerator = OdinArchive.GetJobEnumerator(envelope)
    ; run through all envelope jobs and archive
    nextJob = jobEnumerator.MoveNext()
    while (nextJob)
      ; get current job from enumerator
      job = jobEnumerator.Current
      ; as a test get job ID
      jobId = job.GetValue("ODJ_ID")
      ; and now archive, call the OnArchive script
      OdinArchive.ArchiveJob(envelope, job)
      ; process next Job
      nextJob = jobEnumerator.MoveNext()
    end-while
    ; process next envelope
    nextEnv = envEnumerator.MoveNext()
  end-while
else
  ; Online Stack: get an job enumerator for the stack
  jobEnumerator = OdinArchive.GetJobEnumerator()
  ; run through all stack jobs and archive
  nextJob = jobEnumerator.MoveNext()
  while (nextJob)
```

```
      ; get current job ID from enumerator
      job = jobEnumerator.Current
      ; as a test get job ID
      jobId = job.GetValue("ODJ_ID")
      ; and now archive, call the OnArchive script
      OdinArchive.ArchiveJob(job)
      ; process next Job
      nextJob = jobEnumerator.MoveNext()
    end-while
  end-if
  ; Finish archive
  OdinArchive.End()
end-if
```

OdinConvert

About OdinConvert

The OdinConvert component enables you to convert an Odin print stream to a print format like PDF, PCL, AFP, PS, or TIFF.

Note The component does not read the job datasets from the database but from the description file of the stream, as long as the description file is created during streaming.

Modifying the Output Tray List in a Compart Profile

The Odin outsourcing printer has 5 defined trays, tray 1 - 3 as well as firstbin and nextbin.

Example: Output tray list of the PCL profile mffpcl.pro

```
...
<output>
  <traylist>
    ...
    <!-- paper output bins -->
    <outputtray deviceid='15' name="Tray1" />
    <outputtray deviceid='259' name="Tray2"/>
    <outputtray deviceid='260' name="Tray3"/>
```

```
    <outputtray deviceid='15' name="firstbin"/>
    <outputtray deviceid='250' name="nextbin"/>
    ...
  </traylist>
</output>
```

To modify the output tray list in the Compart profile of the respective output format to map the **virtual trays** to the corresponding physical trays, compete the following step.

- In the [`InstallationDirectory]\profiles`, open the respective profile and modify the `deviceid`.

Properties of OdinConvert

Show All

Property	Description
ConvertOptions	Comma separated string containing the metadata and options used during conversion. You can also set these options in the OnConvertStart event to use, for example, the stack data to define the metadata. Metadata TITLE: Document title. SUBJECT: Document subject. AUTHOR: Document author. KEYWORDS: Document keywords. Options

TEXTNOTE_ CHANGES	PDF 1.3/1.4 TRUE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `True`. FALSE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `False`.
CHANGES	PDF 1.3/1.4 TRUE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `True`. FALSE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `False`.
FILL	PDF 1.4 TRUE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `True`. FALSE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `False`.
COPY	PDF 1.4 TRUE = PDF security attributes for Document Assembly, Copy Content, Copy Content for Accessibility, Page Extraction, Comment, Sign and Create Template are set to `True`. FALSE = PDF security attributes for Document Assembly, Copy Content, Copy Content for Accessibility, Page Extraction, Comment, Sign and Create Template are set to `False`.
DIGITAL_COPY	PDF 1.4 TRUE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `True`. FALSE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `False`.

ASSEMBLE	PDF 1.4 TRUE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `True`. FALSE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `False`. Assembling more than one document is not allowed.
ACCESSIBILITY	PDF 1.4 TRUE = PDF security attributes for Copy Content, Copy Content for Accessibility, Page Extraction are set to `True`. FALSE = PDF security attributes for Copy Content, Copy Content for Accessibility, Page Extraction are set to `False`.
DUPLEX_ ADDPAGE	TRUE = This option specifies whether an empty page should be added to a duplex document with an odd number of pages. FALSE = Default setting. This option sets simplex mode for the last page of a duplex document with an odd number of pages.
DUPLEX_ OPTIMIZE	TRUE = Duplex settings are only set in the output stream, when duplex mode changes. FALSE = Duplex settings are set in the output stream.
MAP_COLOR	TRUE/FALSE This parameter is used to define whether color mapping for the AFP-HighlightColor Definition is executed or not. The individual color definitions can be specified as MapColor_1 to MapColor_n. Example: Map_Color=TRUE,MapColor_1=-1,20,black,MapColor_2=0,20,red Note As MapColor parameters contain comma separated values themselves, the complete string must be enclosed in quotation marks.

	The structure is as follows: MapColor_???= HSB-Color,Threshold,ColorName HSB-Color=HSB Color Angle Threshold= Threshold value of the HSB color ColorName= Name of the color as defined in the AFP profile under HighlightColorList. The example above requires the following definitions in the AFP profile: </highlightcolorlist <highlightcolor name='red' type='OCA' number='0x0002' foreground-rgb="#00FF0000" /> <highlightcolor name='black' type='OCA' number='0x0008' foreground-rgb="#00000000" /> </highlightcolorlist> and <optimizepa value="never"/> <color value="FALSE" /> <usehighlightcolor value="TRUE" /> <enablebilevelimage value="TRUE" />
SHOW_ BOOKMARKS	TRUE = Bookmarks show in the opened PDF. FALSE = Bookmarks do not show in the opened PDF.
OVERLAYS	TRUE = OVERLAYS are created. FALSE = Default. OVERLAYS are not created.
MERGE_FONTS	TRUE: Default

	FALSE
MERGE_ SPLITWORDS	TRUE = Default, PCL output optimization is activated. FALSE = PCL output optimization is deactivated. This option should be set to FALSE if you have character spacing problems in a generated PCL. The default value is TRUE.
ROTATE_ LANDSCAPE	TRUE = If page orientation is landscape, the page rotates by 90°. FALSE = The page does not rotate. The default value is TRUE.

DB Alias
Odin database on which the component executes its actions.

Image Path
Main path to the image files, which can be added to the documents using the **Image** command.
The declaration is optional.

LicenceFilename
Path and name of the Compart .NET Toolkit license file, for example cpsdk.lic.

Master Path
Main path to documents and spool files.
The declaration is optional.

NextStackType
Specifies the next stack type set after successful execution.
The following table displays all possible stack types.
Any additionally required stack types must be defined in the file **ModusSuite.Common.PropertyEditors.dll.config**.

Default stack types

Stack type	Name	Description
0	Print	Stack is ready to be printed to an online printer or

		email printer (online print).
10	PortoOptimization	Stack is ready for IT franking. This function is currently not supported.
20	Streaming	Stack is ready for streaming.
30	Convert	Stack is ready to be converted to the appropriate output format.
40	StreamPrint	Stack (stream) is ready for batch printing to an outsourcing printer.
50	Archive	Stack is ready for archiving.
99	Finish	Stack processing is finished.

Output FileName	Returns the current name of the output file. This property is only visible on the component, not in the **Object Inspector**.
Output Format	The output format to which the stream converts. Corresponds to the file extension PCL, PS, AFP, or PDF.
Output Path	Path to output directory.
Profile Path	Directory for the Compart profile files.
StackSelect	SQL select statement to identify the next stack to be processed. You can either specify the select statement or refer to a lookup table, for example $OdinSQL.PrintStackSelect$.
TraceFileName	Name of the Compart trace file.
Trace level	Specifies which error categories are written to the trace file.

OdinConvert Events

Property	Meaning
OnConvertEnd	The specified script is executed at the end of the process when calling the **End** function.
OnConvertStart	The specified script is executed at the start of the processing when calling the **Start** function. The script can also be used to define component properties.
OnDocumentConvert	The specified script is executed for each document.
OnEnvelopeEnd	The specified script is executed at the end of the envelope.
OnEnvelopeStart	The specified script is executed at the beginning of the envelope.
OnStackEnd	The specified script is executed at the end of the stack.
OnStackStart	The specified script is executed at the beginning of the stack.

OdinConvert Functions

ConvertDocument

Converts a document and writes it to the output stream.

ConvertDocument

SYNTAX ConvertDocument(doc)

ARGUMENTS

doc	The document you want to convert.

End

Ends conversion and executes an update on the stack. A script assigned to the OnConvertEnd event is executed here.

End

SYNTAX End()

GetCurrentStackRow

Show All

After calling Start() and before calling End() the function returns an object whose interface provides read access to the current stack dataset.

GetCurrentStackRow

SYNTAX GetCurrentStackRow()

RETURN Object interface that provides read access to the current stack dataset.

The following functions are available.

GetValue

Returns the value of the given column.

GetValue

SYNTAX GetValue(columnName)

ARGUMENTS

columnName	Column name

RETURN The column value.

If the data type of the database column is DATE, Datetime or any valid date type, the date read generally returns in the format MM/DD/YYYY hh:mm:ss.

EXAMPLE

```
GetObject("DateVar","MLDate")
StringDateVar = CurrentJobRecord.GetValue("ODJ_create_at")
;Convert date to the required format with the help of the MLDate object
IntDateVar = DateVar.DateStringToInt64("MM/dd/yyyy hh:mm:ss", StringDateVar )
v_LETTERDATE= DateVar.Int64ToDateString("MM.dd.yyyy hh:mm:ss ", IntDateVar)
```

GetFieldNameEnumerator

Returns an enumerator.

GetFieldNameEnumerator

SYNTAX GetFieldNameEnumerator()

RETURN Enumerator with the following functions.

Current	Returns the current object.
MoveNext()	Moves the position to the next object to process. SYNTAX MoveNext() RETURN TRUE = Successful FALSE = The end of the collection is passed.
Reset	Resets the enumerator back to the beginning of the first element.

GetDocumentEnumerator

Returns a document enumerator.

GetDocumentEnumerator

SYNTAX GetDocumentEnumerator()

RETURN Enumerator with the following functions.

Current	Returns the current document.

MoveNext()	Moves the position to the next document to process.
	SYNTAX MoveNext()
	RETURN TRUE = Successful
	FALSE = The end of the collection is passed.
	EXAMPLE docEnumerator = OdinPrint.GetDocumentEnumerator()
	nextElement = docEnumerator.MoveNext()
	while (nextElement)
	envelope = docEnumerator.Current;
	...
	nextElement = docEnumerator.MoveNext()

Reset	Resets the enumerator to the beginning of the first element.

GetNextStack

Determines the next stack to convert according to the corresponding SQL, which is stored in the StackSelect property, and returns its ID.

GetNextStack	
SYNTAX	GetNextStack()
RETURN	ID of the stack found, or an empty string if none found.

SetStackError

Sets the status of a stack to error.

SetPdfSigner		
SYNTAX	SetStackError(errorMessage)	
ARGUMENTS	errorMessage	Error message

Start

Initializes the stack for conversion. A script assigned to the OnConvertStart event executes here.

Start		
SYNTAX	Start(stackId)	
ARGUMENTS	stackId	Stack ID

Sample Script OdinConvert

```
; Get component for converting
OdinConvert = ModusProcess.GetComponent("OdinConvert")
;
; Determine the next stack for streaming
stackId = OdinConvert.GetNextStack()
;
; Stack found ?
if (stackID <> "")
;
  ; Yes, then initialize stack / conversion.
  OdinConvert.Start(stackId)
  ;
  ;
  ; Get a document enumerator
```

```
  enumerator = OdinConvert.GetDocumentEnumerator()
  ;
  ; Are there more documents in the stream?
  nextDoc = enumerator.MoveNext()
  while (nextDoc = true)
  ;
    ; Yes, then determine the document
    doc = enumerator.Current
    ;
    ; For tests determine the number of pages and name of the document
    pageCount = doc.PageCount
    fileName = doc.FileName
    id = doc.Id
    ;
    ; and log
    Protocol("id: {0} - pages: {1} - {2}", 0, id, pageCount, fileName)
    ;
    ; and write to output stream
    OdinConvert.ConvertDocument(doc)
    ;
    ; Are there more documents in the stream?
    nextDoc = enumerator.MoveNext()
    ;
  end-while
  ; end of conversion, update stack
  OdinConvert.End()
end-if
```

OdinCustomProcess

About OdinCustomProcess

You must have an Odin license installed to use the **OdinCustomProcess** component.

The OdinCustomProcess component enables you to process stacks and works similar to the OdinArchive component.

Defining an Additional Stack Type

You can define your own stack types in the file **ModusSuite.Common.PropertyEditors.dll.config** so that these too will appear in the drop-down list for **StackType** and **NextStacktype**.

Sample ModusSuite.Common.PropertyEditors.dll.config

```
<PropertyEditor>
  <Odin>
    <StackTypes>
      <StackType value="60" name="Transfer"/>
    </StackTypes>
    <OutputFormat>
      <Format value="pdf" name="Portable Document Format" />
      <Format value="pcl" name="Printer Command Language" />
      <Format value="pos" name="Postscript" />
      <Format value="afp" name="Advanced Function Presentation" />
      <Format value="tif" name="Tagged Image File Format" />
    </OutputFormat>
  </Odin>
</PropertyEditor>
```

Properties of OdinCustomProcess

Property	Description
DB	Odin database on which the component executes its actions.

Alias

NextStackType

Specifies the next stack type set after successful execution.

The following table displays all possible stack types.

Any additionally required stack types must be defined in the file **ModusSuite.Common.PropertyEditors.dll.config**.

Default stack types

Stack type	Name	Description
0	Print	Stack is ready to be printed to an online printer or email printer (online print).
10	PortoOptimization	Stack is ready for IT franking. This function is currently not supported.
20	Streaming	Stack is ready for streaming.
30	Convert	Stack is ready to be converted to the appropriate output format.
40	StreamPrint	Stack (stream) is ready for batch printing to an outsourcing printer.
50	Archive	Stack is ready for archiving.
99	Finish	Stack processing is finished.

Stack Select

SQL select to identify the next stack to be processed.

You can either specify the select statement or refer to a lookup table, for example $OdinSQL.PrintStackSelect$.

Stack Type

Specifies the stack type that this component processes.

Select the stack type previously defined in the custom process.

OdinCustomProcess Events

Event	Meaning

OnAfterStack	The specified script is executed after the stack processing when calling the End function. The event corresponds to the **AfterStack** event of a printer in Modus 4. **Available context:** Context OdinCustomProcess
OnBeforeStack	The specified script is executed when calling the Start() function, before executing the stack processing. The event corresponds to the AfterStack event of a printer in Modus 4. **Available context:** Context OdinCustomProcess
OnProcessData	The specified script is executed when the component function ProcessJob is called. Stack, envelope or job datasets are made available. **Available context:** Context OdinCustomProcess

OdinCustomProcess Functions

End

Ends conversion and executes an update on the stack. A script assigned to the OnConvertEnd event is executed here.

End

SYNTAX End()

GetCurrentStackRow

Show All

After calling Start() and before calling End() the function returns an object whose interface provides read access to the current stack dataset.

GetCurrentStackRow

SYNTAX GetCurrentStackRow()

RETURN Object interface that provides read access to the current stack dataset.

The following functions are available.

GetValue

Returns the value of the given column.

GetValue

SYNTAX GetValue(columnName)

ARGUMENTS

columnName	Column name

RETURN The column value.

If the data type of the database column is DATE, Datetime or any valid date type, the date read generally returns in the format MM/DD/YYYY hh:mm:ss.

EXAMPLE

```
GetObject("DateVar","MLDate")
StringDateVar = CurrentJobRecord.GetValue("ODJ_create_at")
;Convert date to the required format with the help of the MLDate object
IntDateVar = DateVar.DateStringToInt64("MM/dd/yyyy hh:mm:ss", StringDateVar )
v_LETTERDATE= DateVar.Int64ToDateString("MM.dd.yyyy hh:mm:ss ", IntDateVar)
```

GetFieldNameEnumerator

Returns an enumerator.

GetFieldNameEnumerator

SYNTAX GetFieldNameEnumerator()

RETURN Enumerator with the following functions.

Current	Returns the current object.
MoveNext()	Moves the position to the next object to process. SYNTAX MoveNext() RETURN TRUE = Successful FALSE = The end of the collection is passed.
Reset	Resets the enumerator back to the beginning of the first element.

GetEnvelopeEnumerator

Returns an envelope enumerator. Use the GetValue() function to retrieve envelope data.

GetEnvelopeEnumerator

SYNTAX GetEnvelopeEnumerator()

RETURN Enumerator with the following functions.

Curre	Returns the current envelope.

nt

MoveNext()	Moves the position to the next envelope to process.

SYNTAX	MoveNext()
RETURN	TRUE = Successful FALSE = The end of the collection is passed.

EXAMPLE	envEnumerator = OdinPrint.GetEnvelopeEnumerator()
	nextElement = envEnumerator.MoveNext()
	while (nextElement)
	envelope = envEnumerator.Current;
	...
	nextElement = envEnumerator.MoveNext()

Reset	Resets the enumerator to the beginning of the first element.

GetJobEnumerator

Returns a job enumerator. Use the GetValue() function to retrieve job data.

GetJobEnumerator

SYNTAX	GetJobEnumerator() GetJobEnumerator(envelope)

ARGUMENTS	envelope	The envelope whose jobs you want to access.

RETURN	Enumerator with the following functions.

Current	Returns the current job.

MoveNext()	SYNTAX MoveNext()
	RETURN TRUE = successful
	FALSE = The end of the collection is passed.
	EXAMPLE jobEnumerator = OdinPrint.GetJobEnumerator()
	nextElement = envEnumerator.MoveNext()
	while (nextElement)
	;
	job = jobEnumerator.Current;
	;
	...
	nextElement = jobEnumerator.MoveNext()

Reset	Resets the enumerator to the beginning of the first element.

GetNextStack

Determines the next stack to convert according to the corresponding SQL, which is stored in the StackSelect property, and returns its ID.

GetNextStack	
SYNTAX	GetNextStack()
RETURN	ID of the stack found, or an empty string if none found.

HasEnvelopes

Checks whether the stack loaded by the Start function contains envelopes.

HasEnvelopes	
SYNTAX	HasEnvelopes()
RETURN	TRUE = The stack contains envelopes. FALSE = The stack contains no envelopes.

Process

Processes the specified job. The job is set in the OdinCustomProcessContext and is available to the OnProcessData event.

Process		
SYNTAX	Process(job)	
ARGUMENTS	job	Job to process.

ProcessJob

Processes the given job of the specified envelope.

The job and the envelope are set in the OdinCustomProcessContext and are thus available to the OnProcessData event.

ProcessJob		
SYNTAX	ProcessJob(envelope, job)	
ARGUMENTS	envelope	Envelope
	job	Job to process.

ProcessJobs

Processes all jobs of the specified envelope.

The job is set in the OdinCustomProcessContext and is thus available to the OnProcessData event.

ProcessJobs		
SYNTAX	ProcessJobs(envelope)	
ARGUMENTS	envelope	Envelope

Start

Initializes the stack for conversion. A script assigned to the OnConvertStart event executes.

Start		
SYNTAX	Srv_Start(stackId)	
ARGUMENTS	stackId	Stack ID

Sample Script OdinCustomProcess

```
; Get CustomProcess component
OdinCustomProcess = ModusProcess.GetComponent("OdinCustomProcess")
; determine next stack
stackID = OdinCustomProcess.GetNextStack()
;
if (stackID <> "")
;
  ; Yes, then initialize stack processing,
  ; Execute the OnBeforeStack script.
  OdinCustomProcess.Start(stackID)
  ;
  ; Does the stack have envelopes ?
  if (OdinCustomProcess.HasEnvelopes)
  ;
```

```
; Outsourcing Stack:
; Yes, get an envelope enumerator
envEnumerator = OdinCustomProcess.GetEnvelopeEnumerator()
;
; run through all envelopes and process the jobs
nextEnv = envEnumerator.MoveNext()
while (nextEnv)
  ;
  ; get current envelope from enumerator
  envelope = envEnumerator.Current
  ; as a test get envelope ID
  envelopId = envelope.GetValue("ODE_ID")
  ;
  ; get an job enumerator for the envelope.
  jobEnumerator = OdinCustomProcess.GetJobEnumerator(envelope)
  ;
  ; run through and process all envelope jobs
  nextJob = jobEnumerator.MoveNext()
  while (nextJob)
    ;
    ; get current job from enumerator
    job = jobEnumerator.Current
    ; as a test get job ID
    jobId = job.GetValue("ODJ_ID")
    ;
    ; Process job, call the OnProcessData script
    OdinCustomProcess.ProcessJob(envelope, job)
    ;
    ; process next Job
    nextJob = jobEnumerator.MoveNext()
```

```
        ;
      end-while
      ;
      ; process next envelope
      nextEnv = envEnumerator.MoveNext()
      ;
    end-while
  else
    ; Online Stack:
    ; get an job enumerator for the stack
    jobEnumerator = OdinCustomProcess.GetJobEnumerator()
    ;
    ; run through and process all stack jobs
    nextJob = jobEnumerator.MoveNext()
    while (nextJob)
      ; get current job from enumerator
      job = jobEnumerator.Current
      ; as a test get job ID
      jobId = job.GetValue("ODJ_ID")
      ;
      ; and process, the OnProcessData script is called
      OdinCustomProcess.ProcessJob(job)
      ;
      ; process next Job
      nextJob = jobEnumerator.MoveNext()
    end-while
  end-if
  ; Stack processing finished, call the OnAfterStack script
  OdinCustomProcess.End()
end-if
```

OdinDbTasks

About OdinDbTasks

The OdinDbTasks component enables you to delete and reset Odin datasets.

Properties of OdinDbTasks

Property	Description
DB Alias	Defines the Odin database in which the component executes its actions.
OnDeleteJob	Script that executes after **OnDeleteStack** for each job the component deletes.
OnDeleteStack	Script that executes before **OnDeleteJob** for each stack the component deletes.

OdinDbTasks Functions

DeleteStack

Deletes a stack including all envelopes and jobs belonging to it. The document files of the jobs are also deleted. If empty import directories are left over after the document files are deleted, these are also deleted.

DeleteStack

SYNTAX DeleteStack(CustomStackSelect)

ARGUMENTS

CustomStackSelect	Select statement to find the stacks you want to delete. If the argument is an empty string, the standard select statement is used. **Standard select for an SQL Server** Select TOP 1 ODS_ID FROM dbo.ODIN_STACK WITH (updlock, rowlock) WHERE (ODS_STATE IN (2,4)) and (ODS_DELETE_AT <= GETDATE()) and (ODS_LOCKBY is NULL) **Standard select for an Oracle database**

	Select ODS_ID FROM ODIN_STACK Where (ODS_STATE In (2,4)) and (ODS_DELETE_AT <= SYSDATE) and (ODS_LOCKBY is NULL) and (RowNum = 1) for update
RETURN	The ID of the deleted stack or an empty string if no stack was deleted.

EXAMPLE	DbTask = ModusProcess.GetComponent("OdinDbTasks")
	;no select statement defined, the default statement is used
	DeletedStackId = DbTask.DeleteStack("")
	;if a stack is deleted, its ID is returned. Otherwise an empty string is returned.
	if (StrLen(DeletedStackId) > 0)
	  Protocol("Stack with the ID {0} was deleted", 10,DeletedStackId)
	else
	  Protocol("No stack deleted", 10)
	end-if

DeleteProcessRecords

Deletes process datasets from the Odin_Process table to which no more job datasets are assigned.

DeleteProcessRecords

SYNTAX	DeleteProcessRecords(NumberOfDataSets)	
ARGUMENTS	NumberOfDataSets	Maximum number of process datasets to delete by the function call. The higher the number, the longer the execution time.
RETURN	Number of deleted processes.	
EXAMPLE	DbTask = ModusProcess.GetComponent("OdinDbTasks")	

```
ProcessCounter = DbTask.DeleteProcessRecords(100)
Protocol("'{0}' Process(es) deleted", 10, ProcessCounter)
```

SetBackStackState

Automatic reset of stacks with the status **Locked** to the status **Wait**.

Each call of this function looks for a stack dataset that meets the following conditions.

- ODS_STATE = 3 (Locked)
- ODS_RETRY_AT <= current time
- ODS_LOCKBY is NULL

If a dataset exists, it is reset.

In addition, the following database fields are updated.

- ODS_STATE = 0 (Wait)
- ODS_MESSAGE_TEXT = automatic reset

```
SetBackStackState

SYNTAX      SetBackStackState()

RETURN      The ID of the reset stack or an empty string if no stack is found or processed.

EXAMPLE     ;*************************************************************************
            ;Reset stack with status Locked to Wait
            ;*************************************************************************
            ;
            DbTask = ModusProcess.GetComponent(OdinDbTasks)
            StackId = DbTask.SetBackStackState()
            ;
            ;if a stack is reset, its ID returns. Otherwise an empty string returns.
            if (StrLen(StackId) > 0)
            Protocol("Stack with the ID {0} was reset", 10, StackId)
            else
```

```
                Protocol("No stack reset", 10)
                end-if
```

Sample Script OdinDbTasks

```
DbTask = ModusProcess.GetComponent("OdinDbTasks")
StackId = DbTask.SetBackStackState()
;
;if a stack is reset, it's ID is returned. Otherwise an empty string is returned.
if (StrLen(StackId) > 0)
  Protocol("Stack with the ID {0} was reset", 10, StackId)
else
  Protocol("No stack reset", 10)
end-if
```

OdinEmail

About OdinEmail

The OdinEmail component enables you to send emails through an SMTP server.

Properties of OdinEmail

Property	Description
BodyHtml	Sets or returns the HTML body text of the email.

BodyHmtlImages	Sets or returns the body HTML images as a comma separated string.
	The string can have the following structure.
	• id=file name
	Example
	`pic1=c:\images\pic1.jpg,picN=c:\`
	• Just the file name. In this case, the name without a file extension is used as the ID.
	Example
	`c:\images\pic1.jpg,c:\images\picN.jpg`

	• Or a mix of both **Example** `pic1=c:\images\pic1.jpg,c:\images\test.jpg,` The values always return in the form of id=file name.
BodyText	Sets or returns the body text of the email.
Encoding	Sets or returns the encoding used for the email.
EncodingCodepage	Sets or returns the encoding code page used for the email.
IsBodyHtml	Sets or returns whether the email is in HTML format. TRUE = Use BodyHtml as HTML text and BodyText as alternative plain text. FALSE = Use only the BodyText as plain text.
MailBcc	Sets or returns the Bcc recipient of the email.
MailCC	Sets or returns the CC recipient of the email. Use a comma separated list to specify more than one recipient.
MailFrom	Sets or returns the sender of the email.
MailReplyTo	Sets or returns the reply to address of the email.
MailTo	Sets or returns the recipient of the email. Use a comma separated list to specify more than one recipient.
MailSender	Specifies the originating email address used for the SMTP "MAIL FROM" command. This is also the return or bounce address in case the message is not delivered.
SmtpDomain	Specifies the domain or the computer on which the login information is checked.
SmtpHost	Host on which the SMTP server is running.
SmtpPassword	Password to authenticate to the SMTP server. The password must be encoded. **Note** To encode the password, use the tool **Encoder.exe** in the installation directory.
SmtpPickupDirectory	Sets or returns the directory to which emails are saved, if **SmtpDeliveryMethod.SpecifiedPickupDirectory** is set.

SmtpPort	Port of the SMTP server.
SmtpUserName	User name for SMTP server authentication.
Subject	Sets or returns the subject of the email.
useDefaultCredentials	TRUE = Authenticate to the SMTP server using the logged in user. FALSE = Authenticate to the SMTP server using SmtpUserName and SmtpPassword.
UseSsl	TRUE = An encoded SSL connection is used to communicate with the SMTP server. FALSE = No encoded connection is used to communicate with the SMTP server. The default value is FALSE.

OdinEmail Functions

AddAttachment

Adds an attachment to an email.

AddAttachment		
SYNTAX	AddAttachment(string fileName)	
ARGUMENTS	filename	Name of the attachment file.
EXAMPLE	email.AddAttachment("c:\temp\test.pdf")	

AddBodyHtmlImage

Adds an image to the HTML body.

AddBodyHtmlImage	
SYNTAX	AddBodyHtmlImage(string id, string imageFileName)

ARGUMENTS	id	ID of the image (cid). If no ID is specified, the file name without its extension is used.
	imageFileName	File name of the image.
EXAMPLE	email.AddBodyHtmlImage("", "c:\temp\no-fear.jpg")	

AddToHeader

Adds a name and value entry to an email header.

AddToHeader		
SYNTAX	AddToHeader(string name, string value)	
ARGUMENTS	name	Name of the entry. If the entry already exists, its value is overwritten. Otherwise, a new entry is created.
	value	Entry value.
EXAMPLE	`Email.AddToHeader("Organization", "Hyland Software")`	

ClearAttachments

Deletes all loaded attachments.

ClearAttachments	
SYNTAX	ClearAttachments()
EXAMPLE	ClearAttachments()

ClearBodyHtmlImages

Deletes all images loaded in the HTML body.

ClearBodyHtmlImages	
SYNTAX	ClearBodyHtmlImages()
EXAMPLE	ClearBodyHtmlImages()

ClearCertificates

Deletes all loaded or defined certificates.

ClearCertificates	
SYNTAX	ClearCertificates()

GetAttachments

Returns a comma separated string containing the file names of all attachments.

GetAttachments	
SYNTAX	GetAttachments()

GetDeliveryNotificationOptions

Returns the notification options for emails.

GetDeliveryNotificationOptions		
SYNTAX	GetDeliveryNotificationOptions()	
RETURN	OdinDeliveryNotificationOptions.None	No notification.
	OdinDeliveryNotificationOptions.OnSuccess	Notification on successful delivery.
	OdinDeliveryNotificationOptions.OnFailure	Notification on unsuccessful delivery

OdinDeliveryNotificationOptions.Delay	Notification on delayed delivery.
OdinDeliveryNotificationOptions.Never	No notification.

GetMailPriority

Returns the priority of an email.

GetMailPriority

SYNTAX GetMailPriority()

RETURN

OdinMailPriority.Normal	The email has normal priority.
OdinMailPriority.Low	The email has low priority.
OdinMailPriority.High	The email has high priority.

GetSmtpDeliveryMethod

Returns the option of how emails are delivered.

GetSmtpDeliveryMethod

SYNTAX GetSmtpDeliveryMethod()

RETURN

Value	Description
OdinSmtpDeliveryMethod.Network	Emails are sent through the configured SMTP server (SmptHost / SmtpPort).
OdinSmtpDeliveryMethod.SpecifiedPickupDirectory	Emails are copied to the directory

	specified in the property SmtpPickupDirectory and must be sent by an external program.
OdinSmtpDeliveryMethod.PickupDirectoryFromIis	Emails are copied to the directory configured in the Internet Information Server (IIS).

LoadBodyFromFile

Loads a body text from a file.

LoadBodyFromFile

SYNTAX LoadBodyFromFile(fileName, isHtml)

ARGUMENTS

fileName	File name,
isHtml	TRUE = Is HTML FALSE = Is not HTML

LoadCertificateFromCertFile

Loads the X.509 certificate from the specified file to send the email.

LoadCertificateFromCertFile

SYNTAX LoadCertificateFromCertFile(fileName)

ARGUMENTS

fileName	File name.

LoadCertificateFromSignedFile

Loads the X.509 certificate from the specified file to send the email.

LoadCertifacteFromSignedFile		
SYNTAX	LoadCertifacteFromSignedFile(fileName)	
ARGUMENTS	fileName	File name.

RemoveAttachment

Removes an attachment from an email.

RemoveAttachment		
SYNTAX	RemoveAttachment(fileName)	
ARGUMENTS	fileName	Name of the file to remove from the attachments.

RemoveFromHeader

Removes an entry from an email header.

RemoveFromHeader		
SYNTAX	RemoveFromHeader(name)	
ARGUMENTS	name	Name of the entry to remove.
EXAMPLE	Email.RemoveFromHeader("Organisation")	

Reset

Resets settings to defined component values.

Reset	
SYNTAX	Reset()

Send

Send the email with the current settings.

Send	
SYNTAX	Send()

SetCredentials

Sets the user credentials user name, password, and domain.

SetCredentials		
SYNTAX	SetCredentials(userName, password, domain)	
ARGUMENTS	userName	User name
	password	Encoded password **Note** To encode the password, use the tool Encoder.exe in the installation directory.
	domain	Domain
RETURN	TRUE = Attribute value is set. FALSE = Attribute value could not be set.	

SetDeliveryNotificationOptions

Defines delivery notification for emails.

SetDeliveryNotificationOptions		
SYNTAX	SetDeliveryNotificationOptions(deliveryNotificationOptions)	
ARGUMENTS	deliveryNotificationOptions	**Possible values**

	OdinDeliveryNotificationOptions.None = No notification OdinDeliveryNotificationOptions.OnSuccess = Notification on successful delivery OdinDeliveryNotificationOptions.OnFailure = Notification on unsuccessful delivery OdinDeliveryNotificationOptions.Delay = Notification on delayed delivery OdinDeliveryNotificationOptions.Never = No notification whatsoever

SetMailPriority

Sets the priority of an email.

SetMailPriority		
SYNTAX	SetMailPriority(mailPriority)	
ARGUMENTS	mailPriority	**Possible values** OdinMailPriority.Normal = The email has normal priority. OdinMailPriority.Low = The email has low priority. OdinMailPriority.High = The email has high priority.

SetSmtpDeliveryMethod

Defines how emails are delivered.

SetSmtpDeliveryMethod	
SYNTAX	SetSmtpDeliveryMethod(smtpDeliveryMethod)

ARGUMENTS	smtpDeliveryMethod	**Possible values** OdinSmtpDeliveryMethod.Network = Emails are sent through the configured SMTP Server (SmptHost / SmtpPort). OdinSmtpDeliveryMethod.SpecifiedPickupDirectory = Emails are copied to the directory specified in the property SmtpPickupDirectory and must be sent by an external program. OdinSmtpDeliveryMethod.PickupDirectoryFromIis = Emails are copied to the directory configured in the Internet Information Server (IIS).

SetSmtpServer

Sets the SMTP server properties host, port, and useDefaultCredentials.

SetSmtpServer		
SYNTAX	SetSmtpServer(host, port, useDefaultCredentials)	
ARGUMENTS	host	Host on which the SMTP server runs
	port	Port of the SMTP server
	useDefaultCredentials	TRUE = Authenticate to SMTP server using the logged on user. FALSE = Authenticate to SMTP server using SmptUserName and SmtpPassword.

Sample Script OdinEmail

```
; Get Email component
email = ModusProcess.GetComponent("OdinEmail")
;
; Test: Encoding by display name / name / code page
email.Encoding = "Western European" (Windows)
```

```
;email.Encoding = "windows-1252"
;email.EncodingCodepage = 1252
;
; SMTP-Server + User
email.SmtpHost = "192.168.24.166"
email.SmtpPort = 25
email.UseDefaultCredentials = false
email.SmtpUserName = "test"
email.SmtpPassword = "test"
email.SmtpDomain = ""
;
; Can alternatively be set by the function SetSmtpServer/SetCredentials
;email.SetSmtpServer("192.168.24.166", 25, false)
;email.SetCredentials("test", "test", "")
;
; SetSmtpDeliveryMethod (OdinSmtpDeliveryMethod ->
Network/SpecifiedPickupDirectory/PickupDirectoryFromIis)
;email.SetSmtpDeliveryMethod(OdinSmtpDeliveryMethod.Network)
;email.SmtpPickupDirectory = "c:\temp\"
;
; Email From/To/CC/Bcc/ReplyTo
; The comma separated string for email addresses can have the following structure.
; a) "DisplayName" <email@domain>,<email@domain>,email@domain
; or: "displayName-1" <email-1@domain-1.de>,"displayName-N" <email-N@domain-N.de>,<email-
X@domain-X.de>,email-Y@domain-Y.de
email.MailFrom = """Jo"" <jo@mydomain.de>"
email.MailTo = """Test"" <test@mydomain.de>,jo@mydomain.de"
email.MailReplyTo = """Jo"" <jo@mydomain.de>"
;
; SetMailPriority (OdinMailPriority -> Normal/High/Low)
```

```
email.SetMailPriority(OdinMailPriority.High)
;
; SetDeliveryNotification (OdinDeliveryNotificationOptions -> None/OnSuccess/OnFailure/Delay/Never)
email.SetDeliveryNotificationOptions(OdinDeliveryNotificationOptions.None)
;
; Subject + Body
email.Subject = "test-mail..."
email.BodyText = "plain-text body"
email.BodyHtml = "<!DOCTYPE HTML PUBLIC ""-//W3C//DTD HTML 4.0 Transitional//EN""><HTML><HEAD><META http-equiv=Content-Type content=""text/html; charset=iso-8859-1""><HEAD><BODY><DIV><FONT face=Arial color=#ff0000 size=5>test: this is a html mail from Jo</FONT></DIV><img alt="""" hspace=0 src=""cid:No-Fear"" align=baseline border=0></BODY></HTML>"
email.IsBodyHtml = true
;
; Alternatively, bodies can be loaded using LoadBodyFromFile()
;email.LoadBodyFromFile("plain.txt", false) ; = BodyText
;email.LoadBodyFromFile("html.txt", true) ; = BodyHtml
;
; image is inserted using cid: No-Fear, that means insert image
; If no ID is specified, the file name without its extension is used as the ID.
email.AddBodyHtmlImage("", "c:\temp\no-fear.jpg")
;
; Insert PDF as attachment
email.AddAttachment("c:\temp\test.pdf")
;
; or possibly load X509 certificate (SSL is then used)
;email.LoadCertificateFromCertFile("certFile.cer")
;email.LoadCertifacteFromSignedFile("signedFile.cer")
;
; and send email...
```

email.Send()

OdinFileConvert

About OdinFileConvert

The OdinFileConvert component enables you to convert and collate 1-n documents in the output format. The available output formats, for example, PDF or TIFF, depend on the license.

You can use this component in archiving processes to convert an XPS rendition in the output format before the rendition is archived.

Note You can still modify the documents during batch conversion.

Properties of OdinFileConvert

Property	Meaning
LicenceFilename	Path and name of the Compart .NET Toolkit license file.
NumberOfFilesConverted	Returns the number of files already converted and appended to the output file during batch conversion.
OutputFormat	Returns the output format of the stream. The drop-down list shows all the possible output formats as defined in the ModusSuite.Common.PropertyEditors.dll.config file. The formats depend on the corresponding Compart .NET-Toolkit license. **Sample ModusSuite.Common.PropertyEditors.dll.config** <PropertyEditor> <Odin> <StackTypes> <StackType value="60" name="Transfer"/> </StackTypes> <OutputFormat> <Format value="pdf" name="Portable Document Format" /> <Format value="pcl" name="Printer Command Language" /> <Format value="pos" name="Postscript" /> <Format value="afp" name="Advanced Function Presentation" />

	<Format value="tif" name="Tagged Image File Format" /> </OutputFormat> </Odin> </PropertyEditor>
ProfilePath	Path to the Compart profiles set in the respective input and output filters.
TraceFileName	Path and name of the trace file for the Compart .NET-Toolkit.

OdinFileConvert Functions

AddDocument

Converts the specified Odin document and adds it to the output file.

AddDocument		
SYNTAX	AddDocument(document)	
ARGUMENTS	document	Document you want to add.
EXAMPLE	OdinFileConverter.AddDocument(document)	

AddFile

Converts the specified file and adds it to the output file during batch conversion.

AddFile		
SYNTAX	AddFile(fileName)	
ARGUMENTS	filename	The file you want to add.
EXAMPLE	OdinFileConverter.AddFile("c:\temp\Document.xps")	

CreatePdfSigner

Show All

The functions create a PDF signer and return its IOdinPdfSigner interface.

The certificate used for signing can either come from an exported certificate file (PFX) or from a certificate store.

The PDF signer object can be used to assign a digital signature to a PDF file.

This can be used after an output file has been created from conversion in an OnConvertEnd event, or can be used by an email printer in the event OnConvert (see Context OdinEmailConvert).

Syntax

CreatePdfSigner(string pfxFileName, string password)

CreatePdfSigner(OdinStoreName storeName, OdinStoreLocation storeLocation, OdinX509FindType findType, object findValue, bool validOnly)

CreatePdfSigner

SYNTAX

CreatePdfSigner(pfxFileName, password)

CreatePdfSigner(storeName, storeLocation, findType, findValue, validOnly)

ARGUMENTS

The password required for access to X.509 certificate data.

The password must be encrypted.

To encode the password, use the tool **Encoder.exe** in the Bin-Directory.

storeName

One of the OdinStoreName values:

Value	Definition
OdinStoreName.AddressBook	The X.509 certificate store for other users.
OdinStoreName.AuthRoot	The X.509 certificate store for third-party certificate authorities (CAs).
OdinStoreName.CertificateAuthority	The X.509 certificate store for intermediate certificate authorities (CAs).
OdinStoreName.Disallowed	The X.509 certificate store for revoked certificates.
OdinStoreName.My	The X.509 certificate store for personal

	certificates.
OdinStoreName.Root	The X.509 certificate store for trusted root certificate authorities (CAs).
OdinStoreName.TrustedPeople	The X.509 certificate store for directly trusted people and resources.
OdinStoreName.TrustedPublisher	The X.509 certificate store for directly trusted publishers.

storeLocation

One of the OdinX509FindType values:

Value	Name of the certificate file.
password	Encoded password required for access to X.509 certificate data. **Note** To encode the password, use the tool Encoder.exe in the installation directory.
storeName	One of the OdinStoreName values.

Possible values	Description
OdinStoreName.AddressBook	The X.509 certificate store for other users.
OdinStoreName.AuthRoot	The X.509 certificate store for third-party certificate authorities (CAs).
OdinStoreName.CertificateAuthority	The X.509 certificate store for intermediate certificate authorities (CAs).

	OdinStoreName.Disallowed — The X.509 certificate store for revoked certificates. OdinStoreName.My — The X.509 certificate store for personal certificates. OdinStoreName.Root — The X.509 certificate store for trusted root certificate authorities (CAs). OdinStoreName.TrustedPeople — The X.509 certificate store for directly trusted people and resources. OdinStoreName.TrustedPublisher — The X.509 certificate store for directly trusted publishers.
storeLocation	One of the OdinX509StoreLocation values. Possible values: OdinStoreLocation.CurrentUser — The X.509 certificate store used by the current user. OdinStoreLocation.LocalMachine — The X.509 certificate store for third-party certificate authorities (CAs).
findType	One of the OdinX509FindType values.

Possible values	Description
OdinX509FindType.FindByThumbprint	The findValue parameter must be a string representing the thumbprint of the certificate.
OdinX509FindType.FindBySubjectName	The findValue parameter must be a string representing the subject name of the certificate.
OdinX509FindType.FindBySubjectDistinguishedName	The findValue parameter must be a string representing the subject distinguished name of the certificate.
OdinX509FindType.FindByIssuerName	The findValue parameter must be a string representing the issuer name of the certificate.
OdinX509FindType.FindByIssuerDistinguishedName	The findValue parameter must be a string representing the issuer distinguished name of the certificate.
OdinX509FindType.FindBySerialNumber	The findValue parameter must be a string representing the serial number of the certificate as it is displayed by the UI. The serial number must be in reverse order because it is an integer.
OdinX509FindType.FindByTimeValid	The findValue parameter must be a DateTime value in local time, for example Now.
OdinX509FindType.FindByTimeNotYetValid	The findValue parameter must be a DateTime value in local time, for example Now.
OdinX509FindType.FindByTi	The findValue parameter must

meExpired	be a DateTime value in local time, for example Now.
OdinX509FindType.FindByTemplateName	The findValue parameter must be a string representing the template name of the certificate, for example "ClientAuth. A template name is an X509 version 3 extension that specifies the uses of the certificate.
OdinX509FindType.FindByApplicationPolicy	The findValue parameter must be a string representing either the application policy friendly name or the object identifier (OID) of the certificate. For example, "Encrypting File System" or "1.3.6.1.4.1.311.10.3.4" can be used. For an application that is localized, the OID value must be used, because the friendly name is localized.
OdinX509FindType.FindByCertificatePolicy	The findValue parameter must be a string representing either the friendly name or the object identifier (OID) of the certificate policy. The best practice is to use the OID, such as "1.3.6.1.4.1.311.10.3.4". For an application that will be localized, the OID value must be used, because the friendly name is localized.
OdinX509FindType.FindByExtension	The findValue parameter must

	be a string describing the extension to find. The object identifier (OID) is most commonly used to direct the Find function to search for all certificates that have an extension matching that OID value.
OdinX509FindType.FindByKeyUsage	The findValue parameter must be either a string representing the key usage or an integer representing a bit mask containing all the requested key usages. Only one key usage at a time can be specified for the string value.
OdinX509FindType.FindBySubjectKeyIdentifier	The findValue parameter must be a string representing the subject key identifier in hexadecimal, such as "F3E815D45E83B8477B9284113C64EF208E897112", as displayed in the UI.

findValue	Search criteria as string or date time.
validOnly	The findValue parameter must be a string representing the issuer distinguished name of the certificate.
OdinX509FindType.FindBySerialNumber	The findValue parameter must be a string representing the serial number of the certificate as it is displayed by the UI. The serial number must be in reverse order because it is an integer.
OdinX509FindType.FindByTimeValid	The findValue parameter must be a DateTime value in local time, for example Now.
OdinX509FindType.FindByTimeNotYetValid	The findValue parameter must be a DateTime value in local time, for example Now.

OdinX509FindType.FindByTimeExpired	The findValue parameter must be a DateTime value in local time, for example Now.
OdinX509FindType.FindByTemplateName	The findValue parameter must be a string representing the template name of the certificate, for example "ClientAuth". A template name is an X509 version 3 extension that specifies the uses of the certificate.
OdinX509FindType.FindByApplicationPolicy	The findValue parameter must be a string representing either the application policy friendly name or the object identifier (Oid) of the certificate. For example, "Encrypting File System" or "1.3.6.1.4.1.311.10.3.4" can be used. For an application that will be localized, the OID value must be used, because the friendly name is localized.
OdinX509FindType.FindByCertificatePolicy	The findValue parameter must be a string representing either the friendly name or the object identifier (Oid) of the certificate policy. The best practice is to use the OID, such as "1.3.6.1.4.1.311.10.3.4". For an application that will be localized, the OID value must be used, because the friendly name is localized.
OdinX509FindType.FindByExtension	The findValue parameter must be a string describing the extension to find. The object identifier (OID) is most commonly used to direct the Find function to search for all certificates that have an extension matching that OID value.
OdinX509FindType.FindByKeyUsage	The findValue parameter must be either a string representing the key usage or an integer representing a bit mask containing all the requested key usages. Only one key usage at a time can be specified for the string value.
OdinX509FindType.FindBySubjectKeyIdentifier	The findValue parameter must be a string representing the subject key identifier in hexadecimal, such as "F3E815D45E83B8477B9284113C64EF208E897112", as displayed in the UI.

RETURN	Search criteria as string or date time.
validOnly	True, so only valid certificates are returned by the search.

Return value

Interface IOdinPdfSigner

functions of the IOdinPdfSigner Interface

AddSignature

Inserts a digital signature into a PDF file using the certificate for signing defined when the PDF signer was created.

Notes

- If no certificate exists, a dialog box is displayed to select the certificate (My Certificates / Current Users).
- If the PDF is password protected, the password must be defined in the property **PdfPassword**.

Syntax

AddSignature(string inputFileName, string outputFileName)

AddSignature(string inputFileName, string outputFileName, string pdfPassword)

Parameter

inputFileName : Name of the PDF file to be signed.

outputFileName: Name of the signed PDF to be created.

pdfPassword: PDF password. The password must be encoded. To encode the password, use the tool **Encoder.exe** in the Bin-Directory.

Return value

-

Initialize

Initializes the PDF signer.

Syntax

Initialize(OdinSignatureStyle signatureStyle, int pageNumber, bool visible, string fieldName, string signatureImage)

Parameter

Parameter	Meaning
signatureStyle	Defines how the signature is to be created. **Possible values:**

	OdinSignatureStyle.Text: Shows only the specified text. OdinSignatureStyle.Description: Shows only the description (date/reason/place) and not the specified text. OdinSignatureStyle.ImageAndText: Shows text and signature image. OdinSignatureStyle.ImageAndDescription: Shows the description and signature image.
pageNumber	Defines on which page the signature is to be inserted.
visible	Specifies whether the signature is visible.
fieldName	Specifies the name of the signature field to be inserted.
signatureImage	Specifies the image to be shown alongside the text.

Return value

-

RegisterFont

Registers a font file.

Syntax

RegisterFont(string fileName)

RegisterFont(string fileName, string alias)

Parameter

fileName: Name of the TrueType or Type1/3 font file.

alias: Alias name the font is registered as.

Return value

-

RegisterFontDirectory

Registers all fonts in a specified directory.

All fonts in

c:/windows/fonts

and

c:/winnt/fonts

are registered by default.

Syntax

RegisterFontDirectory(string directoryName)

Parameter

directoryName: Name of the directory that contains the fonts to be registered.

Return value

-

SetFont

Defines the font specification for text output.

Syntax

SetFont(string fontName, int fontSize, int fontStyle, string fontColor)

Parameter

fontName: Name of the font.

fontSize: Size of the font.

fontStyle: Style of the font. See OdinFontStyle.

fontColor: Color of the font, for example: "AARRGGBB" or a prescribed name such as "Grey".

Return value

-

SetPosition

Defines the position, size in points and, if required, the background image of the signature.

Conversion of mm -> points: mm / 25,4 => inch * 72 = points

Syntax

SetPosition(double xpos, double ypos, double width, double height, string imageFileName)

Parameter

Xpos: X-position of the signature.

Ypos: Y-position of the signature.

Width: Width of the signature.

Height: Height of the signature.

imageFileName: Name of the background image.

Return value

-

SetText

Sets the signature or description text and the contact information of the signer.

Syntax

SetText(string text, string reason, string location, string contact)

Parameter

text: Defines the signature text (in the case of OdinSignatureStyle.Text or OdinSignatureStyle.ImageAndText)

reason: Defines the reason (in the case of OdinSignatureStyle.Description or OdinSignatureStyle.ImageAndDescription)

location: Defines the location (in the case of OdinSignatureStyle.Description or OdinSignatureStyle.ImageAndDescription)

contact: Defines the contact information of the signer.

Return value

-

SetTimeStampServer

Sets the URL and the user / password properties of the time stamp server to be used.

Syntax

SetTimeStampServer(string tsaServerUrl, string tsaUser, string tsaPassword)

Parameter

Parameter	Meaning
tsaServerUrl	URL of the time stamp server
tsaUser	User name of the time stamp server
tsaPassword	Password of user. The password must be encoded. To encode the password, use the tool **Encoder.exe** in the Bin-Directory.

Return value

-

Properties of the IOdinPdfSigner Interface

Property	Definition
saServer :string	URL of the time stamp server, for example "http://dse200.ncipher.com/TSS/HttpTspServer" If this is empty, the current time of the computer is used and not detected/encrypted through the TSA server.
TsaUserName :string	User name of the TSA server.
TsaPassword :string	Password of the TSA server.
FieldName :string	Name of the signature field in the PDF, by default "Signature".
ImageFileName :string	Specifies the name of the image file to be used for the Signature.
Text :string	Text of the signature. (is used by OdinSignatureStyle.Text or OdinSignatureStyle.ImageAndText).
Reason :string	Reason for the signature (is used by OdinSignatureStyle.Description or OdinSignatureStyle.ImageAndDescription).
Location :string	Location of the signature (is used by OdinSignatureStyle.Description or OdinSignatureStyle.ImageAndDescription).
Contact :string	Contact information of the signer.
PageNumber :int	Page the signature will be inserted on.
Visible :bool	Specifies whether the signature is visible.
XPos	X-position of the signature in points.

:double

YPos :double — Y-position of the signature in points.

Width :double — Width of the signature in points.

Height :double — Height of the signature in points.

FontName :string — Name of the font used for the signature.

FontColor :string — Font color. eg.: "AARRGGBB" or "Gray"

FontSize :int — Size of the font.

FontStyle :int — Style of the font (a combination of: 0=Normal, 1=Bold, 2=Italic, 4=Underline, 8=Strikethrough)

Property	Description
saServer	URL of the time stamp server, for example "http://dse200.ncipher.com/TSS/HttpTspServer". If this property is empty, the current time of the computer is used and not detected or encrypted through the TSA server.
TsaUserName	User name of the TSA server.
TsaPassword	Password of the TSA server.
FieldName	Name of the signature field in the PDF. The default value is "Signature".
ImageFileName	Specifies the name of the image file to be used for the signature.

Text	Text of the signature (used by OdinSignatureStyle.Text or OdinSignatureStyle.ImageAndText).
Reason	Reason for the signature (used by OdinSignatureStyle.Description or OdinSignatureStyle.ImageAndDescription).
Location	Location of the signature (used by OdinSignatureStyle.Description or OdinSignatureStyle.ImageAndDescription).
Contact	Contact information of the signer.
PageNumber	Page the signature is inserted on.
Visible	Specifies whether the signature is visible.
XPos	X-position of the signature in points.
YPos	Y-position of the signature in points.
Width	Width of the signature in points.
Height	Height of the signature in points.
FontName	Name of the font used for the signature.
FontColor	Font color, for example "AARRGGBB" or "Gray".
FontSize	Size of the font.
FontStyle	Style of the font.

Possible values	Description
OdinPdfFontStyle.Normal	0
OdinPdfFontStyle.Bold	1

OdinPdfFontStyle.Italic	2
OdinPdfFontStyle.Underline	4
OdinPdfFontStyle.Strikethrough	8

You can define the integer values in combination, for example: OdinPdfFontStyle.Bold + OdinPdfFontStyle.Italic.

SignatureImage	Signature image (is shown by OdinSignatureStyle.ImageAndText or OdinSignatureStyle.ImageAndDescription).
PdfPassword	PDF owner password.

EXAMPLE

```
; get OdinFileConvert component
OdinFileConvert = ModusProcess.GetComponent(OdinFileConvert)
;
; define variables
; Time stamp server
tsaServer = "http://dse200.ncipher.com/TSS/HttpTspServer"
tsaUser = ""
tsaPassword = ""
;
; Pdf-/Image-Filename
inputPdf = "C:\Hyland\data\Test.pdf"
outputPdf = "C:\Hyland\data\Test_signed.pdf"
pdfPassword = ""
imageFileName = "C:\Hyland\data\signature.JPG"
bgndImageFileName = ""
```

```
;
; Signature-Text
fieldName = "Signature"
reason = "Reason"
location = "Place"
contact = "Contact"
signatureText = "Signature"
signatureStyle = OdinSignatureStyle.Text
;
; Signature position
visible = true
pageNo = 1
xpos = 100
ypos = 100
width = 150
height = 50
;
; Font
fontName = "Verdana"
fontSize = 8
fontColor = "00808080"
fontStyle = OdinPdfFontStyle.Bold + OdinPdfFontStyle.Italic
;
; create Pdf-Signer with certificate from a PFX file
;pdfSigner = OdinFileConvert.CreatePdfSigner("C:\Hyland\Certificates\PS.pfx", "HylandSoftware")
; create Pdf-Signer with a certificate from the store (My Certificates / Current Users).
pdfSigner = OdinFileConvert.CreatePdfSigner(OdinStoreName.My, OdinStoreLocation.CurrentUser,
OdinX509FindType.FindBySubjectName, "HylandSoftware", true)
;
; initialize signer
```

```
pdfSigner.Initialize(signatureStyle, pageNo, visible, fieldName, imageFileName)
;
; initialize position
pdfSigner.SetPosition(xpos, ypos, width, height, bgndImageFileName)
;
; initialize text/description
;pdfSigner.SetText("", reason, location, contact)
pdfSigner.SetText(signatureText, reason, location, contact)
;
; initialize font
pdfSigner.SetFont(fontName, fontSize, fontStyle, fontColor)
;
; initialize time stamp server
pdfSigner.SetTimeStampServer(tsaServer, tsaUser, tsaPassword)
;
; insert signature to PDF
pdfSigner.AddSignature(inputPdf, outputPdf, pdfPassword)
```

ConvertSingleFile

Show All

Converts a single file to the specified format.

Available formats are dependent on the Compart license for input and output filters.

ConvertSingleFile	
SYNTAX	ConvertSingleFile(inputFileName, outputFileName) ConvertSingleFile(inputFileName, outputFileName, singlePage) ConvertSingleFile(inputFileName, outputFileName, singlePage, convertOptions)

ARGUMENTS

inputFileName	The file to convert.
outputFileName	The file to create.
singlePage	TRUE = Write one output file per page according to the following naming convention FILENAME_#PAGENUMBER#.outputformat **Example** OutputFilename_001.pdf The default value is FALSE.
convertOptions	Comma separated string containing the metadata and options used during conversion. Their use is dependent on the output format. The parameters are optional.

Metadata

TITLE	Document title.
SUBJECT	Document subject.
AUTHOR	Document author.
KEYWORDS	Document keywords.

Options

TEXTNOTE_CHANGES	PDF 1.3/1.4 TRUE = PDF security attributes

	for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `True`. FALSE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `False`.
CHANGES	PDF 1.3/1.4 TRUE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `True`. FALSE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `False`.
FILL	PDF 1.4 TRUE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `True`. FALSE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `False`.
COPY	PDF 1.4 TRUE = PDF security attributes for Document Assembly, Copy Content, Copy Content for Accessibility, Page Extraction,

	Comment, Sign and Create Template are set to `True`. FALSE = PDF security attributes for Document Assembly, Copy Content, Copy Content for Accessibility, Page Extraction, Comment, Sign and Create Template are set to `False`.
DIGITAL_COPY	PDF 1.4 TRUE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `True`. FALSE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `False`.
ASSEMBLE	PDF 1.4 TRUE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `True`. FALSE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `False`. Assembling more than one document is not allowed.
ACCESSIBILITY	PDF 1.4 TRUE = PDF security attributes for Copy Content, Copy Content for Accessibility, Page Extraction

	are set to `True`. FALSE = PDF security attributes for Copy Content, Copy Content for Accessibility, Page Extraction are set to `False`.
DUPLEX_ ADDPAGE	TRUE = This option specifies whether an empty page should be added to a duplex document with an odd number of pages. FALSE = Default setting. This option sets simplex mode for the last page of a duplex document with an odd number of pages.
DUPLEX_ OPTIMIZE	TRUE = Duplex settings are only set in the output stream, when duplex mode changes. FALSE = Duplex settings are set in the output stream.
MAP_COLOR	TRUE/FALSE This parameter is used to define whether color mapping for the AFP-HighlightColor Definition is executed or not. The individual color definitions can be specified as MapColor_1 to MapColor_n. Example: Map_Color=TRUE,MapColor_ 1=-1,20,black,MapColor_ 2=0,20,red

	Note As MapColor parameters contain comma separated values themselves, the complete string must be enclosed in quotation marks. The structure is as follows: MapColor_???= HSB-Color,Threshold,ColorName HSB-Color=HSB Color Angle Threshold= Threshold value of the HSB color ColorName= Name of the color as defined in the AFP profile under HighlightColorList. The example above requires the following definitions in the AFP profile: </highlightcolorlist <highlightcolor name='red' type='OCA' number='0x0002' foreground-rgb="#00FF0000" /> <highlightcolor name='black' type='OCA' number='0x0008' foreground-rgb="#00000000" /> </highlightcolorlist> and <optimizepa value="never"/>

	<color value="FALSE" /> <usehighlightcolor value="TRUE" /> <enablebilevelimage value="TRUE" />
SHOW_ BOOKMARKS	TRUE = Bookmarks show in the opened PDF. FALSE = Bookmarks do not show in the opened PDF.
OVERLAYS	TRUE = OVERLAYS are created. FALSE = Default. OVERLAYS are not created.
MERGE_FONTS	TRUE: Default FALSE
MERGE_ SPLITWORDS	TRUE = Default, PCL output optimization is activated. FALSE = PCL output optimization is deactivated. This option should be set to FALSE if you have character spacing problems in a generated PCL. The default value is TRUE.
ROTATE_ LANDSCAPE	TRUE = If page orientation is landscape, the page rotates by 90°. FALSE = The page does not rotate. The default value is TRUE.

ConvertXDocToHtml

Converts an XDOC file to HTML.

ConvertXDocToHtml

SYNTAX ConvertXDocToHtml(xDocFileName, htmlOutputFileName, saveImages)

ARGUMENTS

xDocFileName	The XDOC file to convert.
htmlOutputFileName	The HTML file to create.
saveImages	TRUE = Save images to the same directory.

ConvertXDocToString

Converts an XDOC file to a string using the line break character specified for the environment or set as a parameter.

ConvertXDocToString

SYNTAX
ConvertXDocToString(xDocFileName)
ConvertXDocToString(xDocFileName, newLine)

ARGUMENTS

xDocFileName	The XDOC file you want to convert.
newLine	Defines the type of line break.

Possible values	Result
NewLineLF	LineFeed ("\n" - $0A)
NewLineCRLF	CarriageReturn/LineFeed ("\r\n" - $0D $0A)

RETURN The XDOC file as string.

ConvertXDocToText

Converts an XDOC file to a text file.

ConvertXDocToText

SYNTAX

ConvertXDocToText(xDocFileName,

ConvertXDocToText(xDocFileName, txtOutputFileName, encoding)

ConvertXDocToText(xDocFileName, txtOutputFileName, encoding, newLine)

ARGUMENTS

xDocFileName	The XDOC file you want to convert.
txtOutputFileName	The text file you want to create.
encoding	You can use the following values as encoding parameters. • Empty string. In this case, the default encoding (ANSI code page of the operating system) is used. • Code page, for example, "1252" • Name, for example, "windows-1252" • Friendly name, for example, "Western European" (Windows) Possible encoding parameters

Code page	name	Display name
37	IBM037	IBM EBCDIC (USA-Canada)
437	IBM437	OEM USA
500	IBM500	IBM EBCDIC (International)
708	ASMO-708	Arabic (ASMO 708)
720	DOS-720	Arabic (DOS)
737	ibm737	Greek (DOS)
775	ibm775	Baltic (DOS)
850	ibm850	Western European (DOS)
852	ibm852	Eastern European (DOS)
855	IBM855	OEM Cyrillic

857	ibm857	Turkish (DOS)
858	IBM00858	OEM Multilingual Latin I
860	IBM860	Portuguese (DOS)
861	ibm861	Icelandic (DOS)
862	DOS-862	Hebrew (DOS)
863	IBM863	French, Canada (DOS)
864	IBM864	Arabic (864)
865	IBM865	Nordic (DOS)
866	cp866	Cyrillic (DOS)
869	ibm869	Greek, modern (DOS)
870	IBM870	IBM EBCDIC (Multilingual Latin-2)
874	windows-874	Thai (Windows)
875	cp875	IBM EBCDIC (Greek, modern)
932	shift_jis	Japanese (Shift-JIS)
936	gb2312	Simplified Chinese (GB2312)
949	ks_c_5601-1987	Korean
950	big5	Chinese (Traditional) (Big5)
1026	IBM1026	IBM EBCDIC (Turkish, Latin-5)
1047	IBM01047	IBM Latin-1
1140	IBM01140	IBM EBCDIC (USA-Canada-European)
1141	IBM01141	IBM EBCDIC (Germany-European)
1142	IBM01142	IBM EBCDIC (Denmark-Norway-European)
1143	IBM01143	IBM EBCDIC (Finland-Sweden-European)
1144	IBM01144	IBM EBCDIC (Italian-European)
1145	IBM01145	IBM EBCDIC (Spain-European)
1146	IBM01146	IBM EBCDIC (Great Britain-European)

1147	IBM01147	IBM EBCDIC (France-European)
1148	IBM01148	IBM EBCDIC (International-European)
1149	IBM01149	IBM EBCDIC (Icelandic-European)
1200	utf-16	Unicode
1201	unicodeFFFE	Unicode (Big Endian)
1250	windows-1250	Central European (Windows)
1251	windows-1251	Cyrillic (Windows)
1252	Windows-1252	Western European (Windows)
1253	windows-1253	Greek (Windows)
1254	windows-1254	Turkish (Windows)
1255	windows-1255	Hebrew (Windows)
1256	windows-1256	Arabic (Windows)
1257	windows-1257	Baltic (Windows)
1258	windows-1258	Vietnamese (Windows)
1361	Johab	Korean (Johab)
10000	macintosh	Western European (Mac)
10001	x-mac-japanese	Japanese (Mac)
10002	x-mac-chinesetrad	Chinese traditional (Mac)
10003	x-mac-korean	Korean (Mac)
10004	x mac-arabic	Arabic (Mac)
10005	x-mac-hebrew	Hebrew (Mac)
10006	x-mac-greek	Greek (Mac)
10007	x-mac-cyrillic	Cyrillic (Mac)
10008	x-mac-chinesesimp	Chinese simplified (Mac)
10010	x-mac-romanian	Romanian (Mac)
10017	x-mac-ukrainian	Ukrainian (Mac)
10021	x-mac-thai	Thai (Mac)
10029	x-mac-ce	Central European (Mac)

10079	x-mac-icelandic	Icelandic (Mac)
10081	x-mac-turkish	Turkish (Mac)
10082	x-mac-croatian	Croatian (Mac)
12000	utf-32	Unicode (UTF-32)
12001	utf-32BE	Unicode (UTF-32-Big Endian)
20000	x-Chinese-CNS	Chinese traditional (CNS)
20001	x-cp20001	TCA Taiwan
20002	x-Chinese-Eten	Chinese traditional (Eten)
20003	x-cp20003	IBM5550 Taiwan
20004	x-cp20004	TeleText Taiwan
20005	x-cp20005	Wang Taiwan
20105	x-IA5	Western European (IA5)
20106	x-IA5-German	German (IA5)
20107	x-IA5-Swedish	Swedish (IA5)
20108	x-IA5-Norwegian	Norwegian (IA5)
20127	us-ascii	US-ASCII
20261	x-cp20261	T.61
20269	x-cp20269	ISO-6937
20273	IBM273	IBM EBCDIC (Germany)
20277	IBM277	IBM EBCDIC (Denmark-Norway)
20278	IBM278	IBM EBCDIC (Finland-Sweden)
20280	IBM280	IBM EBCDIC (Italy)
20284	IBM284	IBM EBCDIC (Spain)
20285	IBM285	IBM EBCDIC (Great Britain)
20290	IBM290	IBM EBCDIC (Japanese Katakana)
20297	IBM297	IBM EBCDIC (France)
20420	IBM420	IBM EBCDIC (Arabic)
20423	IBM423	IBM EBCDIC (Greek)
20424	IBM424	IBM EBCDIC (Hebrew)
20833	x-EBCDIC-KoreanExtended	IBM EBCDIC (Korean, extended)

20838	IBM-Thai	IBM EBCDIC (Thai)
20866	koi8-r	Cyrillic (KOI8-R)
20871	IBM871	IBM EBCDIC (Icelandic)
20880	IBM880	IBM EBCDIC (Cyrillic, Russian)
20905	IBM905	IBM EBCDIC (Turkish)
20924	IBM00924	IBM Latin-1
20932	EUC-JP	Japanese (JIS 0208-1990 and 0212-1990)
20936	x-cp20936	GB2312-80 Chinese (simple)
20949	x-cp20949	Korean Wansung
21025	cp1025	IBM EBCDIC (Cyrillic, Serbian-Bulgarian)
21866	koi8-u	Cyrillic (KOI8-U)
28591	iso-8859-1	Western European (ISO)
28592	iso-8859-2	Central European (ISO)
28593	iso-8859-3	Latin 3 (ISO)
28594	iso-8859-4	Baltic (ISO)
28595	iso-8859-5	Cyrillic (ISO)
28596	iso-8859-6	Arabic (ISO)
28597	iso-8859-7	Greek (ISO)
28598	iso-8859-8	Hebrew (ISO-Visual)
28599	iso-8859-9	Turkish (ISO)
28603	iso-8859-13	Estonian (ISO)
28605	iso-8859-15	Latin 9 (ISO)
29001	x-Europa	Europe
38598	iso-8859-8-i	Hebrew (ISO-Logical)
50220	iso-2022-jp	Japanese (JIS)
50221	csISO2022JP	Japanese (JIS, 1 Byte Kana allowed)
50222	iso-2022-jp	Japanese (JIS, 1 Byte Kana allowed -SO/SI)
50225	iso-2022-kr	Korean (ISO)

50227	x-cp50227	ISO-2022 Chinese (simplified)
51932	euc-jp	Japanese (EUC)
51936	EUC-CN	Chinese simplified (EUC)
51949	euc-kr	Korean (EUC)
52936	hz-gb-2312	Chinese simplified (HZ)
54936	GB18030	GB18030 Chines simplified
57002	x-iscii-de	ISCII Devanagari
57003	x-iscii-be	ISCII Bengali
57004	x-iscii-ta	ISCII Tamil
57005	x-iscii-te	ISCII Telugu
57006	x-iscii as	ISCII Assamese
57007	x-iscii-or	ISCII Oriya
57008	x-iscii-ka	ISCII Kannada
57009	x-iscii-ma	ISCII Malayalam
57010	x-iscii-gu	ISCII Gujarati
57011	x-iscii-pa	ISCII Punjabi
65000	utf-7	Unicode (UTF-7)
65001	utf-8	Unicode (UTF-8)

newLine	Defines the type of line break. **Possible values** NewLineCRLF: CarriageReturn/LineFeed ("\r\n" - $0D $0A) NewLineLF: LineFeed ("\n" - $0A)

ConvertXDocToXps

Converts an XDOC file to an XPS file.

ConvertXDocToString
SYNTAX ConvertXDocToXps(xDocFileName, xpsOutputFileName)

ARGUMENTS	xDocFileName	The XDOC file you want to convert.
	xpsOutputFileName	The XPS file you want to create.

End

Ends processing and frees up the respective resources.

End	
SYNTAX	End()

LoadDocument

Loads the specified file and returns the IOdinStreamDocument interface with which, for example text output can be written to a page of the document.

For more information, refer to Context OdinStreamDocument.

LoadDocument		
SYNTAX	LoadDocument(fileName)	
ARGUMENTS	fileName	The file to load.
RETURN	IOdinStreamDocument interface with the following properties and functions. AddBookmark AddComment AddPJLCommand EditPage GetInputBin Image IsDuplex IsPortrait	

Line

Pagecount

Post

Rectangle

SetFont

SetInputBin

TextOut

NumberOfFilesConverted

Returns the number of files already converted and appended to the output file during batch conversion.

NumberOfFilesConverted	
SYNTAX	NumberOfFilesConverted()
RETURN	Number of converted files.

Start

Show All

Initializes processing for batch conversion. Use the AddFile and AddDocument functions to write 1-n documents to the output file.

Start		
SYNTAX	Start(outputFileName) Start(outputFileName, singlePage) Start(outputFileName, singlePage, convertOptions)	
ARGUMENTS	outputFileName	Name of the output file
	singlePage	TRUE = Write one output file per page according to the

following naming convention.

FILENAME_#PAGENUMBER#.outputformat

Example

OutputFilename_001.pdf

The default value is FALSE.

convertOptions

Used to specify options for conversion and metadata.

Options are defined in the form of a comma-separated string. Their use depends on the output format.

The parameters are optional.

Metadata

TITLE	Document title.
SUBJECT	Document subject.
AUTHOR	Document author.
KEYWORDS	Document keywords.

Options

TEXTNOTE_CHANGES	PDF 1.3/1.4 TRUE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `True`.

	FALSE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `False`.
CHANGES	PDF 1.3/1.4 TRUE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `True`. FALSE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `False`.
FILL	PDF 1.4 TRUE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `True`. FALSE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `False`.
COPY	PDF 1.4 TRUE = PDF security attributes for Document Assembly, Copy Content, Copy Content for Accessibility, Page Extraction, Comment, Sign and Create Template are set to `True`. FALSE = PDF security attributes

	for Document Assembly, Copy Content, Copy Content for Accessibility, Page Extraction, Comment, Sign and Create Template are set to `False`.
DIGITAL_COPY	PDF 1.4 TRUE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `True`. FALSE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `False`.
ASSEMBLE	PDF 1.4 TRUE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `True`. FALSE = PDF security attributes for Document Assembly, Page Extraction, Comment, Sign and Create Template are set to `False`. Assembling more than one document is not allowed.
ACCESSIBILITY	PDF 1.4 TRUE = PDF security attributes for Copy Content, Copy Content for Accessibility, Page Extraction are set to `True`. FALSE = PDF security attributes for Copy Content, Copy Content

	for Accessibility, Page Extraction are set to `False`.
DUPLEX_ ADDPAGE	TRUE = This option specifies whether an empty page should be added to a duplex document with an odd number of pages. FALSE = Default setting. This option sets simplex mode for the last page of a duplex document with an odd number of pages.
DUPLEX_ OPTIMIZE	TRUE = Duplex settings are only set in the output stream, when duplex mode changes. FALSE = Duplex settings are set in the output stream.
MAP_COLOR	TRUE/FALSE This parameter is used to define whether color mapping for the AFP-HighlightColor Definition is executed or not. The individual color definitions can be specified as MapColor_1 to MapColor_n. Example: Map_Color=TRUE,MapColor_ 1=-1,20,black,MapColor_ 2=0,20,red Note As MapColor parameters contain comma separated values themselves, the complete string must be enclosed in quotation

	marks. The structure is as follows: MapColor_???= HSB-Color,Threshold,ColorName HSB-Color=HSB Color Angle Threshold= Threshold value of the HSB color ColorName= Name of the color as defined in the AFP profile under HighlightColorList. The example above requires the following definitions in the AFP profile: </highlightcolorlist <highlightcolor name='red' type='OCA' number='0x0002' foreground-rgb="#00FF0000" /> <highlightcolor name='black' type='OCA' number='0x0008' foreground-rgb="#00000000" /> </highlightcolorlist> and <optimizepa value="never"/> <color value="FALSE" /> <usehighlightcolor value="TRUE" />

	<enablebilevelimage value="TRUE" />
SHOW_ BOOKMARKS	TRUE = Bookmarks show in the opened PDF. FALSE = Bookmarks do not show in the opened PDF.
OVERLAYS	TRUE = OVERLAYS are created. FALSE = Default. OVERLAYS are not created.
MERGE_FONTS	TRUE: Default FALSE
MERGE_ SPLITWORDS	TRUE = Default, PCL output optimization is activated. FALSE = PCL output optimization is deactivated. This option should be set to FALSE if you have character spacing problems in a generated PCL. The default value is TRUE.
ROTATE_ LANDSCAPE	TRUE = If page orientation is landscape, the page rotates by 90°. FALSE = The page does not rotate. The default value is TRUE.

Sample Script OdinFileConvert: 1 file

```
; get FileConverter component
OdinFileConverter = ModusProcess.GetComponent("OdinFileConvert")
```

```
;
StringListAdd("options", "USER_PASSWORD=ymM/Mgjelzq/rdPdf6q6rg==")
StringListAdd("options", "OWNER_PASSWORD=uiuCKFZ+QE1VHSQAA/xQQQ==")
StringListAdd("options", "LINEARIZED=true")
StringListAdd("options", "TITLE=document-title")
StringListAdd("options", "SUBJECT=document-subject")
StringListAdd("options", "AUTHOR=JKlinger")
StringListAdd("options", "KEYWORDS=document-keywords")
StringListAdd("options", "PRINT=true")
StringListAdd("options", "TEXTNOTE_CHANGES=false")
StringListAdd("options", "CHANGES=false")
StringListAdd("options", "FILL=false")
StringListAdd("options", "COPY=true")
StringListAdd("options", "DIGITAL_COPY=false")
StringListAdd("options", "ASSEMBLE=false")
StringListAdd("options", "ACCESSIBILITY=false")
;
options = StringListGetCommaText("options")

; convert single XPS file to PCL
OdinFileConverter.OutputFormat = "pcl"
OdinFileConverter.ConvertSingleFile("c:\temp\Document.xps", "c:\temp\output.pcl", false, options)
;
; convert single XPS file to Postscript
OdinFileConverter.OutputFormat = "pos"
OdinFileConverter.ConvertSingleFile("c:\temp\Document.xps", "c:\temp\output.pos", false, options)
;
; convert single XPS file to PDF
OdinFileConverter.OutputFormat = "pdf"
```

```
OdinFileConverter.ConvertSingleFile("c:\temp\Document.xps", "c:\temp\output.pdf", false, options)
```

Sample Script OdinFileConvert: 1-n Files

```
; Get component for converting
OdinFileConverter = ModusProcess.GetComponent("OdinFileConvert")
;
; Start batch conversion
OdinFileConverter.OutputFormat = "pdf"
OdinFileConverter.Start("c:\temp\output.pdf")
;
; Load document #1 (2 pages)
document = OdinFileConverter.LoadDocument("c:\temp\TestDocument1.xps")
;
; Create bookmarks on both pages and output an additional text
document.EditPage(0)
document.AddBookmark("doc1", "", "Document 1")
document.AddBookmark("doc1_page1", "doc1", "Page 1")
document.SetFont("Tahoma", 16)
document.TextOut(20.5, 10.5, "Document#1 Page#1", false, false)
document.Post()
;
document.EditPage(1)
document.AddBookmark("doc1_page2", "doc1", "Page 2", 100, 100)
document.TextOut(20.5, 10.5, "Document#1 Page#2", false, false)
document.Post()
;
; Insert document in the output file
OdinFileConverter.AddDocument(document)
;
; Document #2: as test inserting a document with AddFile in between, no bookmarks added
```

```
OdinFileConverter.AddFile("c:\temp\TestDocument2.xps")
;
; Add document #3
document = OdinFileConverter.LoadDocument("c:\temp\TestDocument3.xps")
;
; Create bookmarks on both pages and output an additional text
document.EditPage(0)
document.AddBookmark("doc3", "", "Dokument 3")
document.AddBookmark("doc3_page1", "doc3", "Seite 1")
document.SetFont("Tahoma", 16)
document.TextOut(20.5, 10.5, "Document#3 Page#1", false, false)
document.Post()
;
document.EditPage(1)
document.AddBookmark("doc3_page2", "doc3", "Seite 2", 100, 100)
document.TextOut(20.5, 10.5, "Document#3 Page#2", false, false)
document.Post()
;
; Insert document in the output file
OdinFileConverter.AddDocument(document)
;
; Finish batch conversion
OdinFileConverter.End()
```

Sample Script OdinFileConvert: xDoc to XPS

```
; get FileConverter component
OdinFileConverter = ModusProcess.GetComponent("OdinFileConvert")
;
; convert an xDoc file (DocX) to XPS
OdinFileConverter.ConvertXDocToXps("c:\temp\TestDocument.docx", "c:\temp\output.xps")
```

Convert Doc to Docx

ConvertDocToDocX

Converts and saves the specified DOC file to DOCX. You can invoke the function repeatedly.

ConvertDocToDocX

SYNTAX ConvertDocToDocX(docFileName, docXFilename)

ARGUMENTS

docFileName	DOC file.
docXFilename	DOCX file.

EndDocConvert

Ends conversion.

EndDocConvert

SYNTAX EndDocConvert()

StartDocConvert

Initializes Microsoft Word for conversion.

StartDocConvert

SYNTAX StartDocConvert(wordRestart, showWord)

ARGUMENTS

wordRestart	Defines the number of conversion operations Word performs before being automatically restarted.
showWord	Defines whether Word is visible during the operation.

OdinFilterManager

About OdinFilterManager

You must have an Odin license installed to use the OdinFilterManager component.

The OdinFilterManager component enables you to group job data sets to stacks or envelopes for further processing.

Properties of OdinFilterManager

Property	Meaning
DB Alias	Odin database on which the component executes its actions.
FilterScript	Script that executes in the filter component context. **Available context** OdinFilter

OdinFilterManager Functions

Run

Executes the script assigned to the FilterScript property.

Run	
SYNTAX	Run()

Sample Script OdinFilterManager

```
OdinFilter = ModusProcess.GetComponent("OdinFilterManager")

OdinFilter.Run()
```

OdinOnBaseArchive

About OdinOnBaseArchive

The OdinOnBaseArchive component enables you to archive documents into OnBase using the OnBase XML DIP importer.

Property	Description
DB Alias	Odin database on which the component executes its actions.
HotFolderDirectory	The component stores the archival documents and the required OnBase DIP file into this directory so they can be imported into OnBase archive with the XML DIP file importer. It can be a local drive or a network path (a mapped drive letter or UNC notation). **Notes** • Before setting a Hotfolder directory, ensure that the Windows user account possesses access rights to write to this folder. • If you specify a drive letter of a mapped network drive, ensure that the OnBase XML DIP importer can also use the same drive letter mapping.
License Filename	Path and name of the Compart .NET Toolkit license file, for example cpsdk.lic.
NextStackType	Specifies the next stack type set after successful execution. The following table displays all possible stack types. Any additionally required stack types must be defined in the file **ModusSuite.Common.PropertyEditors.dll.config**. **Default stack types**

Stack type	Name	Description
0	Print	Stack is ready to be printed to an online printer or email printer (online print).
10	PortoOptimization	Stack is ready for IT franking. This function is currently not supported.
20	Streaming	Stack is ready for streaming.
30	Convert	Stack is ready to be converted to the appropriate output format.
40	StreamPrint	Stack (stream) is ready for batch printing to an outsourcing printer.
50	Archive	Stack is ready for archiving.
99	Finish	Stack processing is finished.

Profile path	Profile of a user.
Trace level	Specifies which error categories are written to the trace file.
TraceFileName	Name of the Compart trace file

StackSelect	SQL select to identify the next stack to process.
	You can either specify the select statement directly or a reference to a lookup table, for example $OdinSQL.ArchiveStackSelect$.
	
	Example for a SQL Server Select statement
	`select top 1 ods_id from odin_stack with (updlock, holdlock) where ods_state = 0 and ods_stacktype = 50 and ods_lockby is null`
	
	Example for a Oracle Select statement
	`select ods_id from odin_stack where ods_state = 0 and ods_stacktype = 50 and ods_lockby is null and rownum = 1 for update`

.

OdinOnBaseArchive Functions

ArchiveStack

This function executes the SQL Select statement defined with property `StackSelect` on the Odin database defined with property DB Alias. The first selected stack record is processed and then the function returns the ID of the successfully archived stack.

This function has the following output files located in the Hotfolder path specified with property HotFolderDirectory.

- 1 OnBase XML DIP file that contains information for all archival documents of this stack
- 1-n archival document files.

If the stack does not contain any document configured for OnBase archival processing, the stack is successfully processed but no files are created in the hotfolder. If no stack record exists for archival processing, the function returns an empty string. This function can process a Direct Print or an Outsourcing stack.

ArchiveStack	
SYNTAX	ArchiveStack()
RETURN	String. The ID of the successfully archived stack or an empty string if no stack record was available for archiving.
EXAMPLE	`StackId = OdinOnBaseArchive.ArchiveStack()`

OdinPrint

About OdinPrint

The OdinPrint component enables you to output the spool files of stack jobs to a physical (online) printer.

You can assign the jobs of a single stack to different printers. The used printer and form are retrieved from the individual job dataset. Both the specified printer and form objects must exist in the repository.

Specific printer events execute according to the particular process steps carried out during job output.

The component supports the following scenarios.

- Processing the jobs of a specific stack.
- Processing stacks determined by a configurable select statement.

Properties of OdinPrint

Property	Description
DB Alias	Odin database on which the component executes its actions.
IsRetryable	An enumerator specifying whether the stack can be reprinted in the event of an error. **Possible values** OdinRetryOptions.NotRetryAble: The stack is set to locked and no attempt to reprint is made. OdinRetryOptions.RetryAbleContinue: The whole stack is reset to reprint. Only faulted or stack jobs not yet printed are processed. OdinRetryOptions.RetryAbleRestart: The whole stack is reset to reprint. All jobs of the stack are processed.
MaxRetryCo	Maximum number of retries, before the stack status sets to **Locked**.

unt

NextStackType

Specifies the next stack type set after successful execution.

The following table displays all possible stack types.

Any additionally required stack types must be defined in the file **ModusSuite.Common.PropertyEditors.dll.config**.

Default stack types

Stack type	Name	Description
0	Print	Stack is ready to be printed to an online printer or email printer (online print).
10	PortoOptimization	Stack is ready for IT franking. This function is currently not supported.
20	Streaming	Stack is ready for streaming.
30	Convert	Stack is ready to be converted to the appropriate output format.
40	StreamPrint	Stack (stream) is ready for batch printing to an outsourcing printer.
50	Archive	Stack is ready for archiving.
99	Finish	Stack processing is finished.

Retry Pause

Time in minutes before this stack is reprinted or reset through the **OdinDbTasks** component.

Stack Select

Defines the select statement to detect the next stack to be processed.

You can either specify the select statement or refer to a lookup table, for example $OdinSQL.PrintStackSelect$.

OdinPrint Events

Event	Meaning

OnAfterStack	The specified script is executed after the stack processing when calling the End function. The event corresponds to the **AfterStack** event of a printer in Modus 4. **Available context:** Context OdinCustomProcess
OnBeforeStack	The specified script is executed when calling the Start() function, before executing the stack processing. The event corresponds to the **AfterStack** event of a printer in Modus 4. **Available context:** **Context OdinCustomProcess**

OdinPrint Functions

AddBatchPrintJob

Adds a job to the internal list.

An exception is thrown if batch print is not initialized for the printer specified in the job, or the job is added redundantly.

AddBatchPrintJob		
SYNTAX	AddBatchPrintJob(job)	
ARGUMENTS	job	Job you want to insert.
EXAMPLE	job = MailJobEnumerator.Current OdinPrint.AddBatchPrintJob(job)	

BatchJobCount

Returns the current number of print jobs added to a specific printer by the AddBatchPrintJob function.

If you do not specify a printer name, the function returns the total number of print jobs added to all initialized batch prints.

BatchJobCount		
SYNTAX	BatchJobCount() BatchJobCount(printerName)	
ARGUMENTS	printerName	Printer name.
RETURN	Number of jobs.	
EXAMPLE	BatchJobCount(printerName)	

CurrentPrinter

Returns the name of the printer currently running a batch print job.

CurrentPrinter	
SYNTAX	CurrentPrinter()
RETURN	Printername

End

Ends component processing, sets the stack to the next stack type and, if relevant, executes a script assigned to the OnAfterStack event.

End		
SYNTAX	End() End(errorMessage)	
ARGUMENTS	errorMessage	Error message.

The following values are set.

ODS_Print_End:	Current date and time.
ODS_Delete_At:	Current date, time, and stack hold time.

If processing ends correctly, the following values are set.

ODS_PrintState:	2 (OdinPrintState.OK)
ODS_State:	NextStackState of the component.
ODS_StackType:	NextStackState of the component.

On error, the following values are set.

ODS_PrintState:	3 (OdinPrintState.Error)
ODS_State:	3 (OdinStackState.Locked)
ODS_MessageText:	Error message.

ExecuteBatchPrint

Executes all initialized batch print jobs and prints the jobs on the corresponding printer.

Prints on all printers initialized with InitializeBatchPrint().

All jobs added with AddBatchPrintJob() are printed.

ExecuteBatchPrint	
SYNTAX	ExecuteBatchPrint() ExecuteBatchPrint(printername)

ARGUMENTS	printername	Printer name. The specified printer must have been initialized with InitializeBatchPrint().
RETURN	TRUE = Printing successful FALSE = Error while printing	

GetCurrentStackRow

Show All

After calling Start() and before calling End() the function returns an object whose interface provides read access to the current stack dataset.

GetCurrentStackRow

SYNTAX GetCurrentStackRow()

RETURN Object interface that provides read access to the current stack dataset.

The following functions are available.

GetValue

Returns the value of the given column.

GetValue		
SYNTAX	GetValue(columnName)	
ARGUMENTS	columnName	Column name
RETURN	The column value.	

	If the data type of the database column is DATE, Datetime or any valid date type, the date read generally returns in the format MM/DD/YYYY hh:mm:ss.
EXAMPLE	GetObject("DateVar","MLDate") StringDateVar = CurrentJobRecord.GetValue("ODJ_create_at") ;Convert date to the required format with the help of the MLDate object IntDateVar = DateVar.DateStringToInt64("MM/dd/yyyy hh:mm:ss", StringDateVar) v_LETTERDATE= DateVar.Int64ToDateString("MM.dd.yyyy hh:mm:ss ", IntDateVar)

GetFieldNameEnumerator

Returns an enumerator.

GetFieldNameEnumerator

SYNTAX	GetFieldNameEnumerator()
RETURN	Enumerator with the following functions.

Current	Returns the current object.
MoveNext()	Moves the position to the next object to process. SYNTAX: MoveNext() RETURN: TRUE = Successful FALSE = The end of the collection is passed.
Reset	Resets the enumerator back to the beginning of the first element.

GetNextStack

Determines the next stack to convert according to the corresponding SQL, which is stored in the StackSelect property, and returns its ID.

GetNextStack	
SYNTAX	GetNextStack()
RETURN	ID of the stack found, or an empty string if none found.

GetPrinterEnumerator

Returns an enumerator for the printers used in a stack.

On loading, the printers used in a stack are sorted in the following order.

- Printer type (online printer first, then email printer)
- Printer priority for identical printer types
- Printer name for identical priorities

GetPrinterEnumerator	
SYNTAX	GetPrinterEnumerator()
RETURN	Enumerator with the following functions. Current — Returns the current object. MoveN — Moves the position to the next object to process.

ext()	SYNTAX MoveNext() RETURN TRUE = Successful FALSE = The end of the collection is passed.
Reset	Resets the enumerator to the beginning of the first element.

GetPrintJobEnumerator

Returns an enumerator for the jobs of a printer. Use the function GetValue() to retrieve job data.

GetPrintJobEnumerator

SYNTAX GetPrintJobEnumerator(printerName)

ARGUMENTS

printerName	Printer name.

RETURN Enumerator with the following functions.

Current	Returns the current object.
MoveNext()	Moves the position to the next object to process. SYNTAX MoveNext() RETURN TRUE = Successful FALSE = The end of the collection is passed.
Reset	Resets the enumerator to the beginning of the first element.

GetMailEnumerator

Returns an enumerator for the email address in the jobs of a printer.

The enumerator uses **Current** to retrieve an email address, with the help of an iterator it can iterate through the various email recipients in the jobs.

GetMailEnumerator

SYNTAX GetMailEnumerator(printerName)

ARGUMENTS

printerName	Printer name.

RETURN

Enumerator with the following functions.

Current	Returns the current object.
MoveNext()	Moves the position to the next object to process. SYNTAX MoveNext() RETURN TRUE = Successful FALSE = The end of the collection is passed.
Reset	Resets the enumerator to the beginning of the first element.

GetMailJobEnumerator

Returns an enumerator for the jobs of a printer with a specific email recipient. Use the GetValue() function to retrieve job data.

GetMailJobEnumerator

SYNTAX GetMailJobEnumerator(printerName, email)

X

ARGUMENTS

printerName	Printer name.
email	Email address.

RETURN

Enumerator with the following functions.

Current	Returns the current object.
MoveNext()	Moves the position to the next object to process.

SYNTAX	MoveNext()
RETURN	TRUE = Successful FALSE = The end of the collection is passed.

Reset	Resets the enumerator to the beginning of the first element.

InitializeBatchPrint

Initializes batch print for a specified printer. You can initialize only one batch print per printer. A second initialization raises an exception.

InitializeBatchPrint

SYNTAX

InitializeBatchPrint(printerName)

InitializeBatchPrint(printerName, useOneSpoolfile)

ARGUMENTS

printerName	Printer name.
useOneSpoolfile	TRUE = Creates a single printer spool file for all inserted jobs. FALSE = Jobs print in succession.

IsEmailPrinter

Checks whether the printer in question is an email printer.

IsEmailPrinter		
SYNTAX	GetSectionNames(printerName)	
ARGUMENTS	printerName	Printer name.
RETURN	TRUE = Printer is an email printer. FALSE = Printer is not an email printer.	

IsPrinterActive

Checks whether a printer is activated.

IsPrinterActive		
SYNTAX	IsPrinterActive(printerName)	
ARGUMENTS	printerName	Printer name.
RETURN	TRUE = The printer is activated. FALSE = The printer is deactivated due to an error.	

PrintJob

Prints a job to a printer. The printer name is taken from the data field ODJ_Printer_Name.

PrintJob		
SYNTAX	PrintJob(job)	
ARGUMENTS	job	Job to print.
RETURN	TRUE = Print successful.	

	FALSE = Error while printing. The error automatically inserts to the corresponding stack record and can be accessed through the LastErrorMessage property.

Start

Loads and initializes the given stack and executes a script assigned to the OnBeforeStack event.

The stack must be of the corresponding stack type to that required of the component. The status must be set to Wait.

Start		
SYNTAX	Start(stackId)	
ARGUMENTS	stackId	Stack ID

After successful loading, the stack is locked and the statuses set as follows.

ODS_LockBy	Name of the current user
ODS_State	1 (OdinStackState.InWork)
ODS_PrintState	1 (OdinPrintState.InWork)
ODS_Print_Start	Current date and time
ODS_MessageText	Empty string

Sample Script OdinPrint

```
; define whether print all jobs in 1 spool file or print separately
useOneSpoolfile = true
;
; initialize print components
OdinPrint = ModusProcess.GetComponent("OdinPrint")
;
; determine next stack
```

```
stackID = OdinPrint.GetNextStack()
;
; was a stack found ?
if (stackID <> "")
;
    ; Yes, now initialize stack print
    OdinPrint.Start(stackID)
    ;
    ; get a printer enumerator and print the jobs for each printer
    PrinterEnumerator = OdinPrint.GetPrinterEnumerator()
    ;
    ; are there more printers in the stack for which jobs are to be processed?
    nextPrinter = PrinterEnumerator.MoveNext()
    ;
    while (nextPrinter = true)
      ; yes, now print all jobs for this printer in one spool file
      printerName = PrinterEnumerator.Current
      ;
      ; Is the printer active?
      if (OdinPrint.IsPrinterActive(printerName))
        ; yes, is it an Email Printer.
        if (OdinPrint.IsEmailPrinter(printerName))
        ;
          ; yes, now initialize batch print for the Email Printer
          OdinPrint.InitializeBatchPrint(printerName)
          ;
          ; get an email enumerator for this printer
          MailEnumerator = OdinPrint.GetMailEnumerator(printerName)
          ;
          ; any more emails for this printer?
```

```
nextEmail = MailEnumerator.MoveNext()
;
while (nextEmail = true and OdinPrint.IsPrinterActive(printerName))
  ; get email address from enumerator
  emailAdr = MailEnumerator.Current
  ;
  ; get an mail job enumerator for this email address
  MailJobEnumerator = OdinPrint.GetMailJobEnumerator(printerName, emailAdr)
  ;
  ; and print all jobs for this Email Printer and for this email address
  nextMailJob = MailJobEnumerator.MoveNext()
  ;
  while (nextMailJob = true)
    ; get current job from enumerator
    job = MailJobEnumerator.Current
    ; as a test get job ID
    jobId = job.GetValue("ODJ_ID")
    ;
    ; insert job as BatchPrintJob
    OdinPrint.AddBatchPrintJob(job)
    ;
    ; process next Job
    nextMailJob = MailJobEnumerator.MoveNext()
  end-while
  ;
  ; any BatchPrintJobs found?
  ; If an empty string passed here, the number of jobs
  ; for *all* initialized BatchPrints is returned.
  printed = true
  ;
```

```
    if (OdinPrint.BatchJobCount(printerName) > 0)
      ; yes, now execute print for this printer
      printed = OdinPrint.ExecuteBatchPrint(printerName)
      ;
      if (printed = false)
        Protocol("Error in print to printer '{0}':", 1, printerName)
      end-if
      ; This function executes all initialized BatchPrints in the order
      ; they were initialized.
      ;OdinPrint.ExecuteBatchPrint()
    end-if
    ;
    ; If printed, check if there are more emails for the printer
    ; to process.
    if (printed = true)
      nextEmail = MailEnumerator.MoveNext()
    end-if
  end-while
else
  ; Initialize Batchprint for this Online-Printer
  ; and print all Jobs as one spool file!
  OdinPrint.InitializeBatchPrint(printerName, useOneSpoolfile)
  ; This call sets useOneSpoolfile to TRUE !
  ;OdinPrint.InitializeBatchPrint(printerName)
  ;
  ; get a job enumerator for this printer
  JobEnumerator = OdinPrint.GetPrintJobEnumerator(printerName)
  ;
  ; and insert or print all jobs
  nextJob = JobEnumerator.MoveNext()
```

```
;
while (nextJob = true)
  ; get current job from enumerator
  job = JobEnumerator.Current
  ; as a test get job ID
  jobId = job.GetValue("ODJ_ID")
  ;
  ; This function enables you to print the individual job directly
  ;OdinPrint.PrintJob(job)
  ;
  ; insert job as BatchPrintJob
  OdinPrint.AddBatchPrintJob(job)
  ;
  ; process next Job
  nextJob = JobEnumerator.MoveNext()
  ;
end-while
;
; any BatchPrintJobs found?
; If an empty string passed here, the number of jobs
; for *all* initialized BatchPrints is returned.
;
if (OdinPrint.BatchJobCount(printerName) > 0)
  ;
  ; yes, now execute print for this printer
  printed = OdinPrint.ExecuteBatchPrint(printerName)
  ;
  if (printed = false)
    Protocol("Error while printing on printer '{0}':", 1, printerName)
  end-if
```

```
            ; This function executes all initialized BatchPrints in the order
            ; they were initialized.
            ;OdinPrint.ExecuteBatchPrint()
          end-if
        end-if
      end-if
      ;
      ; are there any more printers that need to be processed?
      nextPrinter = PrinterEnumerator.MoveNext()
    end-while
    ;
    ; finish stack print
    OdinPrint.End()
;
end-if
```

OdinProcessImport

About OdinProcessImport

Use this component to import print processing data into the Odin database tables for further processing.

Properties of OdinProcessImport

Property	Description
CurrentProcessDocumentDirectory	Path to the documents the process currently imports.
CurrentProcessFileName	Original name of the import file currently processed.
CurrentProcessId	ID of the process currently imported.
DB Alias	Odin database on which the component executes its actions.
DocumentBaseDestinationDirectory	Defines the directory where document files store on import. You can either specify a full path statement or refer to an entry in a

lookup table.

If, for example, the entry **ImportPath** in the lookup table **Odin** contains the preferred import path, you must specify **$Odin.ImportPath$**.

The following directory structure is created below the directory defined.

[Time stamp of when the Import Component was started]

[Date of the day the directory was created]

[Hour of the day the directory was created]

[Directory with the name 1000, 2000, 3000]

In the following example, **DocumentBaseDestinationDirectory** is defined as **odinImport**.

On import of the first process, the following directory structure was created.

The import component moves the first 1000 process documents to the directory named **1000**.

Afterwards, a new directory is created.

If the current day and hour are still the same, the new directory is created below the directory **10** with the name **2000**.

If not, then a new day and/or hour directory with a **1000** subdirectory is created.

The document files stored in the directory structure are deleted when the component **OdinDbTasks** deletes finished (processed) stack datasets.

If empty directories result from the deletion process, these too are deleted.

LastErrorProcessFile	The path and name of the last process file that failed to import.
OdinSystemOId	Specifies the object ID of the system where Odin printers and forms load from. If not set, Odin objects load from the system the process executing the component was loaded from.
Synchronize	If more than one OdinImport component use the same import directory simultaneously, this property should be set to True. TRUE = A process file cannot be processed by more than one instance of OdinImport. FALSE = No synchronization measures are taken.

OdinProcessImport Functions

ImportProcessFile

Imports a process file.

ImportProcessFile

SYNTAX ImportProcessFile(processFile, documentFileDirectory)

ARGUMENTS

processFile	The file to import.
documentFileDirectory	Directory path where the document files belonging to the process XML file are stored.

RETURN ID of the imported process.

EXAMPLE
```
OdinImport = ModusProcess.GetComponent("OdinProcessImport")
ImportedProcessId = OdinImport.ImportProcessFile("c:\OdinImport\TestProcess.xml", "c:\OdinImport")
```

SearchAndImportProcessFile

Looks for files with the XML extension in the specified directory and imports them.

SearchAndImportProcessFile

SYNTAX SearchAndImportProcessFile(ProcessFileDirectory, DocumentBaseDirectory)

ARGUMENTS

ProcessFileDirectory	The directory to look for process files
DocumentBaseDirectory	Path of the directory to look for process document files in the respective subfolders. The association between a process file and its document directory is made by using the same name.

RETURN Process ID of the imported process, or an empty string if no process file exists

EXAMPLE OdinImport = ModusProcess.GetComponent("OdinProcessImport")

ImportedProcessId =OdinImport.SearchAndImportProcessFile("d:\OdinImport", "d:\OdinImport")

Sample Script OdinProcessImport

```
OdinImport = ModusProcess.GetComponent("OdinProcessImport")
;
processId = OdinImport.SearchAndImportProcessFile(SYSTEM.ImportDirectory, SYSTEM.ImportDirectory)
if (processId <> "")
  ;Process data imported... Repeat the loop
  SYSTEM.ExecuteLoop = true
else
  ;no data to import... no loop necessary
  SYSTEM.ExecuteLoop = false
end-if
```

Sample Script OdinProcessImport error handling

```
odinProcessImport = ModusProcess.GetComponent("OdinProcessImport")
try
  Protocol("----------",5 )
  processId = odinProcessImport.SearchAndImportProcessFile("D:\Temp\Single","D:\Temp\Single")
```

```
  Protocol("Odin Process imported: {0}" ,5, processId)
  OnError("ProcessHasNoJobs") ; Specific error - import file contains documents to import
    Protocol("Odin Process failed to import {2} file {0} - dir {1} "
,0,odinProcessImport.CurrentProcessFileName,odinProcessImport.CurrentProcessDocumentDirectory,odi
nProcessImport.CurrentProcessId  )
    Protocol(" The Process contains no documents!",0)
  OnError ; Generic error
    Protocol("Odin Process failed to import {2} file {0} - dir {1} "
,0,odinProcessImport.CurrentProcessFileName,odinProcessImport.CurrentProcessDocumentDirectory,odi
nProcessImport.CurrentProcessId  )
    Protocol("Import file {1} is now error file {0}" ,0,odinProcessImport.LastErrorProcessFile
,odinProcessImport.CurrentProcessFileName)
end-try
```

OdinRendition

About OdinRendition

The OdinRendition component enables you to convert document files of jobs to the Odin spool file format XPS. The XPS format is used to create print streams.

The component supports the following scenarios.

- Processing jobs of a specific process (stack / online print)
- Processing of jobs determined by a preconfigured select statement

The component supports the following input formats.

- DOCX files created by Content Composer
- Static DOCX files (converted to XPS by Microsoft Word so that the original layout is 100% guaranteed)
- PDF
- TIFF
- XPS

Notes

- You can print documents created with Content Composer directly to a printer, while third-party formats such as PDF start the corresponding program for the print operation.
- The spool file that is created is also saved in addition to the original document.

Properties of OdinRendition

Property	Meaning
DB Alias	Odin database on which the component executes its actions.
JobOrder	Defines the fields used for the current sorting. The following fields are set as default. ODJ_BUNDLE_SEQ ASC, ODJ_SEQUENCE ASC, ODJ_ID ASC
ProcessSelect	Specifies the select statement executed to detect the process to be handled. If the **CreateRenditionForProcess** function is called, you do not need to set this property. The select statement must specify an update lock to enable a parallel deployment of multiple rendition instances. For MSSQL servers, the select statement must contain `with (Updlock)`. For Oracle servers, the select statement must contain `for update`.
StackType	Defines the stack type for direct print stacks if direct prints are found and a new stack is created. The default value is 0. The following stack types can be set.

Print	0
Archive	50
Transfer	60
Finished	99 When stack type is set to Finished: • ODS_STATE (status) is set to **On Hold**. • ODS_DELETE_AT (delete date) is set to **now + hold time**.

The following stack types result in an error if you set them in a script.

PortoOptimization	10
Streaming	20
Convert	30
Stream Print	40

OdinRendition Events

Property	Meaning
OnAfterJobScript	This event is called after a job rendition has been created. The file name of the rendition is then available in the job dataset (ODJ_RENDITION). **Available context:** Context OdinRendition
OnBeforeJobScript	This event is executed before a job is processed. By using the attribute **CurrentJobRecord** and the functions GetValue() or SetValue(), a script assigned here has read and write access to the columns of the current job dataset. Any changes made to the job dataset here are taken into account while creating the rendition file and are saved to the database if job rendition was successful. **Available context:** Context OdinRendition

	Sample Script The following example checks whether the current job dataset contains a printer object ID. If not, an ID is set as it is required when creating a rendition. Printer = CurrentJobRecord.GetValue("ODJ_PRINTER_OID") if (Printer = "") CurrentJobRecord.SetValue("ODJ_PRINTER_OID", "test-pr-5") end-if
OnProcessEnd	This event is executed after all jobs of a process have been processed. **Available context:** Context OdinRendition **Properties** Only the following properties of the context **OdinRendition** are available: • CurrentProcessId • ProcessRow • JobCount **Functions** Only the following function of the context OdinRendition is available: • GetJobRow
OnProcessStart	This event is executed at the start of process handling. **Available context:** None

OdinRendition Functions

CreateRenditionForProcess

Creates the rendition (spool) files for the jobs / documents of the specified process.

CreateRenditionForProcess	
SYNTAX	CreateRenditionForProcess(processId)
ARGUMENTS	processId ID of the process to handle.
RETURN	ID of the direct print stack created for the handled process . An empty string returns if the process does not have a direct print stack because no direct print documents exist.
EXAMPLE	RenditionManager = ModusProcess.GetComponent("OdinRendition") DirectPrintStackId = RenditionManager.CreateRenditionForProcess("{002A2ACE-29A1-4B3E-A573-5B95ED0EE8B5}")

CreateRenditionForSelectedProcess

Determines the process to handle by way of the process select SQL.

Following this the same processing is done as in the CreateRenditionForProcess function.

The ID of the process to be handled is determined by executing the process select statement.

CreateRenditionForSelectedProcess	
SYNTAX	CreateRenditionForSelectedProcess()
RETURN	ID of the process handled or an empty string if no process exists.
EXAMPLE	RenditionManager = ModusProcess.GetComponent("OdinRendition") RenditionManager.ProcessSelect = "Select TOP 1 ODP_ID From Odin_Process where ODP_State = 0 and ODP_Lockby is Null Order By Odp_Create_At Asc" ProcId = RenditionManager.CreateRenditionForSelectedProcess()

Sample Script OdinRendition

```
rendition = ModusProcess.GetComponent("OdinRendition")
;rendition.ProcessSelect =
```

```
;"select top 1 odp_id from Odin_process with (updlock) where odp_state = 0 and odp_lockby is null " +
;"and odp_bundle_name = Invoice " +
;"order by odp_create_at asc"
;
if (processId <> "")
  ;create XPS files for the imported Process
  directPrintStackId = rendition.CreateRenditionForProcess(processId)
end-if
```

OdinStream

About OdinStream

You must have an Odin license installed to use the OdinStream component.

The OdinStream component enables you to create XML format stream files, their description files, and the corresponding protocol log files.

Accessing the Virtual Database Tables

Each script has access to the virtual database tables for the stack (**STACK**), the envelopes (**ENVELOPE**) and jobs of the envelope (**JOB**).

You can use these tables to determine information about stacks, envelopes, and jobs. This is done by using the functionality of the virtual database (VirtualDatabase).

Example of access to stack, envelope, and job columns

```
stackID = GetField("STACK", "ODS_ID")
envID = GetField("ENVELOPE", "ODE_ID")
jobID = GetField("JOB", "ODJ_ID")
```

Setting OMR Definitions

OMR codes are set in the stack created, for example in a filter using SetProperty("OMR_CODE").

Content Composer currently supports the following OMR codes.

- C4_OMR
- C4EF_OMR
- K34_OMR
- K34K_OMR
- BELL_OMR

- AS1A_OMR
- STIELOW_OMR
- SI68_OMR
- KERN_POST_OMR
- Kern Post 2

For each OMR definition, you can define the following settings.

OMR_POS_X / OMR_ POS_Y	Horizontal and vertical start position for the OMR code on paper in mm.
OMR_LINELENGTH / OMR_LINEHEIGHT	Length and width of the OMR lines in mm.
OMR_GAP	Space between each line in mm.
OMR_EDGE	Specifies whether the OMR code is printed on the long edge (=LONG) or short edge (=SHORT) of a page. Default: LONG.

If required, you can define varying configurations for a single OMR by specifying another configuration name separated by a dot.

Example

C4_OMR=OMR_POS_X=10,OMR_POS_Y=10,OMR_LineLength=5,OMR_LineHeight=0.5,OMR_Gap=5,OMR_Edge=Long

C4_OMR.ShortEdge=OMR_POS_X=10,OMR_POS_Y=10,OMR_LineLength=5,OMR_LineHeight=0.5,OMR_Gap=5,OMR_Edge=Short

OdinStream Output Files

The following XML files are created as an OPC-Package (=zipped):

- Stream file .str

The stream file contains the following XML files:

- /odin/Stream.xml
  This XML describes the structure of the stream and contains the "page overlays" in XML format.
- /Odin/StreamInfo.xml
  This XML contains information on streaming.
  Firstly, OMR code settings and secondly, information such as number of pages, sheets, envelopes, jobs as well as on the forms and virtual trays used by the stream.

Description file .dsc

Contains an XML with the data / datasets of the stack /envelope/documents.

- /odin/StreamDescription.xml
  This file stores the stream datasets.

Structure of the XML file

```
<streamdescription>
  <stack>
    <fields>
      <field name="Fieldname" value="Wert" />
      ...
    </fields>
    <envelopes>
      <envelope>
        <fields>
          <field name="Fieldname" value="Wert" />
          ...
        </fields>
        <jobs>
          <job>
            <fields>
              <field name="Fieldname" value="Wert" />
              ...
            </fields>
          </job>
        </jobs>
        ...
      </envelope>
      ...
    </envelopes>
  </stack>
```

```
</streamdescription>
Example
<?xml version="1.0" encoding="utf-8" ?>
<streamdescription>
<stack>
<fields>
  <field name="ODS_ID" value="28" />
  <field name="ODS_STATE" value="1" />
  <field name="ODS_PRIORITY" value="200" />
  <field name="ODS_CREATE_AT" value="10/23/2011 12:34:48" />
  <field name="ODS_CREATE_BY" value="user" />
  <field name="ODS_EDIT_AT" value="11/21/2011 16:12:34" />
  <field name="ODS_EDIT_BY" value="user" />
  <field name="ODS_PRINTSTATE" value="1" />
  <field name="ODS_PRINT_START" value="11/21/2011 16:26:13" />
  <field name="ODS_PRINT_END" value="11/21/2011 16:12:34" />
  <field name="ODS_LASTPRINTED" value="" />
  <field name="ODS_HOLDTIME" value="10" />
  <field name="ODS_DELETE_AT" value="11/22/2011 02:12:34" />
  <field name="ODS_STACKTYPE" value="20" />
  <field name="ODS_INITIATORTYPE" value="" />
  <field name="ODS_FILTERNAME" value="Postage3" />
  <field name="ODS_LASTARCHIVED" value="" />
  <field name="ODS_MESSAGETEXT" value="" />
  <field name="ODS_LOCKBY" value="user" />
  <field name="ODS_RETRY_AT" value="" />
  <field name="ODS_RETRY_COUNT" value="0" />
  <field name="ODS_PRINTSTREAM" value="d:\Import\output\28\28.str" />
  <field name="ODS_OMR_TYPE" value="C4_OMR.Test1" />
```

```
  <field name="ODS_STACKOBJECTSQL" value="" />
  <field name="ODS_PAPER_COUNT" value="2077" />
  <field name="ODS_PAGE_COUNT" value="2940" />
  <field name="ODS_JOB_COUNT" value="179" />
  <field name="ODS_ENVELOPE_COUNT" value="10" />
</fields>
<envelopes>
  <envelope>
    <fields>
      <field name="ODE_ID" value="7" />
      <field name="ODE_STACK_ID" value="28" />
      <field name="ODE_CREATE_AT" value="10/23/2011 12:34:34" />
      <field name="ODE_CREATE_BY" value="user" />
      <field name="ODE_STATE" value="0" />
      <field name="ODE_POSTAGE" value="Postage3" />
      <field name="ODE_WEIGHT" value="416,00000" />
      <field name="ODE_NAME" value="Envelope1" />
      <field name="ODE_DEVICE" value='Device x' />
      <field name="ODE_DVF" value="" />
      <field name="ODE_DVF_TT" value="" />
      <field name="ODE_DVF_MMJJ" value="" />
      <field name="ODE_DVF_ABRNR" value="" />
      <field name="ODE_DVF_ELIEPLZ" value="" />
      <field name="ODE_DVF_BRUTTOENTGELD" value="" />
      <field name="ODE_DVF_CURRENCY" value="" />
      <field name="ODE_OMR_ANNEX" value="0" />
      <field name="ODE_OMR_REJECTION" value="" />
      <field name="ODE_REC_NAME" value="Locksley Sir Robin of" />
      <field name="ODE_REC_AKZ" value="D" />
      <field name="ODE_REC_ZIP" value="71711" />
```

```
      <field name="ODE_REC_CITY" value="Steinheim" />
      <field name="ODE_REC_STREET" value="Castle Rock" />
      <field name="ODE_PAPER_COUNT" value="246" />
      <field name="ODE_PAGE_COUNT" value="350" />
      <field name="ODE_SEQUENCE" value="" />
      <field name="ODE_JOB_COUNT" value="21" />
    </fields>
    <jobs>
      <job>
        <fields>
          <field name="ODJ_ID" value="707" />
          <field name="ODJ_STACK_ID" value="" />
          <field name="ODJ_ENV_ID" value="7" />
          <field name="ODJ_SEQUENCE" value="1" />
          <field name="ODJ_CREATE_AT" value="10/23/2011 11:26:45" />
          <field name="ODJ_CREATE_BY" value="PAthena" />
          <field name="ODJ_STATE" value="1" />
          <field name="ODJ_ANNEX" value="" />
          <field name="ODJ_DOCNAME" value="" />
          <field name="ODJ_FILENAME" value="D:\Import\Bin\071023_
112908421\071023\11\1000\707.docx" />
          <field name="ODJ_RENDITION" value="D:\Import\Bin\071023_
112908421\071023\11\1000\707.xps" />
          <field name="ODJ_NUMBER_OF_COPIES" value="0" />
          <field name="ODJ_DUPLEX" value="0" />
          <field name="ODJ_OMR_CHANNEL" value="" />
          <field name="ODJ_ARCHIVED" value="" />
          <field name="ODE_REC_NAME" value="Locksley Sir Robin of" />
          <field name="ODJ_REC_AKZ" value="D" />
          <field name="ODJ_REC_ZIP" value="71711" />
```

```
        <field name="ODJ_REC_CITY" value="Stonehome" />
        <field name="ODJ_REC_STREET" value="Castle Rock" />
        <field name="ODJ_PROCESS_ID" value="{683de553-a858-4934-9581-8c6b9bf0a98d}" />
        <field name="ODJ_BUNDLE_SEQ" value="0" />
        <field name="ODJ_BUNDLE_COUNT" value="0" />
        <field name="ODJ_BUNDLE_NAME" value="" />
        <field name="ODJ_BUNDLE_OID" value="" />
        <field name="ODJ_VSNR" value="" />
        <field name="ODJ_PARNR" value="" />
        <field name="ODJ_IS_COPY" value="N" />
        <field name="ODJ_DOCTITLE" value="" />
        <field name="ODJ_PAPER_COUNT" value="24" />
        <field name="ODJ_PAGE_COUNT" value="24" />
        <field name="ODJ_DIRECTPRINT" value="0" />
        <field name="ODJ_PRINTER_OID" value="odin-opr-14" />
        <field name="ODJ_PRINTER_NAME" value="PrinterOutsourcing" />
        <field name="ODJ_FORM_OID" value="odin-fo-9" />
        <field name="ODJ_FORM_NAME" value="Form1" />
        <field name="ODJ_TO_ARCHIVE" value="" />
        <field name="ODJ_DOCTYPE" value="1" />
        <field name="ODJ_DOC_OID" value="" />
        <field name="ODJ_SORTING" value="" />
        <field name="ODJ_GROUPING" value="" />
        <field name="ODJ_LOCKBY" value="" />
        <field name="ODJ_MESSAGETEXT" value="" />
        <field name="ODJ_TIFF_LAYOUT" value="" />
      </fields>
    </job>
  </envelope>- <fields>  <field name="ODE_STACK_ID" value="28" />
</envelopes>- <fields>  <field name="ODE_STACK_ID" value="28" />
```

</stack>

</streamdescription>- <fields> <field name="ODE_STACK_ID" value="28" />

Properties of OdinStream

Show All

Property	Description
CreateDescriptionFile	TRUE = Write datasets to a separate description file (.dsc). FALSE = Do not write datasets to a separate description file (.dsc).
DB Alias	Odin database on which the component executes its actions.
ExtendedText	Specifies the lookup table for the additional texts printed on the documents. You can define additional texts for stacks, envelopes and documents and again for either portrait or landscape layouts. **Additional Texts (ExtendedText)** These additional texts are defined in lookup tables assigned to the **ExtendedText** property of a streaming component. You can define lookup values for stack and envelope cover pages as well as for documents and again for either portrait or landscape layouts.

ExtendFlyLeafStack_Portrait_Def	Additional text for stack cover page portrait layout
ExtendFlyLeafStack_Landscape_Def	Additional text for stack cover page landscape layout
ExtendFlyLeafEnvelope_Portrait_Def	Additional text for envelope cover page portrait layout
ExtendFlyLeafEnvelope_Landscape_Def	Additional text for envelope cover page landscape layout
ExtendAllPages_Portrait_Def	Additional text for document page portrait layout
ExtendAllPages_Landscape_Def	Additional text for document page landscape layout

Currently text output (Type=Text), Lines (Type=LINE), rectangles (Type=RECTANGLE) and images (Type=IMAGE) are supported.

You can enumerate the entries to enable multiple outputs.

Example

ExtendAllPages_Portrait_
Def=TYPE=TEXT,XPOS=25.5,YPOS=280.5,FONTSIZE=6,BOLD=N,Italic=N,FONTNAME=ARIAL,Angle=0,TEXT="%S.ODS_ID%"

ExtendAllPages_Portrait_
Def2=TYPE=TEXT,XPOS=22.5,YPOS=260.5,FONTSIZE=6,BOLD=N,Italic=Y,FONTNAME=ARIAL,Angle=0,TEXT=2nd text

Text (Type=Text)

COLOR	Color of the text (Default: Black)
LAYER	Defines the layer of the output. 0 or foreground = Foreground layer. Drawing is done on top of the original document. 1 or background = Background layer. Drawing is done behind the original document.
XPOS/YPOS	Specifies the X / Y text position in mm. The default value is 10 mm.
FONTSIZE	Size of the font. The default value is 6.
BOLD	Y = Bold N = Not bold The default value is N.
ITALIC	Y = Italic N = Not italic The default value is N.

FONTNAME	Name of the font used for text output. The default value is Arial.
ANGLE	Angle of the text. The default value is 0.
TEXT	Text to be written. If the text contains spaces, you must define the text as follows. "TEXT=This is a text with spaces!" Placeholders in the text are enclosed by %. Specify placeholder for values from a database as follows. %S.Field name% - Field in the stack table "Odin_Stack" %E.Field name% - Field in the envelope table "Odin_Envelope" %J.Field name% - Field in the job table "Odin_Job"

In addition, you can also use all properties accessed by the GetProperty function of the OdinStream context.

Example

ExtendFlyLeafStack_Portrait_
Def=TYPE=TEXT,XPOS=25,YPOS=280,FONTSIZE=6,BOLD=N,Italic=N,FONTNAME=ARIAL,Angle=0,"TEXT=Deckblatt Stapel #%S.ODS_ID%"

ExtendFlyLeafEnvelope_Portrait_
Def=TYPE=TEXT,XPOS=25,YPOS=280,FONTSIZE=6,BOLD=N,Italic=N,FONTNAME=ARIAL,Angle=0,"TEXT=Deckblatt Kuvert #%E.ODE_ID%"

ExtendAllPages_Portrait_
Def=TYPE=TEXT,XPOS=10,YPOS=260,FONTSIZE=6,BOLD=N,Italic=N,FONTNAME=Arial,Angle=270,TEXT=%S.ODS_ID%/%E.ODE_ID%/%J.ODJ_ID%

ExtendAllPages_Portrait_
Def2=TYPE=TEXT,XPOS=28,YPOS=290,FONTSIZE=6,BOLD=N,Italic=N,FONTNAME=Arial,Angle=0,"TEXT=Envelope#%StackEnvelope%/%TotalStackEnvelopes% - Env_job#%EnvelopeJob%/%TotalEnvelopeJobs% - Env_Page#%EnvelopePage%/%TotalEnvelopePages% Page#%JobPage%/%TotalJobPages%"

ExtendAllPages_Landscape_
Def=TYPE=TEXT,XPOS=39,YPOS=10,FONTSIZE=6,BOLD=N,Italic=N,FONTNAME=Arial,Angle=180,TEXT=%S.ODS_ID%/%E.ODE_ID%/%J.ODJ_ID%

ExtendAllPages_Landscape_
Def2=TYPE=TEXT,XPOS=8,YPOS=28,FONTSIZE=6,BOLD=N,Italic=N,FONTNAME=Ari

al,Angle=270,"TEXT=Envelope#%StackEnvelope%/%TotalStackEnvelopes% - Env_
job#%EnvelopeJob%/%TotalEnvelopeJobs% - Env_
Page#%EnvelopePage%/%TotalEnvelopePages% Page#%JobPage%/%TotalJobPages%"

Line (Type=LINE)

XPOS/YPOS	Specifies the X / Y start position in mm. The default value is 10 mm.
XPOS2/YPOS2	Specifies the X / Y end position in mm. The default value is 10 mm.
LineWidth	Specifies line weight in mm. The default value is 1 mm.
LineStyle	Specifies line style. The default value is 0. 0 or solid = Solid 1 or dashed = Dashed 2 or dotted = Dotted
LineColor	Line color
Layer	Defines the layer of the output. 0 or foreground = Foreground layer. Drawing is done on top of the original document. 1 or background = Background layer. Drawing is done behind the original document.

Example

ExtendAllPages_Portrait_
Def3=TYPE=LINE,XPOS=10,YPOS=10,XPOS2=60,YPOS2=60,LineWidth=0.5,LineStyle=2,LineColor=Red,Layer=1

ExtendAllPages_Portrait_
Def3=TYPE=LINE,XPOS=10,YPOS=10,XPOS2=60,YPOS2=60,LineWidth=0.5,LineStyle=dotted,LineColor=Red,Layer=background

Rectangle (Type=RECTANGLE)

XPOS/YPOS	Specifies the X / Y position in mm. The default value is 10 mm.
Width/Height	Width/height of the rectangle in mm. The default value is 10 mm.
LineWidth	Specifies line weight in mm. The default value is 1 mm.
LineStyle	Specifies line style (Default: 0=Solid) 0 or solid = Solid 1 or dashed = Dashed 2 or dotted = Dotted
LineColor	Color of the border. The default value is black.
Fill	TRUE = The rectangle is filled with color. FALSE = The rectangle is not filled with color. The default value is false.
FillColor	The color the rectangle is filled with.
Layer	Defines the layer of the output. 0 or foreground = Foreground layer. Drawing is done on top of the original document. 1 or background = Background layer. Drawing is done behind the original document.

Example

ExtendAllPages_Portrait_
Def4=Type=RECTANGLE,XPOS=20,YPOS=20,Width=10,Height=10,LineWidth=0.5,LineStyle=1,LineColor=Black,fill=true,fillColor=yellow,Layer=0

ExtendAllPages_Portrait_
Def4=Type=RECTANGLE,XPOS=20,YPOS=20,Width=10,Height=10,LineWidth=0.5,LineS

tyle=dashed,LineColor=Black,fill=true,fillColor=yellow,Layer=foreground

Image (Type=IMAGE)

XPOS/YPOS	Specifies the X / Y position in mm The default value is 10 mm.
File	Name of the image file to be painted.

Example

ExtendAllPages_Portrait_Def5=Type=IMAGE,XPOS=40,YPOS=40,file=d:\temp\pa_logo.bmp,Layer=1

You have to define colors as a string. Alpha-Transparency is not currently supported during converting. If 255 (FF) is specified for the text for Alpha-Transparency, the text is not transparent otherwise the transparency is drawn in the corresponding color.

Possible value

The HTML color name or the ARGB value in format #AARRGGBB (AA=Alpha-Transparency, RR=Red, GG=Green, BB=Blue).

HTML color name	ARGB value
AliceBlue	#FFF0F8FF
AntiqueWhite	#FFFAEBD7
Aqua	#FF00FFFF
Aquamarine	#FF7FFFD4
Azure	#FFF0FFFF
Beige	#FFF5F5DC

Bisque	#FFFFE4C4
Black	#FF000000
BlanchedAlmond	#FFFFEBCD
Blue	#FF0000FF
BlueViolet	#FF8A2BE2
Brown	#FFA52A2A
BurlyWood	#FFDEB887
CadetBlue	#FF5F9EA0
Chartreuse	#FF7FFF00
Chocolate	#FFD2691E
Coral	#FFFF7F50
CornflowerBlue	#FF6495ED
Cornsilk	#FFFFF8DC
Crimson	#FFDC143C
Cyan	#FF00FFFF
DarkBlue	#FF00008B
DarkCyan	#FF008B8B
DarkGoldenrod	#FFB8860B
DarkGray	#FFA9A9A9
DarkGreen	#FF006400
DarkKhaki	#FFBDB76B

DarkMagenta	#FF8B008B
DarkOliveGreen	#FF556B2F
DarkOrange	#FFFF8C00
DarkOrchid	#FF9932CC
DarkRed	#FF8B0000
DarkSalmon	#FFE9967A
DarkSeaGreen	#FF8FBC8B
DarkSlateBlue	#FF483D8B
DarkSlateGray	#FF2F4F4F
DarkTurquoise	#FF00CED1
DarkViolet	#FF9400D3
DeepPink	#FFFF1493
DeepSkyBlue	#FF00BFFF
DimGray	#FF696969
DodgerBlue	#FF1E90FF
Firebrick	#FFB22222
FloralWhite	#FFFFFAF0
ForestGreen	#FF228B22
Fuchsia	#FFFF00FF
Gainsboro	#FFDCDCDC
GhostWhite	#FFF8F8FF

Gold	#FFFFD700
Goldenrod	#FFDAA520
Gray	#FF808080
Green	#FF008000
GreenYellow	#FFADFF2F
Honeydew	#FFF0FFF0
HotPink	#FFFF69B4
IndianRed	#FFCD5C5C
Indigo	#FF4B0082
Ivory	#FFFFFFF0
Khaki	#FFF0E68C
Lavender	#FFE6E6FA
LavenderBlush	#FFFFF0F5
LawnGreen	#FF7CFC00
LemonChiffon	#FFFFFACD
LightBlue	#FFADD8E6
LightCoral	#FFF08080
LightCyan	#FFE0FFFF
LightGoldenrodYellow	#FFFAFAD2
LightGray	#FFD3D3D3
LightGreen	#FF90EE90

LightPink	#FFFFB6C1
LightSalmon	#FFFFA07A
LightSeaGreen	#FF20B2AA
LightSkyBlue	#FF87CEFA
LightSlateGray	#FF778899
LightSteelBlue	#FFB0C4DE
LightYellow	#FFFFFFE0
Lime	#FF00FF00
LimeGreen	#FF32CD32
Linen	#FFFAF0E6
Magenta	#FFFF00FF
Maroon	#FF800000
MediumAquamarine	#FF66CDAA
MediumBlue	#FF0000CD
MediumOrchid	#FFBA55D3
MediumPurple	#FF9370DB
MediumSeaGreen	#FF3CB371
MediumSlateBlue	#FF7B68EE
MediumSpringGreen	#FF00FA9A
MediumTurquoise	#FF48D1CC
MediumVioletRed	#FFC71585

MidnightBlue	#FF191970
MintCream	#FFF5FFFA
MistyRose	#FFFFE4E1
Moccasin	#FFFFE4B5
NavajoWhite	#FFFFDEAD
Navy	#FF000080
OldLace	#FFFDF5E6
Olive	#FF808000
OliveDrab	#FF6B8E23
Orange	#FFFFA500
OrangeRed	#FFFF4500
Orchid	#FFDA70D6
PaleGoldenrod	#FFEEE8AA
PaleGreen	#FF98FB98
PaleTurquoise	#FFAFEEEE
PaleVioletRed	#FFDB7093
PapayaWhip	#FFFFEFD5
PeachPuff	#FFFFDAB9
Peru	#FFCD853F
Pink	#FFFFC0CB
Plum	#FFB0E0E6

Purple	#FF800080
Red	#FFFF0000
RosyBrown	#FFBC8F8F
RoyalBlue	#FF4169E1
SaddleBrown	#FF8B4513
Salmon	#FFFA8072
SandyBrown	#FFF4A460
SeaGreen	#FF2E8B57
SeaShell	#FFFFF5EE
Sienna	#FFA0522D
Silver	#FFC0C0C0
SkyBlue	#FF87CEEB
SlateBlue	#FF6A5ACD
SlateGray	#FF708090
Snow	#FFFFFAFA
SpringGreen	#FF00FF7F
SteelBlue	#FF4682B4
Tan	#FFD2B48C
Teal	#FF008080
Thistle	#FFD8BFD8
Tomato	#FFFF6347

Turquoise	#FF40E0D0
Violet	#FFEE82EE
Wheat	#FFF5DEB3
White	#FFFFFFFF
WhiteSmoke	#FFF5F5F5
Yellow	#FFFFFF00
YellowGreen	#FF9ACD32

NextStackType

Specifies the next stack type set after successful execution.

The following table displays all possible stack types.

Any additionally required stack types must be defined in the file **ModusSuite.Common.PropertyEditors.dll.config**.

Default stack types

Stack type	Name	Description
0	Print	Stack is ready to be printed to an online printer or email printer (online print).
10	PortoOptimization	Stack is ready for IT franking. This function is currently not supported.
20	Streaming	Stack is ready for streaming.
30	Convert	Stack is ready to be converted to the appropriate output format.
40	StreamPrint	Stack (stream) is ready for batch printing to an outsourcing printer.

50	Archive	Stack is ready for archiving.
99	Finish	Stack processing is finished.

OMR

Specifies the lookup table for the OMR definitions.

OMR-Definition

OMR codes are set in the stack created, for example in a filter through SetProperty("OMR_CODE").

All position and size definitions for an OMR code are made in mm.

The following OMR codes are currently supported:

C4_OMR

C4EF_OMR

K34_OMR

K34K_OMR

BELL_OMR

AS1A_OMR

STIELOW_OMR

SI68_OMR

KERN_POST_OMR

Kern Post 2

The following settings can be defined for each OMR definition:

OMR_POS_X / OMR_POS_Y	Horizontal and vertical start position for the OMR code on paper.
OMR_ LINELENGTH / OMR_ LINEHEIGHT	Length and width of the OMR lines.
OMR_GAP	Space between each line
OMR_EDGE	Specifies whether the OMR code is printed on the long edge (=LONG) or short edge (=SHORT) of a page.

	Default: LONG If varying configurations for a single OMR type are required, they can be defined by specifying another configuration name separated by a dot: OMR_TYP.CONFIG Example C4_OMR=OMR_POS_X=10,OMR_POS_Y=10,OMR_LineLength=5,OMR_ LineHeight=0.5,OMR_Gap=5,OMR_Edge=Long C4_OMR.ShortEdge=OMR_POS_X=10,OMR_POS_Y=10,OMR_LineLength=5,OMR_ LineHeight=0.5,OMR_Gap=5,OMR_Edge=Short
StackSelect	SQL statement to identify the next stack to be processed. Either specify the select statement directly or refer to a lookup table, for example $OdinSQL.PrintStackSelect$.
StreamOutputPath	Output directory for the stream-/description file. The directory in which a subdirectory with the name of the stack ID is created and the stream file is saved to.

OdinStream Events

Show All

Event	Meaning
OnDocumentStream	This event is called for each document and offers the possibility to modify document pages. Available context: Context OdinStreamDocument Sample Script Event*OnDocumentStream*with Bookmarks ; ; Test access to database tables ods_id = CurrentStack.GetValue("ODS_ID") ode_id = CurrentEnvelope.GetValue("ODE_ID") odj_id = CurrentJob.GetValue("ODJ_ID")

	;
	; Set Bookmark for the 1. job page.
	jobBookmark = FormatStr("Job %s", odj_id)
	envelopeBookmark = FormatStr("Envelope %s", ode_id)
	;
	x.EditPage(0)
	streamdoc.AddBookmark(jobBookmark, envelopeBookmark, jobBookmark, 0, 0)
	streamdoc.Post()
	;
	pagenumber = 0
	while (pagenumber < streamdoc.PageCount)
	  streamdoc.EditPage(pagenumber)
	  ;
	  ; sets for each page a Bookmark as child of Job-Bookmark
	  pageBookmark = FormatStr("Page %d", pagenumber + 1)
	  pageBookmarkId = FormatStr("%s_%s", jobBookmark, pageBookmark)
	  streamdoc.AddBookmark(pageBookmarkId, jobBookmark, pageBookmark, 0, 0)
	  ;
	  streamdoc.Post()
	  pagenumber = pagenumber + 1
	;
	end-while
	;

OnEnvelopeCoverPage	This event is called for each envelope to create a cover page for an envelope and add it to the stream. Available context: Context OdinStream

Sample Script Event *OnEnvelopeCoverPage* with Bookmarks

```
; load empty cover page. The trays firstbin/nextbin must exists
document = stream.LoadDocument("c:\PAG\Docs\Empty_Page.xps",
OdinDuplexSetting.Simplex, "firstbin","nextbin")
;
; Determine envelope information
envPages = stream.GetProperty("TotalEnvelopePages")
envJobs = stream.GetProperty("TotalEnvelopeJobs")
;
ods_id = CurrentStack.GetValue("ODS_ID")
ode_id = CurrentEnvelope.GetValue("ODE_ID")
;
; as a test, get data from the VirtualDatabase
zip = GetField("ENVELOPE", "ODE_REC_ZIP")
city = GetField("ENVELOPE", "ODE_REC_CITY")
country = GetField("ENVELOPE", "ODE_REC_AKZ")
;
AdrName = GetField("ENVELOPE", "ODE_REC_NAME")
AdrStreet = GetField("ENVELOPE", "ODE_REC_STREET")
AdrCity = formatstr("%s - %s %s", country, zip, city)
EnvId = FormatStr("%s", ode_id)
EnvJobPages = formatstr("%d/%d", envJobs, envPages)
;
stackBookmark = FormatStr("Stapel %s", ods_id)
envelopeBookmark = FormatStr("Envelope %s", ode_id)
coverPageBookmark = formatstr("Envelope_Cover_Page_%s", EnvId)
;
; Print information on the cover page
document.EditPage(0)
stackBookmark = FormatStr("Stapel %s", ods_id)
```

```
envelopeBookmark = FormatStr("Envelope %s", ode_id)
document.AddBookmark(envelopeBookmark, stackBookmark,
envelopeBookmark, 0, 0)
document.AddBookmark(coverPageBookmark, envelopeBookmark,
"Kuvertdeckblatt", 0, 0)
;
document.SetFont("Tahoma", 16)
document.TextOut(20.5, 35, "EnvelopeCoverPage", false, false)
document.Line(20.5, 37, 70, 37, 0.5, OdinLineStyle.Solid, "black")
;
document.SetFont("Tahoma", 12)
document.TextOut(20.5, 42, "Envelope-ID:", false, false)
document.TextOut(20.5, 46, "Number Jobs/Pages:", false, false)
;
document.TextOut(65, 42, EnvId, true, false, "Gray", 0)
document.TextOut(65, 46, EnvJobPages, true, false, "Gray", 0)
;
document.TextOut(20.5, 55, "Recipient-Address:", true, false)
document.TextOut(20.5, 62, AdrName, false, false, "Gray", 0)
document.TextOut(20.5, 66, AdrStreet, false, false, "Gray", 0)
document.TextOut(20.5, 70, AdrCity, false, false, "Gray", 0)
;
document.Image(10, 5, "d:\temp\2_compart-logo-small.gif")
document.Post()
;
; Add cover page to stream
stream.AppendDocument(document, true, false)
;
; Navigate to the next envelope for VirtualDB Test/GetField
Next("Envelope"))
;
```

OnStackCoverPage	This event is called at the beginning of the stack to create a cover for the stack and add it to the stream. Available context: Context OdinStream Sample Script Event *OnStackCoverPage* with Bookmarks ; load empty cover page. The trays firstbin/nextbin must exists document = stream.LoadDocument("c:\PAG\Docs\Empty_Page.xps", OdinDuplexSetting.Simplex, "firstbin","nextbin") ; ; Determine stack information TotalPages = stream.GetProperty("TotalStackPages") TotalPapers = stream.GetProperty("TotalStackPapers") TotalEnvelopes = stream.GetProperty("TotalStackEnvelopes") TotalJobs = stream.GetProperty("TotalStackJobs") ; ods_id = CurrentStack.GetValue("ODS_ID") ; StackId = FormatStr("%s", ods_id) EnvJobCount = FormatStr("%d/%d", TotalEnvelopes, TotalJobs) PageCount = FormatStr("%d/%d", TotalPages, TotalPapers) ; ; Print information on the cover page document.EditPage(0) ; ; set Bookmark for stack stackBookmark = FormatStr("Stapel %s", ods_id) document.AddBookmark(stackBookmark, "", stackBookmark, 0, 0) document.AddBookmark("StackCoverPage", stackBookmark, "StackCoverPage", 0, 0) ;

	document.SetFont("Tahoma", 16) document.TextOut(20.5, 35, "StackCoverPage", false, false) document.Line(20.5, 37, 60, 37, 0.5, OdinLineStyle.Solid, "black") ; document.SetFont("Tahoma", 12) document.TextOut(20.5, 42, "stack-ID:", false, false) document.TextOut(20.5, 46, "Number Envelopes/Jobs:", false, false) document.TextOut(20.5, 50, "Number Pages/Sheets:", false, false) ; document.TextOut(65, 42, StackId, true, false, "Gray", 0) document.TextOut(65, 46, EnvJobCount, true, false, "Gray", 0) document.TextOut(65, 50, PageCount, true, false, "Gray", 0) document.Image(10, 5, "d:\temp\2_compart-logo-small.gif") ; document.Post() ; ; Add cover page to stream stream.AppendDocument(document, true, false)
OnStackEnd	This event is called after stack processing / streaming according to the case in hand. Available context: Context OdinStream
OnStackStart	This event is called after stack processing / streaming according to the case in hand. Available context: Context OdinStream

OdinStream Functions

AddBookmark

Adds a bookmark to a document. If the output format is PDF, a bookmark is created for every bookmark defined. The IDs must be unique for each bookmark.

AddBookmark

SYNTAX

AddBookmark(string id, string parentId, string title)

AddBookmark(string id, string parentId, string title, double x, double y)

ARGUMENTS

id	Bookmark ID
parentId	An empty string here means the bookmark is inserted at the highest level. Otherwise, the bookmark is inserted as a child of the bookmark specified.
title	Text for the bookmark.
x	X-position on the page.
y	Y-position on the page.

EXAMPLE

```
document.AddBookmark("doc1_page1", "doc1", "Page 1")
document.AddBookmark("doc1_page2", "doc1", "Page 2", 0, 0)
```

End

Ends component processing, sets the stack to the next stack type and, if relevant, executes a script assigned to the OnStackEnd event.

End

SYNTAX

End()

End(errorMessage)

ARGUMENTS

errorMessage	Error message.

The following values are set.

ODS_Print_End	Current date and time.
ODS_Delete_At	Current date, time, and stack hold time.

If processing ends correctly, the following values are set.

ODS_PrintState	2 (OdinPrintState.OK)
ODS_State	NextStackState of the component.
ODS_StackType	NextStackState of the component.

On error, the following values are set.

ODS_PrintState	3 (OdinPrintState.Error)
ODS_State	3 (OdinStackState.Locked)
ODS_MessageText	Error message.

GetCurrentStackRow

Show All

After calling Start() and before calling End() the function returns an object whose interface provides read access to the current stack dataset.

GetCurrentStackRow	
S Y	GetCurrentStackRow()

NTAX

RETURN

Object interface that provides read access to the current stack dataset.

The following functions are available.

GetValue

Returns the value of the given column.

GetValue	
SYNTAX	GetValue(columnName)
ARGUMENTS	columnName Column name
RETURN	The column value. If the data type of the database column is DATE, Datetime or any valid date type, the date read generally returns in the format MM/DD/YYYY hh:mm:ss.
EXAMPLE	GetObject("DateVar","MLDate") StringDateVar = CurrentJobRecord.GetValue("ODJ_create_at") ;Convert date to the required format with the help of the MLDate object IntDateVar = DateVar.DateStringToInt64("MM/dd/yyyy hh:mm:ss", StringDateVar) v_LETTERDATE= DateVar.Int64ToDateString("MM.dd.yyyy hh:mm:ss ", IntDateVar)

GetFieldNameEnumerator

Returns an enumerator.

GetFieldNameEnumerator	
SYNT	GetFieldNameEnumerator()

AX

RETURN Enumerator with the following functions.

Current	Returns the current object.
MoveNext()	Moves the position to the next object to process.

SYNTAX	MoveNext()
RETURN	TRUE = Successful FALSE = The end of the collection is passed.

Reset	Resets the enumerator back to the beginning of the first element.

GetEnvelopeEnumerator

Returns an envelope enumerator. Use the GetValue() function to retrieve envelope data.

GetEnvelopeEnumerator

SYNTAX GetEnvelopeEnumerator()

RETURN Enumerator with the following functions.

Current	Returns the current envelope.
MoveNe	Moves the position to the next envelope to process.

xt()

SYNTAX	MoveNext()
RETURN	TRUE = Successful FALSE = The end of the collection is passed.

EXAMPLE	envEnumerator = OdinPrint.GetEnvelopeEnumerator()
	nextElement = envEnumerator.MoveNext()
	while (nextElement)
	envelope = envEnumerator.Current;
	…
	nextElement = envEnumerator.MoveNext()

Reset	Resets the enumerator to the beginning of the first element.

GetJobEnumerator

Returns a job enumerator. Use the GetValue() function to retrieve job data.

GetJobEnumerator

SYNTAX

GetJobEnumerator()

GetJobEnumerator(envelope)

ARGUMENTS

envelope	The envelope whose jobs you want to access.

RETURN

Enumerator with the following functions.

Current	Returns the current job.

MoveNext()	SYNTAX MoveNext()
	RETURN TRUE = successful
	FALSE = The end of the collection is passed.
	EXAMPLE jobEnumerator = OdinPrint.GetJobEnumerator()
	nextElement = envEnumerator.MoveNext()
	while (nextElement)
	;
	job = jobEnumerator.Current;
	;
	...
	nextElement = jobEnumerator.MoveNext()

Reset	Resets the enumerator to the beginning of the first element.

GetNextStack

Determines the next stack to convert according to the corresponding SQL, which is stored in the StackSelect property, and returns its ID.

GetNextStack	
SYNTAX	GetNextStack()
RETURN	ID of the stack found, or an empty string if none found.

SetStackError

Sets the status of a stack to error.

SetPdfSigner

SYNTAX	SetStackError(errorMessage)
ARGUMENTS	errorMessage Error message

Start

Loads and initializes the given stack and executes a script assigned to the OnStackCoverPage event.

The stack must be of the corresponding stack type to that required of the component. The status must be set to Wait.

Start	
SYNTAX	Start(stackId)
ARGUMENTS	stackId Stack ID

After successful loading, the stack is locked and the statuses set accordingly.

ODS_LockBy:	Name of the current user
ODS_State:	1 (OdinStackState.InWork)
ODS_PrintState:	1 (OdinPrintState.InWork)
ODS_Print_Start:	Current date / time
ODS_MessageText:	Empty string

StreamDocument

Streams the document of the given job.

StreamDocument

SYNTAX	StreamDocument(job)
ARGUMENTS	job Job object

Sample Script OdinStream

```
;
; get component for streaming
OdinStream = ModusProcess.GetComponent("OdinStream")
;
; determine the next stack for streaming
stackId = OdinStream.GetNextStack()
;
; was a stack found ?
if (stackID <> "")
;
  ; yes, now initialize stack and stream.
  ; call the StackCoverPage event.
  OdinStream.Start(stackId)
    ; get an envelope enumerator.
    envEnumerator = OdinStream.GetEnvelopeEnumerator()
    ;
    ; run through all envelopes and stream jobs
    nextEnv = envEnumerator.MoveNext()
    while (nextEnv)
      ;
      ; get current envelope from enumerator
      envelope = envEnumerator.Current
      ; as a test get envelope ID
```

```
  envelopId = envelope.GetValue("ODE_ID")
  ;
  ; all jobs in the envelope could be streamed with the function StreamDocuments
  ;OdinStream.StreamDocuments(envelope)
  ;
  ; get an job enumerator for the envelope.
  jobEnumerator = OdinStream.GetJobEnumerator(envelope)
  ; run through all envelope jobs and stream
  nextJob = jobEnumerator.MoveNext()
  while (nextJob)
  ;
    ; get current job from enumerator
    job = jobEnumerator.Current
    ; as a test get job ID
    jobId = job.GetValue("ODJ_ID")
    ;
    ; and now archive, call the OnArchive script
    OdinStream.StreamDocument(job)
    ;
    ; process next Job
    nextJob = jobEnumerator.MoveNext()
    ;
  end-while
  ;
  ; process next envelope
  nextEnv = envEnumerator.MoveNext()
end-while
;
; Streaming finished, call EndStack
OdinStream.End()
```

```
end-if
```

OdinStreamPrint

About OdinStreamPrint

You must have an Odin license installed to use the OdinStreamPrint component.

The OdinStreamPrint component enables you to output an Odin print stream to a printer driver.

Prerequisite You must define the trays used in the stream for the corresponding outsourcing printer.

Notes

- If no printer is assigned to the component, the component loads the printer defined in the stream file.
- The component uses the tray names in the outsourcing printer to identify the real physical tray IDs for printing.
- The component takes the settings for duplex and page orientation from the stream file.
- The outsourcing printer events execute alongside the OnBeforeStack / OnAfterStack events.

For more information, refer to Properties of the context OdinStreamPrint.

Properties of OdinStreamPrint

Property	Meaning
DB Alias	Odin database on which the component executes its actions.
NextStackType	Specifies the next stack type set after successful execution. The following table displays all possible stack types. Any additionally required stack types must be defined in the file **ModusSuite.Common.PropertyEditors.dll.config**. **Default stack types** Stack type | Name | Description 0 | Print | Stack is ready to be printed to an online printer or email printer (online print).

10	PortoOptimization	Stack is ready for IT franking. This function is currently not supported.
20	Streaming	Stack is ready for streaming.
30	Convert	Stack is ready to be converted to the appropriate output format.
40	StreamPrint	Stack (stream) is ready for batch printing to an outsourcing printer.
50	Archive	Stack is ready for archiving.
99	Finish	Stack processing is finished.

Printer	Outsourcing printer for printing the print stream.
Stack Select	SQL statement to identify the next stack to be processed. Either specify the select statement directly or refer to a lookup table (for example, $OdinSQL.PrintStackSelect$).

OdinStreamPrint Events

Event	Meaning
OnAfterStack	The specified script is executed after the stack processing when calling the **End** function. The event corresponds to the **AfterStack** event of a printer in Modus 4. **Available context:** Context OdinCustomProcess
OnBeforeStack	The specified script is executed when calling the **Start()**function, before executing the stack processing. The event corresponds to the **BeforeStack** event of a printer in Modus 4. **Available context:** Context OdinCustomProcess

OdinStreamPrint Functions

DefinePrinterByName

Defines a printer by its name.

DefinePrinterByName		
SYNTAX	DefinePrinterByName(printerName)	
ARGUMENTS	printerName	Printer name.

DefinePrinterByOId

Defines a printer by its object ID.

DefinePrinterByOId		
SYNTAX	DefinePrinterByOId(printerOID)	
ARGUMENTS	printerOID	Object ID of the printer.

End

Ends component processing, sets the stack to the next stack type and, if relevant, executes a script assigned to the OnAfterStack event.

End		
SYNTAX	End() End(errorMessage)	
ARGUMENTS	errorMessage	Error message.

The following values are set.

ODS_Print_End	Current date and time.
ODS_Delete_At	Current date, time, and stack hold time.

If processing ends correctly, the following values are set.

ODS_PrintState	2 (OdinPrintState.OK)
ODS_State	NextStackState of the component.
ODS_StackType	NextStackState of the component.

On error, the following values are set.

ODS_PrintState	3 (OdinPrintState.Error)
ODS_State	3 (OdinStackState.Locked)
ODS_MessageText	Error message.

GetCurrentStackRow

Show All

After calling Start() and before calling End() the function returns an object whose interface provides read access to the current stack dataset.

GetCurrentStackRow

SYNTAX

```
GetCurrentStackRow()
```

RETURN

Object interface that provides read access to the current stack dataset.

The following functions are available.

GetValue

Returns the value of the given column.

GetValue	
SYNTAX	GetValue(columnName)
ARGUMENTS	columnName Column name
RETURN	The column value. If the data type of the database column is DATE, Datetime or any valid date type, the date read generally returns in the format MM/DD/YYYY hh:mm:ss.
EXAMPLE	GetObject("DateVar","MLDate") StringDateVar = CurrentJobRecord.GetValue("ODJ_create_at") ;Convert date to the required format with the help of the MLDate object IntDateVar = DateVar.DateStringToInt64("MM/dd/yyyy hh:mm:ss", StringDateVar) v_LETTERDATE= DateVar.Int64ToDateString("MM.dd.yyyy hh:mm:ss ", IntDateVar)

GetFieldNameEnumerator

Returns an enumerator.

GetFieldNameEnumerator	
SYNTAX	GetFieldNameEnumerator()
RETU	Enumerator with the following functions.

RN

Current	Returns the current object.
MoveNext()	Moves the position to the next object to process. SYNTAX: MoveNext() RETURN: TRUE = Successful FALSE = The end of the collection is passed.
Reset	Resets the enumerator back to the beginning of the first element.

GetDocumentEnumerator

Returns a document enumerator.

GetDocumentEnumerator

SYNTAX GetDocumentEnumerator()

RETURN

Enumerator with the following functions.

Current	Returns the current document.
MoveNext	Moves the position to the next document to process.

()

SYNTAX	MoveNext()
RETURN	TRUE = Successful FALSE = The end of the collection is passed.

EXAMPLE	docEnumerator = OdinPrint.GetDocumentEnumerator()
	nextElement = docEnumerator.MoveNext()
	while (nextElement)
	envelope = docEnumerator.Current;
	...
	nextElement = docEnumerator.MoveNext()

Re set — Resets the enumerator to the beginning of the first element.

GetNextStack

Determines the next stack to convert according to the corresponding SQL, which is stored in the StackSelect property, and returns its ID.

GetNextStack	
SYNTAX	GetNextStack()
RETURN	ID of the stack found, or an empty string if none found.

PrintDocument

Prints the specified document.

PrintDocument

SYNTAX	PrintDocument(document)
ARGUMENTS	document — The document to print.

SetStackError

Sets the status of a stack to error.

SetPdfSigner

SYNTAX	SetStackError(errorMessage)
ARGUMENTS	errorMessage — Error message

Start

Loads and initializes the given stack and executes a script assigned to the OnBeforeStack event.

The stack must be of the corresponding stack type to that required of the component. The status must be set to Wait.

Start

SYNTAX	Start(stackId)
ARGUMENTS	stackId — Stack ID

After successful loading, the stack is locked and the statuses set accordingly.

ODS_LockBy:	Name of the current user
ODS_State:	1 (OdinStackState.InWork)
ODS_PrintState:	1 (OdinPrintState.InWork)

ODS_Print_Start:	Current date / time
ODS_MessageText:	Empty string

Sample Script OdinStreamPrint

```
; initialize StreamPrint components
OdinStreamPrint = ModusProcess.GetComponent("OdinStreamPrint")
; determine next stack
stackID = OdinStreamPrint.GetNextStack()
;
; was a stack found ?
if (stackID <> "")
;
  ; Yes, now initialize stack print
  OdinStreamPrint.Start(stackID)
  ;
    ; get a DocumentEnumerator and print each document in the stream
    DocumentEnumerator = OdinStreamPrint.GetDocumentEnumerator()
    ;
    nextDocument = DocumentEnumerator.MoveNext()
    ;
    while(nextDocument = true)
      ; print next document in the stream
      document = DocumentEnumerator.Current
      OdinStreamPrint.PrintDocument(document)
      ;
      ; navigate the stream to the next document
      nextDocument = DocumentEnumerator.MoveNext()
    end-while
    ; end stream print
```

```
    OdinStreamPrint.End()
end-if
```

Transfer

About Transfer

You must have a Versioning license installed to use the transfer component.

This component starts scheduled Transfer Jobs.

Properties of Transfer

Parameter	Description
RollbackOnFailure	TRUE = The transfer is retracted in the event of an error. FALSE = The transfer is not retracted in the event of an error.
RepositoryDBAlias	Database on which the component searches for transfer jobs.

Transfer Functions

ExecuteTransfer

Executes a scheduled transfer.

ExecuteSynchron

SYNTAX ExecuteTransfer(scheduleTransferDbId)

ARGUMENTS

scheduleTransferDbId	ID of the scheduled transfer job.

RETURN

>0 (number of errors)	Errors occurred while importing.
-1	Parameter scheduledTransferId is null or empty.
-2	Parameter scheduledTransferId cannot be converted to long.

-3	No scheduled transfer found for the given scheduledTransferId.
-4	More than one scheduled transfer found for the given scheduledTransferId.
-5	Exception thrown while importing the transfer.
-6	Exception thrown while retracting the transfer
-7	The status of the transfer could not be set to InProgress.
-8	The status of the transfer could not be set to Finished.
-9	The scheduled transfer cannot be executed any more.
-100	Exception thrown while executing the transfer.

GetNextTransfer

Checks whether a scheduled transfer job to be executed exists.

GetNextTransfer	
SYNTAX	GetNextTransfer()
RETURN	ID of the scheduled transfer job or an empty string when no scheduled transfer job exists.

XMLProcessingManager

About XMLProcessingManager

Use this component to handle XML files by a process.

The files to be processed can contain one or more print jobs.

Properties of the XmlProcessingManager

Property	Description

FileName	Name of the XML file currently open in the form of processId.wrk.
JobElementPath	Path to the datasets within the multiple XML file, for example, /JOBS/JOB.
LastXmlContent	Last data XML
LastOuterXmlContent	When processing a single XML file, the content is the same as LastXmlContent. When processing a multiple XML file, the property also contains the job node. **Example** <JOBS> <JOB> <Partner ...> … </Partner> </JOB> </JOBS) LastXmlContent: <Partner …….> … </Partner> LastOuterXmlContent: <JOB> <Partner ...> ... </Partner> </JOB>
OriginalFileName	Original name of the XML file.
SearchDirectory	Path where the XML files to be processed are stored, for example, D:\Input\Multi*.xml.

XMLProcessingManager Functions

Cancel

Orders the XmlParser to cancel on the next call of NextJob().

Cancel	
SYNTAX	Cancel()
EXAMPLE	processor.Cancel()

FileExists

Searches the directory specified as SearchDirectory for files containing data for batch processing.

If a file exists, it is opened and the file extension changes to .wrk.

FileExists		
SYNTAX	FileExists(processID)	
ARGUMENTS	processId	Process ID
RETURN	TRUE = File found. FALSE = File not found.	
EXAMPLE	UniqueId = UniqueGuid.ToString() ExecuteTimer = processor.FileExists(UniqueId)	

FileExistsEx

Searches the directory specified as SearchDirectory for files containing data for batch processing.

If a file exists, it is opened and the file extension changes to .wrk.

FileExistsEx

SYNTAX	FileExistsEx(processID)
ARGUMENTS	processId Process ID
RETURN	If a file exists, the whole content of the MultiXml file returns as an XML string. If no file exists, an empty string returns.

GetJobAttribute

Allows access to the attributes of the JOB element in the XML batch file.

GetJobAttribute	
SYNTAX	GetJobAttribute(attributeName)
ARGUMENTS	attributeName Attribute name.
RETURN	Value of the attribute or an empty string, if the attribute does not exist.
EXAMPLE	System.DocumentCollection = processor.GetJobAttribute("name")

NextJob

Reads the data XML of the next job from the XML batch file. The data XML is then available in the attribute **LastXmlContent**.

Syntax

NextJob()

Return value

Value	Definition
True	Next job was read.
False	No next job found.

Example

SYSTEM.DoWork = processor.NextJob()

StartProcessing

Initializes the processing of an XML batch file.

StartProcessing	
SYNTAX	StartProcessing()
EXAMPLE	processor = ModusProcess.GetComponent("XmlProcessingManager") processor.StartProcessing()

Terminate

Ends the processing of an XML batch file.

Terminate	
SYNTAX	Terminate()
EXAMPLE	processor = ModusProcess.GetComponent("XmlProcessingManager") processor.StartProcessing() ... processor.Terminate()

XwsDataSelectionManager

About XwsDataSelectionManager

You must have an Xdata license installed to use the XwsDataSelectionManager component.

This component provides data retrieval through Xdata web services and creates an XML file used for document generation.

Properties of the XwsDataSelectionManager

Parameter	Description
SelectionReference	Selection the component executes.

WebServiceUrl	URL of the Xdata web services.
LastError	Use this property to retrieve information if an error occurs during data collection.
NavigatorXml	If the function GetDataExtended is called with the parameter **retrieveNavigateXml = True**, you can access the navigation XML here.
XmlData	Read the XML of the last data collection as a string.

XwsDataSelectionManagers Functions

GetData

Executes data retrieval.

GetData

SYNTAX

GetData(raiseError)

GetData(selectionParameters, raiseError)

ARGUMENTS

selectionParameters	Selection parameter in XML format **Example** selectionParameters = "<parameters><parameter name=""Partnernumber"" value=""0815"" /></parameters>" If this parameter is not specified, the selection parameters must be stored in the system pool. **Example** system.Partnernumber = "0815"
raiseError	TRUE = An error is thrown if a reading script tries to read a data object with SelVariable that does not exist. FALSE = No error is thrown if a reading script tries to read a data object with SelVariable that does not exist.

RETURN

<> 0 if an error occurred.

GetDataExtended

Executes data retrieval.

GetDataExtended		
SYNTAX	GetDataExtended(raiseError, retrieveNavigateXml) GetDataExtended(selectionParameters, raiseError, retrieveNavigateXml)	
ARGUMENTS	selectionParameters	Selection Parameters **Example** selectionParameters = "<parameters><parameter name=""Partnernumber"" value=""0815"" /></parameters>" If this parameter is not specified, the selection parameters must be stored in the system pool. **Example** system.Partnernumber = "0815"
	raiseError	TRUE = An error is thrown if a reading script tries to read a data object with SelVariable that does not exist. FALSE = No error is thrown if a reading script tries to read a data object with SelVariable that does not exist.
	retrieveNavigateXml	TRUE = Write the navigation XML in the NavigatorXml property. FALSE = Do not write navigation XML in the NavigatorXml property.
RETURN	<> 0 if an error occurred.	

Ping

Checks the reachability of the web service defined in the WebServiceUrl property.

Ping

SYNTAX	Ping()
RETURN	"Xdata Runtime Webservice (*name of the user running the web service*)".

Sample Script XwsDataSelectionManager

```
xws = ModusProcess.GetComponent("XWSDataSelectionManager")
;
pingResult = xws.Ping()
;
; The parameter raiseInSelVar (set to false in all examples) throws an exception if true,
; that means if the selection tries to read a non-existent variable.
;
;-------------------------------------------------------------
;Alternative with Selection Parameter in an XML string
;-------------------------------------------------------------
selectionParameters = "<parameters><parameter name=""Partnernumber"" value=""0815""
/></parameters>"
;no NavigationsXml
rescode = xws.GetData(selectionParameters, false)
data = xws.XmlData
error = xws.LastError
;
;-------------------------------------
;with NavigationsXML (here 3. Parameter)
;-------------------------------------
rescode = xws.GetDataExtended(selectionParameters, false, true)
data = xws.XmlData
navigData = xws.NavigatorXml
error = xws.LastError
```

```
;
;-----------------------------------------------------
;Alternative with Selection Parameter stored in system pool
;no NavigationsXml
;-----------------------------------------------------
system.Partnernumber = "0815"
rescode = xws.GetData(false)
data = xws.XmlData
error = xws.LastError
;
;-------------------------------
;with NavigationsXML (2. Parameter)
;-------------------------------
rescode = xws.GetDataExtended(false, true)
data = xws.XmlData
navigData = xws.NavigatorXml
error = xws.LastError
;
```

Query Definitions

About Query Definitions

Query definitions enable you to compile analyses of database tables that you can run from the Query Assistant.

Creating a Query Definition

To create a query definition, complete the following steps.

1. In the bottom pane of the **Navigator**, right-click and click **New** > **Query Definition**.
2. In the **Create New Query Definition** dialog box, type a name for the query definition and click **OK**.
3. In the right pane of the **Editor** window, in the **Database Alias** list, select a database alias.
   The list of the tables contained in the database alias displays.
4. To display the list of table fields, click the plus sign to the left of the table from which you want to read the data.

5. From the right pane of the **Editor** window, drag the preferred table fields to the bottom-left pane of the **Editor** window named **Result Columns**.
   Note If you need all fields in a table, drag the table to the **Result Columns** pane.
6. Optional. In the **Object Inspector** pane, edit the properties of the query definition.
7. Optional. In the **Object Inspector** pane, edit the properties of the result columns.

Adding a condition

A condition in a query definition corresponds to the WHERE clause of the SQL statement. Each query definition can contain an unlimited number of conditions.

To add a condition to the query definition, complete the following steps.

1. On the **Query DefinitionEditor Tools** tab, click the **Add New Condition** button.
2. In the **Add New Condition** dialog box, type a name for the condition and click **OK**.
3. In the **Object Inspector** pane, edit the **properties of the condition**.
4. On the **Query DefinitionEditor Tools** tab click the **Compile** button.
5. Observe success or failure messages in the **Output** p.

Adding a Variable to a Condition

Each condition can contain an unlimited number of variables, all of which you can use in the WHERE clause and which you can specify as parameters in the query assistant.

To add variables to a condition, complete the following steps.

1. In the **Editor** window click the condition.
2. On the **Query DefinitionEditor Tools** tab, click the **Add New Variable** button.
3. In the **Add New Variable** dialog box, type a name for the variable and click **OK**.
4. In the **Object Inspector** pane, edit the **properties of the variables**.
5. On the **Query DefinitionEditor Tools** tab click the **Compile** button.
6. Observe success or failure messages in the **Output** pane.

Adding a column

To add an additional column to the query definition, complete the following steps.

1. On the **Query DefinitionEditor Tools** tab, click the **Add New Column** button.

2. In the **Add New Column** dialog box, type a name for the column and click **OK**.
3. In the **Object Inspector** pane, in the **DB Column** property, type a function. For example, `Count (*) as`*`Columnname`*
4. Optional. In the **Object Inspector** pane, modify the **properties of the column**.
5. On the **Query DefinitionEditor Tools** tab, click the **Compile** button.
6. Observe success or failure messages in the **Output** pane.

Properties of a Query Definition

Property	Description
Description	Free text description.
DB Alias	DB alias on which the query runs.
DB Table	Corresponds to the FROM statement in the select.
Detail Query	Define a query definition to give a more detailed result.
Category	Category of the query (free text).
Maximum Lines	Number of rows in the query result.

Properties of Result Columns

Property name	Description
Description	Free text description.
DB Column	Name of the database field.
Visible	TRUE = The result column displays in the result. FALSE = The result column does not display in the result.
Column Size	Column width in the query result. Specify -1 to determine the column width automatically depending on contents and column headings.
Title	Column title in the query result.

Properties of a Condition

Property	Description
Description	Free text description.
GroupBy	Database column by which the results are grouped.
OrderBy	Database column by which the results are sorted.
Title	Parameter title.
Where	Corresponds to the WHERE statement in the select. If the condition contains variables, you can use them in one of the following ways. • PR_TYPE = $VariableName$ • PR_TYPE = :VariableName

Properties of a Variable

Property	Description
DefaultValue	Default value for parameter input. You can also use values in the form of **[ModusUser.AttributeName]**.
Prompt	TRUE = The variable displays in the query dialog box and the user can edit the variable. FALSE = The variable does not display in the query dialog box.
Title	Title of the variable as shown in the query assistant.
TypeName	Type of variable (free text)
ValueHelp	Name of the value help assigned to the variable.

OnBase Interfaces

OnBase DIP File Configurations

About OnBase DIP File Configurations

Use this Content Composer interface to add document files to the OnBase archive using a file import interface.

To work with **OnBase DIP File Configuration** objects, the user must possess the right **OnBase DIP File Configuration (75)**.

An **OnBase DIP File Configuration** allows you to specify the keyword values you want to transfer to the OnBase archive.

Possible sources for the keyword values are the XML data used when creating a document or bundle as well as MonaLisa pool variables.

The **OdinOnBaseArchive** component uses this information to create the OnBase XML DIP file.

Creating an OnBase DIP File Configuration

To create an OnBase DIP File Configuration, complete the following steps.

1. In the bottom pane of the **Navigator**, right-click and click **New** > **OnBase DIP File Configuration**.
2. In the **New OnBase DIP File Configuration** dialog box, type a name and click **OK**.
3. To assign a selection, complete the following substeps.
    1. In the **Object Inspector** pane, in the **Data Model** section, click the **Selection** property and then click the ellipsis button next to the property.
    2. In the **Select an Object** dialog box, select the required selection and then click **Select**.
    3. Optional. To assign another selection, repeat the previous steps.
    4. Optional. To remove the selection, in the **Object Inspector**, right-click the property and then click **Reset**.
    5. From the **Data Model** pane, drag the required nodes or attributes to the **XML Data Links** tab.
    6. Optional. To remove a data link, select the respective entry and click the **Delete** button on the **Editor Tools** tab .
4. To create a new pool variable, complete the following substeps.
    1. Click the **Pool Variables** tab.
    2. On the **Editor Tools** tab click **Add**.
    3. In the **Pool Name** column, type the name of the pool.

4. In the **Pool Variable Name** column, type the variable name.
   Note: To avoid problems with case sensitivity, **Content Composer Studio** converts the pool and variable names to lowercase.
5. Optional. In the **Object Inspector**, provide a description in the **Description** property.
6. Click **Save**.

OnBase Document API Configurations

About OnBase Document API Configurations

To work with **OnBase Document API Configuration** objects, the user must possess the right **OnBase Document API Configuration (76)**.

An **OnBase Document API Configuration**allows you to specify the OnBase keywords and setup the data source for the keyword values you want to transfer to the OnBase Document REST API archive interface.

Overview

To archive a document created by Content Composer in OnBase, do the following:

1. In OnBase, decide which **OnBase Document Type** you want to use for archiving the Content Composer document.
2. In Content Composer Studio, create an **OnBase Document API Configuration** object for the selected **OnBase Document Type**.
3. In Content Composer Studio, configure the document template for archiving.
4. Modify the Odin processing and use the functions of the Monalisa object **MLOnBaseDocumentApi** to archive documents in OnBase.

Creating an OnBase Document API Configuration

To create an OnBase Document API Configuration, complete the following steps.

1. In the bottom pane of the **Navigator**, right-click and click **New** > **OnBase Document API Configuration**.
2. In the **New OnBase Document API Configuration** dialog box, type a name and then click **OK**.
3. In the **Object Inspector** pane, click the **Document Type** property and then click the ellipsis button next to the property.
4. In the **Specify the OnBase Document Type** dialog box, do one of the following:
   - Click and then select the required document type from the **Select Document Type** list.
   - Enter **Document Type Name** and **Document Type ID** in the corresponding fields.
     Note: Ensure to enter the correct values, since Content Composer does not verify the entered values.
5. Click **OK**.

6. To select the OnBase keywords that Content Composer passes to OnBase along with the document, on the **Editor Tools** tab, click **Add**.
7. In the **Add OnBase Keyword** dialog box, click and then select the required keywords form the **Select Keywords** list.
8. Click **OK**.
9. Content Composer can retrieve the keyword value either from an Xdata selection or from an archive attribute. To read the keyword value from an Xdata selection, complete the following substeps:
    1. In the **Object Inspector** pane, in the **Data Model** section, click the **Selection** property and then click the ellipsis button next to the property.
    2. In the **Select an Object** dialog box, select the required selection and then click **Select**.
    3. In the upper **Data Model** pane, select the required node.
    4. From the lower **Data Model** pane, drag the required attribute to the corresponding keyword row.
10. To read the keyword value from an archive attribute, complete the following substeps:
    1. Select the keyword row and enter the name of the archive attribute in the column **Archive Attribute Name**.
    2. In a MonaLisa script, call the function SetArchiveAttribute and use the name of the archive attribute for the parameter name.
11. Do one of the following:
    - If the keyword value is a numeric value, in the column **Number format**, select the corresponding value from the drop-down list.
    - If the keyword value is a date or date time value, in the column **Date format**, enter the corresponding format string, for example **MM/dd/yyyy**.
      For more information, see Format Specifiers.
    - If the keyword value is a string, no further actions are required.
12. Click **Save** .

Configuring Archiving Settings for a Document Template

To configure the archiving settings for a document template, complete the following steps:

1. From the **Navigator**, open the bundle.
2. In the bundle editor, select the respective document template.
3. In the **Object Inspector**, set the **Archive** property to **True**.

4. In the **Object Inspector** pane, in the **OnBase Archive Configuration** section, click the **Document Type** property and then click the ellipsis button next to the property.
5. In the **Specify the OnBase Document Type** dialog box, click and then select the required document type from the **Select Document Type** list.
6. In the **OnBase Document API Configuration** field, select the required configuration.
   Note: You can only select an **OnBase Document API Configuration** that matches the selected **Document Type**.
7. To archive this document additionally as a single file, in the **Object Inspector**, set the **Transfer single file to archive** property to **True**.
   Note: This setting is only taken into account if the bundle property **Combine documents in a direct print stack** is set to **True** as well.
8. Click **Save**.

Configuring Archiving Settings for a Bundle

To combine all direct print documents of a bundle into one PDF file and archive this combined file to OnBase, complete the following steps:

1. In the bundle editor, select the bundle.
2. In the **Object Inspector**, in the **OnBase Archive Configuration** section, select the **Document Type** property and then click the ellipsis button next to the property.
3. In the **Specify the OnBase Document Type** dialog box, click and then select the required document type from the **Select Document Type** list.
4. In the **OnBase Document API Configuration** field, select the required configuration.
5. Set the **Combine documents in a direct print stack** property to **True**.
6. Click **Save**.

Odin Processing

Sample Script: Archive Documents in the Component Odin Rendition

For maximum flexibility, the Monalisa object **OnBase Document API** provides functions to archive documents using the OnBase Document REST API.

For more information, see About MLOnBaseDocumentApi.

Assign the following script to the event **OnProcessEnd** of the component **Odin Rendition**.

```
;Sample script for script event "OnProcessEnd" of the component "Odin Rendition"

GetObject("OnBaseDocApi","MLOnBaseDocumentApi")
```

```
try
   ;configure OnBaseDocApi
   OnBaseDocApi.SetOnBaseDocumentApiUrl("http://in01-devpdc117.pvi.com/ApiServer/onbase/core/")
   OnBaseDocApi.SetLicenseFilePath("%Composerdir%\cpsdk.lic")
   OnBaseDocApi.SetProfilePath("%Composerdir%\profiles\")
   OnBaseDocApi.SetRequestTimeout(200)

   procId = CurrentProcessId
   ODP_OB_APIFILE = ProcessRow.GetValue("ODP_OB_APIFILE")

   ;Combine all jobs if process record column "ODP_OB_APIFILE" contains an OnBase keyword file
   doCombineJobs = false
   if (StrLen(ODP_OB_APIFILE) > 0)
      doCombineJobs = true
   end-if

   if (doCombineJobs)
      #region combine all jobs into one PDF

      ;read values from process record
      ODP_OB_DOCTYPE_NAME = ProcessRow.GetValue("ODP_OB_DOCTYPE_NAME")
      OnBaseDocApi.StartArchivePdfFile(ODP_OB_DOCTYPE_NAME)

      OnBaseDocApi.ClearKeywords()
      OnBaseDocApi.LoadKeywordsFromFile(ODP_OB_APIFILE)
      counter = 0
      combinedJobIds = ""
      jobsAdded = false
      while (counter < JobCount)
```

```
jobRow = GetJobRow(counter)
ODJ_ID = jobRow.GetValue("ODJ_ID")

;archive job?
doArchiveJob = false
ODJ_TO_ARCHIVE = StrUpper(jobRow.GetValue("ODJ_TO_ARCHIVE"))
if (ODJ_TO_ARCHIVE = "Y")
   doArchiveJob = true
end-if

;is job already archived?
JobIsArchived = false
ODJ_ARCHIVED = StrUpper(jobRow.GetValue("ODJ_ARCHIVED"))
if (ODJ_ARCHIVED = "Y")
   JobIsArchived = true
end-if

if (doArchiveJob and (JobIsArchived = false))
   ;archive job

   ;read values from from job record
   ODJ_RENDITION = jobRow.GetValue("ODJ_RENDITION")
   OnBaseDocApi.AddFile(ODJ_RENDITION)
   combinedJobIds = combinedJobIds + " " + ODJ_ID
   jobsAdded = true

   ;check for pass job as Single?
   JobPassAsSingle = false
   ODJ_OB_DP_PASS_SINGLE =  StrUpper(jobRow.GetValue("ODJ_OB_DP_PASS_SINGLE"))
   if (ODJ_OB_DP_PASS_SINGLE = "Y")
```

```
            JobPassAsSingle = true
        end-if

        ; only mark archived if not to pass as single additionaly
        if (JobPassAsSingle = false)
        ;mark job as archived
        jobRow.SetValue("ODJ_ARCHIVED", "Y")
        end-if
    end-if

      counter = counter + 1
    end-while ;loop over jobs

    if (jobsAdded)
      docId = OnBaseDocApi.FinishArchivePdfFile()
      Protocol("Process Id {0}: successfully combined and archived the jobs {1}. OnBase document id: {2}",
5, procId, combinedJobIds, docId)
    else
      Protocol("Process Id {0}: no jobs to archive found.", 5, procId)
    end-if
    #endregion
  end-if

  #region archive single jobs
  counter = 0
  while (counter < JobCount)

    jobRow = GetJobRow(counter)
    ODJ_ID = jobRow.GetValue("ODJ_ID")
```

```
;archive job?
doArchiveJob = false
ODJ_TO_ARCHIVE = StrUpper(jobRow.GetValue("ODJ_TO_ARCHIVE"))
if (ODJ_TO_ARCHIVE = "Y")
   doArchiveJob = true
end-if

;is job already archived?
JobIsArchived = false
ODJ_ARCHIVED = StrUpper(jobRow.GetValue("ODJ_ARCHIVED"))
if (ODJ_ARCHIVED = "Y")
   JobIsArchived = true
end-if

if (doArchiveJob and (JobIsArchived = false))
   ;archive job

   ;read values from from job record
   ODJ_OB_APIFILE = jobRow.GetValue("ODJ_OB_APIFILE")
   ODJ_OB_DOCTYPE_NAME = jobRow.GetValue("ODJ_OB_DOCTYPE_NAME")
   ODJ_RENDITION = jobRow.GetValue("ODJ_RENDITION")

   OnBaseDocApi.ClearKeywords()
   OnBaseDocApi.LoadKeywordsFromFile(ODJ_OB_APIFILE)
   docId = OnBaseDocApi.ArchivePdfFile(ODJ_RENDITION, ODJ_OB_DOCTYPE_NAME)
   Protocol("Process Id {0}: successfully archived the job with id {1}. OnBase document id: {2}", 5, procId,
ODJ_ID, docId)

   ;mark job as archived
   jobRow.SetValue("ODJ_ARCHIVED", "Y")
```

```
    end-if

    counter = counter + 1
    end-while ;loop over jobs
  #endregion

finally
  FreeObject("OnBaseDocApi")
end-try
```

Maintaining OnBase Object IDs

Content Composer stores the name and ID of each **OnBase Document Type** and **Keyword**, but the **OnBase Document API** accepts only IDs as input.

If an installation consists of different OnBase server instances for example for development, test, and production, the same set of **OnBase Document Types** and **Keywords** exist on these different servers with the same name, but different IDs.

To align the IDs in the instances, Content Composer provides a tool that connects to an OnBase server instance and updates the IDs for the **Document Types** and **Keywords** stored in the **OnBase Document API Configuration** objects.

When switching the OnBase server for a Content Composer installation or when transferring Content Composer objects of type **OnBase Document API Configuration** to another environment, for example from test to production, complete the following step:

- On the Content Composer server, from the **%Composerdir%** directory, as an administrator, execute the following command:

  Composer.tool.exe -[*sysoid:mytest]* -Cmd:UpdateOnBaseApiIds

Note: To display the help, execute the **Composer.Tool.exe** without specifying the parameters.

Xdata

About Xdata

Xdata is the data retrieval module of Content Composer and performs all data retrieval activities.

Xdata offers the possibility to extract data from various databases, to read data from SAP, and to make the data available to other components as XML.

Xdata runtime provides a framework to read the metadata of configured data bases and SAP systems. This metadata is used to create selections and data objects.

An integrated compiler converts these data selection objects to a .NET assembly. This assembly contains the complete process flow of a selection, which means the whole sequence of individual read data operations.

The following database access providers are supported.

- ODP
- OLEDB
- ORAMS
- MSSQL

Limitations on Calling BAPI Functions

Content Composer allows to call certain SAP BAPI functions.

Regard the following limitations for an SAP BAPI to be called.

Input Parameters (Imports)

Currently, Content Composer supports the following ABAP data types for Import parameters.

- Integer (4-bytes)
- Packed decimal
- Floating point
- Character numeric
- Character
- Date
- Time
- RFC STRING Variable-length string
- XSTRING Variable-length raw byte array.

Return Parameters (Exports)

Currently Content Composer exclusively calls BAPIs with export parameters complying with one of these conditions:

- Parameter is an SAP structure (single value return).
- Parameter is an SAP table (return of lists).

Data Objects

About Data Objects

Data objects are logical views of physical data and contain 1-n attributes, data object variables.

Data objects can stand in relation to other data objects.

Creating a Data Object

You must have an Xdata license installed to create data objects.

To create a data object, complete the following steps.

1. In the bottom pane of the **Navigator**, right-click and click **New** > **Data Object**.
2. In the **Data Object Assistant** window, select the **Create data objects** option and click **Next**.
3. In the **Data Object Assistant** window, complete the following substeps.
    1. From the **Alias Connection** drop-down menu, select an alias.
    As a result, a list of all tables associated with this database (DB) alias displays.
    Note For SAP connections the list displays all available BAPIs. For more information, refer to Limitations on calling SAP BAPIs.
    2. Select the tables from which you want to create new data objects.
    3. Optional. To assign a new name to a target data object, right-click the table and click **Rename**.
    4. Optional. To exclude data fields the data object should not contain, click the plus sign to the left of a table and clear the appropriate check boxes.
    5. Click the **Next** button.
4. In the **Data Object Assistant** window, click the **Create Data Objects** button.
5. Optional. To create additional data objects, click the **Back** button.

Modifying a Data Object

To modify a data object, complete the following steps.

1. Open the data object.
2. Optional. To add a variable, complete the following substeps.
    1. On the **Xdata Editor Tools** tab, click the **New Variable** button.
    2. In the **Create Data Object Variable** dialog box, type a name for the variable and click **OK**.
    Note The data object variables display in the upper-right pane of the **Editor** window.
3. Optional. To rename a variable, complete the following substeps.
    1. In the upper-right pane of the **Editor** window, right-click the variable and click **Rename**.
    2. In the **Rename Data Object Variable** dialog box, type a new name for the variable and click **OK**.
4. Optional. To delete a variable, in the upper-right pane of the **Editor** window, right-click the variable

and click **Delete Variable**.

5. Create the preferred reading scripts.
6. In the **Object Inspector** pane, click the **Active Reading Script** property.
7. Click the down arrow next to the property and select the preferred reading script.
8. Optional. From the bottom pane of the **Navigator**, drag child data objects to the **Editor** window.

Reading Scripts

About reading scripts

Data objects that read data from a database require a reading script that executes at runtime. This reading script executes the specified SQL query and applies the read data to the data object.

Note Data objects that read data from SAP do not use reading scripts.

Creating a Reading Script

To create a reading script, complete the following steps.

1. Open the data object.
2. In the **Editor** window, select the topmost data object.
   Note To create reading scripts for child data objects, open the respective child data object.
3. On the **Xdata Editor Tools** tab, click the **New Reading Script** button.
4. In the **Create New Reading Script** dialog box, type a name for the script and click **OK**.
   Note All available reading scripts display in the bottom-right pane of the **Editor** window.
5. In the bottom-right pane of the **Editor** window, double-click the newly created script.
   Note The reading script already contains two default statements and is fully functional.
   `db.WorkWithQuery("Q1")` ; Executes the specified SQL query ("Q1")
   `db.ApplyDataSet()` ; Applies the read data set to the data object
6. Optional. Modify the reading script. For more information, refer to Xdata Functions.
7. Modify the SQL query.
8. Optional. To create additional SQL queries, complete the following substeps.
   1. In the right pane of the **Editor** window, right-click and click **Create New SQL**.
   2. In the **Create New SQL** dialog box, type a name for the query and click **OK**.
   3. In the bottom pane of the **Editor** window, complete the `where` clause of the SQL statement.
      For more information, refer to Modify an SQL query.
   4. Optional. In the upper pane of the **Editor** window, replace the parameter **Q1** with the name

of the new SQL query.
Example
before
```
db.WorkWithQuery("Q1")
```

after
```
db.WorkWithQuery("Q2")
```

Modifying the SQL Query

When you were creating the reading script, an SQL query Q1 was generated that needs modification. This query contains the SQL statement used to read the data fields.

To modify the SQL query, complete the following steps.

1. In the **Editor** window of the reading script, in the right pane, click the **Q1** entry.
   As a result, the query statement displays in the bottom pane of the **Editor** window.
2. In the bottom pane of the **Editor** window, complete the `where` clause of the SQL statement.
   Example
   ```
   select * from BANKS
    where BNK_BIN=:ACC_BIN
   ```

Notes

- The `where` clause typically contains variables that populate with values at runtime.
- A variable is preceded by a colon.
- The variable content is replaced at runtime according to the following rules.
  - If the parent data object contains a field that matches the name of the variable, the field value is used.
  - If nothing is found, the **SYSTEM** pool is checked for a variable of this name and the corresponding value is used.

Inserting Child Data Objects

A data object can contain child data objects to which it maintains a master-detail relationship.

To insert a child data object in a data object, complete the following steps.

1. Open the data object.
2. From the bottom pane of the **Navigator**, drag one or more data objects to the **Editor** window.
3. In the Editor window, click the child data object.
4. In the **Object Inspector** pane, click the **Active Reading Script** property.
5. Click the down arrow next to the property and select the preferred reading script.
6. In the **Object Inspector** pane, click the **RelationType** property, then click the down arrow next to the property and select the preferred relationship.
   - A **One2One** relationship is one that exists for example between partner and title - each partner has only one title.
   - A **One2Many** relationship is one that exists for example between partner and address - each

partner may have more than one address.

6. To assign the key fields used to define the relationship between the two objects, complete the following substeps.
    1. Click the parent data object.
    2. From the upper-right pane of the **Editor** window, drag one or more data object variables you want to use as masterkey onto the child data object in the left pane of the **Editor** window.
       Result
       The **Masterkey Column(s)** property of the child object changes to the name of the data object variable(s).
    3. Click the child data object.
    4. From the upper-right pane of the **Editor** window, drag one or more data object variables you want to use as key onto the child data object in the left pane of the **Editor** window.
       Result
       The **Key Column(s)** property of the child object changes to the name of the data object variable(s).

Nesting depth of child objects

You can nest objects within each other to any depth. For better performance the editor only displays the next nesting level for each object.

Example

- The data **Partner** object has the **Accounts** child data object.
- The **Accounts** data object has the **Banks** child data object.
- In the data **Partner** object, the **Banks** child object is not visible.

Creating a Summary Node

A summary node is a special function of child data objects.

A summary node allows you to collate datasets, for example from child data objects that should be immediately available in the selection.

Example

The **Partner** data object contains the **Accounts** child object, which itself contains the **Banks** child object.

Use a summary node to have all **Banks** data read directly available in the selection.

To insert the **Banks** data object as a summary node in the **Partner** data object, complete the following steps.

1. Open the **Partner** data object.
2. From the **Navigator**, drag the **Banks** data object to the **Editor** window.
3. In the **Editor** window, click the **Banks** data object.

4. To define the **Banks** child object as a summary node, in the **Object Inspector** pane, in the **Masterkey** property , type *.
5. In the **Object Inspector** pane, in the **Key** property**,** type the table field by which the entries can be exactly identified, for example the field that contains the bank routing number.

Result

The selection reads all the accounts of **Partner** and for each account the corresponding bank data. The **Banks** summary node contains all banks read by the selection.

Note Summary nodes do not need a reading script.

Xdata Editor Tools for Data Objects

Icon	Name	Description
	Refresh	Refreshes the data object. You can only execute this function when you have saved your modifications.
	New Variable	Creates a new data object variable.
	New Reading Script	Creates a new reading script
	Open Data Object	Opens the selected data object.

Empty Data Objects

Creating an Empty Data Object

You must have an Xdata license installed to create empty data objects.

An empty data object is not directly associated to a data source. You can use an empty data object for example to execute SQL queries.

To create an empty data object, complete the following steps.

1. In the bottom pane of the **Navigator**, right-click and click **New** > **Data Object**.
2. In the **Data Object Assistant** window, select the **Create a new empty data object** option and click **Next**.

3. In the **Create New Data Object** dialog box, type a name for the data object and click **OK**.
4. In the **Data Object Assistant** window, click the **Create Data Object** button.

Example of Use

Modifying or Deleting Datasets with an Empty Data Object

To use an empty data object to modify or delete datasets, complete the following steps.

1. Create an empty data object.
2. Create a new reading script.
3. Modify the reading script as follows.
```
db.StartTransaction()
 db.ExecuteQuery("Q1")
 db.Commit()
```
4. Modify the SQL query to an `Update` or `Delete` statement.
Example
```
Update Partner set LastSaved = sysdate where par_id = :PAR_ID
```

Executing a Stored Procedure with an Empty Data Object

To execute a stored procedure in an empty data object, complete the following steps.

1. Create an empty data object
2. For each parameter of the stored procedure, add a variable.
3. For each variable, in the **Object Inspector** pane, in the **DataLink** property, type the name for the respective stored procedure parameter.
4. Create a reading script.
Note The name of the script must correspond with the name of the stored procedure.
5. Modify the reading script corresponding to the following example.
```
System.Inputparam = System.Partnernumber  ;Inputparam of the
StoredProcedure
 db.StartProcedure("ProcedureName", 0)
 db.ApplyDataSet()
```
6. To modify the SQL query, complete the following substeps.
    1. Remove the select statement.
    2. Define the parameter and data type in the following format.
    ParameterName=ParameterTypeDataFieldType
    Example
    ```
    Inputparam=IS
     Outputparam1=OS
     Outputparam2=OI
    ```

```
Outputparam3=O@
```

The character marked in green determines the parameter type.

Parameter Type	Description
I	Input parameter
O	Output parameter
B	Input / Output parameter
R	Return value

The character marked in yellow determines the data field type.

Data Field Type	Description
S	String
I	Integer
F	Float
D	Date
T	Time
@	Datetime

Updating Data Objects

If the structure of a table has altered, you have to update the corresponding data object that accesses the table.

To update data objects, complete the following steps.

1. In the bottom pane of the **Navigator**, select one or more data objects, then right-click one of the selected data objects and click **New** > **Data Object**.
2. In the **Data Object Assistant** window, select the **Update data objects** option and click **Next**.

3. In the **Refresh selected objects** window, check your selection and click **Next**.
4. In the **Data Object Assistant** window, click **Update Data Objects**.

Properties of a Data Object

Property	Description
Active Reading Script	Name of the active reading script.
Description	Free text description.
DataLink	Name of the DB table, DB column, or the SAP BAPI that the selection uses during execution.
DataSource	DataSource Type (Database or SAP).
DBAlias	Associated DB alias.

FieldRange	The text **[FIELDRANGE]** in an SQL statement is replaced with the content of the property **FieldRange** before execution.
	Note You must type FIELDRANGE in uppercase.
	Example
	• The SQL statement of the data object is defined as follows.
	`select [FIELDRANGE] from Contract where Number = :ContractNumber`
	• The property **FieldRange** contains the value "ContractId, Number".
	• At run time, the following SQL statement executes.
	`select ContractId, Number from Contract where Number = :ContractNumber`
	This feature is designed to specify the columns of a select statement but you can also use it to replace other parts of the SQL statement, for example, the table name.
	`select * from [FIELDRANGE] where Number = :ContractNumber`

IsActive	Defines whether the data object is active. This property is only available for child data objects.
Masterkey Column(s)	Masterkey(s) for the parent data objects master-detail relationship.

ObjectName	Unique object name, defines the name of the XML element.
RelationType	Type of relationship. **One2One:** An example of a One2One relationship is that which exists between partner and title - each partner has only one title. **One2Many:** An example of a One2Many relationship is that which exists between partner and address - each partner may have more than one address. This property is only available for child data objects.
Key Column(s)	Key(s) for the master-detail relationship.

Properties of a Data Object Variable

Property	Description
DataLink	Name of the DB table, DB column or the SAP BAPI, that the selection uses during execution.
Fieldlength	Length of the data field.
IsContainerColumn	TRUE = The data object variable is used to store values from another database column that contain one or more value pairs separated by a fixed character from each other. FALSE = The data object variable is not used to store other values. **Example** The database column **CustomerInfo** contains a person's first name and surname. The content has the general structure of FirstName=John, The 2 entries are separated by a COMMA. To automatically assign FirstName and 1. Create the following data object variables. • FirstName • Surname **Note** The value for the **DataLink** attribute must be identical to the key name in the database column, in this example FirstName or Surname.

	2. Set the **IsContainerColumn** property for both variables to `True`. 3. In the reading script, use the**DefineContainer**function to define the column and the separator. Example db.WorkWithQuery(Q1)db.DefineContainer(0, CustomerInfo , 44); 44 = ASCII-Code for the COMMA For more information, refer to **DefineContainer**.
ObjectName	Unique object name, defines the name of the XML element.

Selections

About Selections

Selections contain the root data object and any number of child objects. Their primary function is data retrieval.

There are two selection types available to the user.

- Active Selections read data from a database or from SAP.
- XSD Selections read data from an XML file.

Defining the Start Parameters

To specify start parameters for a selection, complete the following steps.

1. Open a selection for which you want to define start parameters.
2. In the **Object Inspector** pane, click **Start Parameter**, and then click the **Ellipsis** button.
3. In the **Start Parameter** dialog box, to add a start parameter, complete the following substeps.
    1. Click **Add**.
    2. In the **Key name** column, type the name of the key.
    3. Optional. In the **Prompt** column, type the text of the prompt.
    4. Optional. In the **Default Value** column, type a default value for the key.
    5. Optional. In the **Input Mask** column, type an input mask. You can refer to List of formatting characters to obtain the list of characters that you can use to define an input mask.
4. Optional. To delete a start parameter, select the key and then click **Delete**.
5. To copy the value of a cell, complete the following substeps.

1. Double-click a cell and select the value of the cell that you want to copy.
2. Right-click the cell and then click **Copy**.
3. Double-click the cell where you want to paste the selected value.
4. Right-click the cell and then click **Paste**.

6. To copy a row, complete the following substeps.
    1. Select the row you want to copy, right-click and then click **Copy**.
    2. Right-click the row where you want to paste the selected row and then click **Paste**.
7. To copy multiple rows, complete the following substeps.
    1. Select the rows you want to copy, right-click on the rows, and then click **Copy**.
    2. Right-click on the rows where you want to paste the selected rows and then click **Paste** .
    Note If you copy more than one row and paste the selected rows into one row, this creates subsequent rows in the grid with the selected content.
8. Click **OK**.

List of Formatting Characters

This table displays the characters that you can use to define an input mask.

Characters	Description
!	Specifies that optional characters display as leading spaces. If "!" character is not specified, optional characters display as trailing spaces in the mask.
>	Converts all following characters to lower case until the end of the mask is reached or the character "<" is detected. **Note**: This input mask character is currently not supported by the Web Client.
<	Converts all following characters to upper case until the end of the mask is reached or the character ">" is detected. **Note**: This input mask character is currently not supported by the Web Client.
<>	If both characters are used together, no case sensitivity check is made. The data specified is formatted in exactly the same way a user types it.
\	The character following "\" can be used as a placeholder in the input mask.
L	Valid entries are (A-Z, a-z). An entry is not compulsory.
I	

A a	Only alphanumeric characters are valid (A-Z, a-z, 0-9). An entry is not compulsory.
C c	Any character is allowed. An entry is not compulsory. **Note**: The following characters are currently not supported by the Web Client. • - • / • (•) • . • : • Space • + • , • @ • [•] • “ • ‘
0 9	Only a numeric character is allowed. An entry is not compulsory.
#	Only a numeric character or a plus or minus sign is allowed. An entry is not compulsory. **Note**: The plus and minus signs are currently not supported by the Web Client.
:	Used to separate hours, minutes, and seconds in defining a time.
/	Used to separate the date entries day, month, and year from each other.
;	Used to separate the three mask parameters from each other.
_	The underline character _ automatically inserts a space in the text. The cursor skips over these characters when you type data.

Notes

- You can use any character which is not listed in the above table as a first parameter of the mask and as placeholder for input. These characters are inserted automatically and skipped by the cursor on data entry.
- You can also use the special mask characters listed in above table as placeholders as long as you specify the "\" character first.

Saving a Selection as a Schema

To save a selection as an XML schema, complete the following steps.

1. Open the selection.
2. In the **Xdata Editor Tools** tab, click the **Export Xml Schema** button.
3. In the **Save as** dialog box, type a name for the file and click **Save**.

Xdata Editor Tools for Selections

Icon	Name	Description
	Refresh	Refreshes the data object in the selection. You can only execute this function when you have saved your modifications.
	Compile Selection	Compiles the selection.
	Open Data Object	Opens the selected data object.
	Export Xml Schema	For details, refer to Save Selection as a Schema
	Test Selection	For details, refer to Test a Selection
	Debug Selection	For details, refer to Debug a Selection
	Test Sets	For details, refer to Test Sets

	Decorator mode	For details, refer to Recording Data Retrieval
	SelVariable	For details, refer to Missing Data Objects or Attributes
	Show Data	Shows the result. **Note** Use the button **Save XML** in the following dialog box to save the results in an XML file.

Active Selections

Creating an Active Selection

You must have an Xdata license installed to create active selections.

To create an active selection, complete the following steps.

1. In the bottom pane of the **Navigator**, right-click and click **New** > **Selection**.
2. In the **Create New Selection** dialog box, type a name for the selection and click **OK**.
3. From the **Navigator** pane, drag the required data objects to the **Editor** window.
   Note All child objects of the main data object automatically transfer to the selection. To insert or delete child objects in a data object, open the data object itself.
4. Optional. Activate the data objects.
5. Modify the selection interface.
6. In the **Xdata Editor Tools** tab, click the **Compile** button.
7. Observe success or failure messages in the **Output** pane.

Activating a Data Object

When you first insert a data object into a selection, it is deactivated. All child objects are also deactivated. The following icons identify the status of a data object.

- The data object icon identifies an active object.
- The data object icon with a red cross identifies a deactivated data object.
  Note No data is read for deactivated data objects.

To activate a data object, complete one of the following steps.

- In the **Editor** window, right-click the data object and click **Active**.
- Press CTRL and click the data object.
- In the **Object Inspector** pane, set the **IsActive** property to **True**.

Refresh Data Objects in a Selection

If you have modified a data object while the selection that uses the data object is open, complete the following steps.

1. Save the modified data object.
2. In the **Editor** window of the selection, on the **Xdata Editor Tools** tab, click the **Refresh** button.
3. Optional. To update **Key** and **Masterkey** of the data object, in the **Editor** window of the selection, right-click the data object and click **Refresh**.
4. To make sure all modifications to the data object are registered in the selection, in the **Xdata Editor Tools** tab, click the **Compile** button.
   Note If you run the selection before you compile it, the compiled code already stored in the repository and not yet updated with the modifications to the data object executes.

Properties of an Active Selection

Property	Description
Start Parameter	Start parameter of the selection. For more information, refer to Define the start parameters.

Testing Selections

Executing a Selection

To execute a selection, complete the following steps.

1. Open the selection.
2. On the **Xdata Editor Tools** tab, click the **Test Selection** button.
3. In the **Selection Parameter** dialog box, type the selection parameter(s) and clickOK.
4. On the **Xdata Editor Tools** tab, click the **Show Data** button.
5. In the **Selection data** dialog box, in the left pane, click the individual data objects to display the data

of the selection.

6. Optional. To save the results in a local XML file, complete the following substeps.
    1. Click the **Save XML** button.
    2. In the **Specify the file name** dialog box, type a name of the file and click **Save**.

Debug a selection

To debug a selection, complete the following steps.

1. Open the selection.
2. In the **Xdata Editor Tools** tab, click the **Debug Selection** button.
3. In the **Selection Parameter** dialog box, type the selection parameter(s) and click **OK**.
4. Debug the selection using the buttons in the **Xdata Editor Tools** tab.
5. In the **Editor** window, in the left pane, click the individual data objects to display the data read.

Using Test Sets

Test sets contain parameters for frequently used selections.

For information about configuring test sets, refer to the *Content Composer Advanced Design and Setup Guide*.

To execute a test set, complete the following steps.

1. In the **Xdata Editor Tools** tab, in the **Test Sets** drop-down menu, select a test set.
2. In the **Xdata Editor Tools** tab, click the **Test Selection** button.

Recording a Data Retrieval

You can record a data retrieval in a file. You can use this file for tests on another computer by using the **Replay** option.

To record a data retrieval, complete the following steps.

1. Open the selection.
2. In the **Xdata Editor Tools** tab, in the **Recording mode** drop-down menu, select **Record**.
3. In the **Xdata Editor Tools** tab, click the **Test Selection** button.

4. In the **Selection Parameter** dialog box, type the selection parameters and click **OK**.
5. In the **Specify the name of the file** dialog box, type a name for the file and click **Save**.

Replay a data retrieval

To replay a data retrieval, complete the following steps.

1. Open the selection.
2. In the **Xdata Editor Tools** tab, in the **Decorator mode** drop-down menu, select **Replay**.
3. In the **Xdata Editor Tools** tab, click the **Test Selection** button.
4. In the **Select the recorded selection file** dialog box, select a previously recorded XML file and click **Open**.
5. In the **Xdata Editor Tools** tab, click the **Show Data** button.
6. In the **SelectionDataBrowser** dialog box, in the left pane, click the individual data objects to display the data read.
7. Optional. To save the results in a local XML file, complete the following substeps.
    1. Click the **Save XML** button.
    2. In the **Specify the name of the data file to be exported** dialog box, type a name for the file and click **OK**.

Missing Data Objects or Attributes

To trigger an error message when the selection contains a data object or attribute that was not found in the database, complete the following step.

- In the **Xdata Editor Tools** tab, select the option **Raise in SelVariable**.

XSD Selections

Importing an XSD Schema

To create and import an XSD schema, complete the following steps.

1. In the bottom pane of the **Navigator**, right-click and click **New** > **XSD Selection**.
2. In the **Create New XSD Selection** dialog box, in the **Name** box, type a name for the XSD selection, and click **OK**.
3. On the **Xdata Editor Tools** toolbar, click the **Import XSD Selection** button.

4. In the **Open** dialog box, select an XSD file and click **Open**.
5. Optional. To display the first level complex types in the first level of XSD selection, on the **Xdata Editor Tools** tab, select the **Show Complex Type** check box.

Exporting an XSD Schema

To export an XSD schema, complete the following steps.

1. On the **Xdata Editor Tools** toolbar, click the **Export XSD XML schema** button.
2. In the **Save As** dialog box, in the **File name** box, type an appropriate file name and click **Save**. The file is saved in **My Documents** directory by default. The file is saved in XSD format by default.

Comments in XSD Selections

The content of a node **xs:documentation** displays as text in parentheses behind the data object respectively the attribute.

Only the first element is shown, even if more than one **xs:documentation** exists.

Language attribute are not evaluated.

This particular XML-Parser behavior complies with **W3C recommendations** for XML and is described there as **Attribute-Value Normalization**.

Examples

```
<xsd:element name="PNR" minOccurs="0">
  <xs:annotation>
    <xs:documentation xml:lang="en">Partner number</xs:documentation>
    <xs:documentation xml:lang="de">Partnernummer</xs:documentation>
  </xs:annotation>
</xsd:element>


<xs:attribute name="ADR_City" type="xs:string" >
  <xs:annotation>
    <xs:documentation>Shipping address</xs:documentation>
  </xs:annotation>
</xs:attribute>
```

Properties of XSD Selections

Parameter	Description
Startparameter	Start parameter of the selection.
SetXMLData	Script that retrieves the XML data.

Execute an XSD selection

1. Open the XSD selection that you want to execute.
2. On the **Xdata Editor Tools** tab, click the **Test Selection** button. This button is only enabled when a script is attached to the XSD selection and an XSD schema is attached and available. The **Selection Parameter** dialog box appears if you add selection parameters to the XSD selection.
3. Optional. In the **Selection Parameter** dialog box, type the selection parameter and click **OK**. You may need to add multiple selection parameters.
4. On the **Xdata Editor Tools** tab, click the **Show Data** button.
5. In the **Selection data** dialog box, in the left pane, click the individual data objects to display the data of the XSD selection.
6. Optional. To save the results in a local XML file, complete the following substeps.

1. Click the **Save XML** button.
2. In the **Specify the file name** dialog box, type a name of the file and click **Save**.

Debug an XSD selection

To debug an XSD selection, complete the following steps.

1. Open the XSD selection that you want to debug.
2. On the **Xdata Editor Tools** tab, click the **Debug Selection** button. The **Selection Parameter** dialog box appears if you add selection parameters to the XSD selection.
3. Optional. In the **Selection Parameter** dialog box, type the selection parameter and click **OK**. You may need to add multiple selection parameters.
4. Debug the selection using the buttons on the **Xdata Editor Tools** tab.
5. On the **Xdata Editor Tools** tab, click the **Show Data** button.
6. In the **Selection data** dialog box, in the left pane, click the individual data objects to display the data of the XSD selection.
7. Optional. To save the results in a local XML file, complete the following substeps.

1. Click the **Save XML** button.
2. In the **Specify the file name** dialog box, type a name of the file and click **Save**.

Supported XSD Schema Constructs

Rules for Data Objects

Show All

An XSD schema construct is recognized as a data object when one of the following rules apply:

If it is a node type

- xs:sequence
- xs:all or
- xs:choice

the rules are applied to the child nodes.

Rule 1 Node xs:complexType

A node xs:complexType is recognized as a data object.

<xs:schema ... ...

...

<xs:complexType name="MType">

<xs:attribute name="MAttribute" type="xs:string"/>

</xs:complexType>

...

</xs:schema>

Rule 2 Node xs:element with subnode xs:complexType

A node xs:element is recognized as a data object, if it has a child node xs:complexType .

The name of the data object is specified in one of the two nodes in the attribute name.

If the node **xs:element** already has the attribute **name** defined, the attribute in a node **xs:complexType** defined as **name** is ignored.

<xs:schema ... ...

...

<xs:element name="X">

<xs:annotation>

<xs:documentation>This is X Element</xs:documentation>

</xs:annotation>

```
  <xs:complexType>
    <xs:attribute name="XAttribute" type="xs:string"/>
  </xs:complexType>
</xs:element>
...
</xs:schema>
```

Rule 3 Node xs:element with xs:complexType as type

A node **xs:element** is recognized as a data object, if it has as type a **xs:complexType** node.

```
<xs:schema ... ...
...
<xs:element name="X" type="Xtype" >
  <xs:annotation>
    <xs:documentation>This is X Element</xs:documentation>
  </xs:annotation>
</xs:element>
...
<xs:complexType name="XType">
  <xs:attribute name="XAttribute" type="xs:string"/>
</xs:complexType>
...
</xs:schema>
```

Rule 4 Node xs:element references a data object element

A node **xs:element** is recognized as a data object, if **ref** references a node **xs:element** that itself on application of one of the rules is a data object.

```
<xs:schema ... ...
...
  <xs:element ref="X" >
    <xs:annotation>
      <xs:documentation>This is X Element</xs:documentation>
    </xs:annotation>
```

```
  </xs:element>
  <xs:element ref="Y" />
...
<xs:element name="X">
  <xs:annotation>
    <xs:documentation>This is X Element</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:attribute name="XAttribute" type="xs:string"/>
  </xs:complexType>
</xs:element>
...
<xs:element name="Y">
  <xs:complexType>
    <xs:attribute name="YAttribute" type="xs:string"/>
  </xs:complexType>
</xs:element>
...
</xs:schema>
```

If a data object was recognized by one of these rules applying, the optional attribute **maxOccurs** is used to detect the relationship type:

Value	Description
maxOccurs	Type of Relationship
>1 or "Unbounded"	1:N
otherwise	1:1

Rules for Attributes

Show All

An XSD schema construct is recognized as an attribute when one of the following rules apply:

If the attribute type is specified in the node to be checked, the first check is to determine whether the node is xs:complexType.

If yes, the rules are applied to this node. If not, the rules are applied to the node originally specified.

Rule 1: Node xs:attribute is a subnode of a node xs:complexType

A node xs:attribute is recognized as an attribute, when it is a child node of a xs:complexType node.

<xs:schema ... ...

...

<xs:complexType name="MType">

  <xs:attribute name="MAttribute" type="xs:string"/>

</xs:complexType>

...

</xs:schema>

Rule 2: Node xs:element has one of the attributes ref, type or name specified

A node xs:element is recognized as an attribute, when one of the following criteria in the specified order .is met

Rule 2.1

If in the node xs:element the attribute ref is specified, it is recognized as an attribute, when the referenced node is recognized as an attribute after applying rule 1 or rule 2.

<xs:schema ... ...

...

  <xs:element ref="Y" />

.....

<xs:element name="Y">

  <xs:complexType>

    <xs:attribute name="YAttribute" type="xs:string"/>

  </xs:complexType>

</xs:element>

...

</xs:schema>

If this is not the case, rule 2.2 is checked.

Rule 2.2

If in the node xs:element already has the attribute type is specified, it is recognized as an attribute, when the specified type is either xs:simpleType or xs:complexType with xs:simpleContent.

```
<xs:schema ... ...
  <xs:element name="X" type="Xtype" />
  <xs:element name="Y" type="Ytype" />
...
<xs:simpleType name="Xtype">
  <xs:restriction base="xs:NMTOKEN">
    <xs:enumeration value="11"/>
    <xs:enumeration value="29"/>
  </xs:restriction>
</xs:simpleType>
...
<xs:complexType name="Ytype">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="prompt" type="xs:string" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
...
</xs:schema>
```

If this is not the case, rule 2.3 is checked.

Rule 2.3

If in the node xs:element the attribute name is specified, it is recognized as an attribute, if it contains a child node xs:simpleType or a child node xs:complexType with xs:simpleContent.

```
<xs:schema ... ...
...
  <xs:element name="X" >
    <xs:simpleType name="Xtype">
      <xs:restriction base="xs:NMTOKEN">
        <xs:enumeration value="11"/>
        <xs:enumeration value="29"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:element >
...
  <xs:element name="Y" >
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute name="prompt" type="xs:string" />
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element >
...
</xs:schema>
```

If none of the rules apply, the node is not recognized as an attribute.

If no attribute is recognized and the node is either

- xs:sequence
- xs:all, xs:choice
- xs:simpleContent or
- xs:extension

the rules are applied to the child nodes.

If one of the rules detects an attribute, the attribute **name** of the node or the attribute **ref** of a referenced node is used to determine the name of the attribute.

Odin

About Odin

Odin, the Open document interface, is an enterprise-wide solution for complex printing processes.

Odin is an integral part of the Content Composer end-to-end document production workflow, performing all tasks related to print and output management.

Odin can also perform the more complex operations of central printing such as envelope stuffing and post-processing.

You use Odin to complete the following actions.

- Convert documents to various formats like TIFF, PDF, and XPS.
- Process the documents in a batch.
- Send documents to different channels, such as email, printers, fax, and archival systems.

Envelopes

About Envelopes

Envelopes enable you to pack one or more documents together after printing.

Creating an Envelope

You must have an Odin license installed to create envelopes.

To create an envelope, complete the following steps.

1. In the bottom pane of the **Navigator**, right-click and click **New** > **Envelope**.
2. In the **Create New Envelope** dialog box, type a name for the envelope and click **OK**.
3. In the **Editor** window, in the **Weight in ounces** box**,** type the weight of the envelope.
   Note Weight specification is decisive in the calculation of envelope weight. It is therefore crucial to provide the exact weight when possible.
4. Optional. In the **Editor** window, in the **Description** box**,** provide a description for the envelope.

Note All available envelopes display in the bottom pane of the **Editor** window.

Enclosures

About Enclosures

Enclosures enable you to add an enclosure to an envelope either by a data driven rule or a definition in the postage definition.

Creating an Enclosure

You must have an Odin license installed to create enclosures.

To create an enclosure, complete the following steps.

1. In the bottom pane of the **Navigator**, right-click and click **New** > **Enclosure**.
2. In the **Create New Enclosure** dialog box, type a name for the enclosure and click **OK**.
3. In the **Editor** window, in the **Weight in ounces** box**,** type the weight of the enclosure.
   Note Weight specification is decisive in the calculation of envelope weight. It is therefore crucial to provide the exact weight when possible.
4. Optional. In the **Editor** window, in the **Tray** box, to change the value, type a number or click an arrow.
5. Optional. In the **Editor** window, in the **Description** box**,** provide a description for the enclosure.

Note All available enclosures display in the bottom pane of the **Editor** window.

Postage Definitions

About Postage Definitions

Postage definitions enable you to group various postage classes.

Creating a Postage Definition

You must have an Odin license installed to create postage definitions.

To create a postage definition, complete the following steps.

1. In the bottom pane of the **Navigator**, right-click and click **New** > **Postage Definition**.
2. In the **Create New Postage Definition** dialog box, type a name for the postage definition and click **OK**.
3. In the **Editor** window, right-click and click **Add New Postage Class**.
4. In the **Create Postage Class** dialog box, type a name for the postage class and click **OK**.
5. In the **Object Inspector** pane, select a value for the **Envelope** property.
   Note The envelope weight adds to the total weight.
6. In the **Object Inspector** pane, type a value for the **Maximum Weight** property.
7. Optional. In the **Object Inspector** pane, select one or more enclosures for the **FixedEnclosure** property.
   Note These enclosures combine with the total weight of the envelope. If, as a result, the weight exceeds that of the postage class, the envelope automatically jumps to the next postage class.
8. Optional. In the **Object Inspector** pane, select one or more enclosures for the **Postage-Class-Neutral Enclosure** property.
   Note These enclosures combine with the envelope only when the weight of the envelope does not

exceed postage class weight limit.

9. Optional. In the **Object Inspector** pane, provide a description for the postage definition.

Notes

All available envelopes display in the top right pane of the **Editor** window.

All available enclosures display in the bottom right pane of the **Editor** window.

Forms

About Forms

Create a form for each kind of paper you need to use.

Creating a Form

To create a form, complete the following steps.

1. In the bottom pane of the **Navigator**, right-click and click **New** > **Form**.
2. In the **Create New Form** dialog box, type a name for the form and click **OK**.
3. In the **Editor** window, in the **Weight in ounces** box**,** type the weight of the form.
   Note Weight specification is decisive in the calculation of envelope weight. It is therefore crucial to provide the exact weight when possible.
4. Optional. In the **Editor** window, in the **Description** box**,** provide a description for the form.

Note All available forms display in the bottom pane of the **Editor** window.

Printers

Email Printers

About Email Printers

Email printers enable you to send email messages through SMTP.

Documents convert according to the respective tray properties, such as Email-Subject, Email-Body, or Attachment/AttachmentCollect.

Creating an Email Printer

To create an email printer, complete the following steps.

1. In the bottom pane of the **Navigator**, right-click and click **New** > **Email Printer**.
2. In the **Create New Email Printer** dialog box, type a name for the email printer and click **OK**.
3. In the **Object Inspector** pane, modify the properties of the printer.
4. On the **Odin Editor Tools** tab, click the **Tray**

button.

5. In the **Add Tray** dialog box, type a name for the tray and click **OK**.
6. In the **Object Inspector** pane, modify the properties of the tray.

Online Printers

About Online Printers

Online printers enable you to print documents to a physical printer driver (online print or direct print).

Creating an Online Printer

To create an online printer, complete the following steps.

1. In the bottom pane of the **Navigator**, right-click and click **New** > **Online Printer**.
2. In the **Create New Online Printer** dialog box, type a name for the online printer and click **OK**.
3. On the **Odin Editor Tools** tab, click the **Printer Selection** button.
4. In the **Printer Selection** dialog box, select a printer and click **OK**.
5. In the **Object Inspector** pane, modify the properties of the printer.
6. On the **Odin Editor Tools** tab, click the **Tray** button.
7. In the **Add Tray** dialog box, type a name for the tray and click **OK**.
8. In the **Object Inspector** pane, modify the properties of the tray.

Creating an Online Printer for Local Printing

To allow local printing, complete the following steps.

1. In the bottom pane of the **Navigator**, right-click and click **New** > **Online Printer**.
2. In the **Create New Online Printer** dialog box, type a name for the online printer and click **OK**.
3. On the **Odin Editor Tools** tab, click the **Printer Selection** button.
4. Complete one of the following substeps.

- To enable local printing for generated bundles, in the **Printer Selection** dialog box, select a local printer and click **OK**.
- To forward the generated bundles to Odin without local printing, in the **Printer Selection** dialog box, select a non-physical printer, such as **Microsoft XPS Document Writer**, and click **OK**.

4. In the **Object Inspector** pane, modify the properties of the printer.
5. Optional. To prevent physical printing caused by Odin processing, in the **Object Inspector** pane, assign an empty script to the **OnPrint** event.
6. On the **Odin Editor Tools** tab, click the **Tray** button.
7. In the **Add Tray** dialog box, type a name for the tray and click **OK**.
8. In the **Object Inspector** pane, modify the properties of the tray.

Outsourcing Printers

About Outsourcing Printers

Outsourcing printers enable you to print a print stream to a physical printer driver through the OdinStreamPrint component (outsourcing print) and to define "virtual" tray names needed to convert streams with the OdinConvert component.

Creating an Outsourcing Printer

You must have an Odin license installed to create outsourcing printers.

To create an outsourcing printer, complete the following steps.

1. In the bottom pane of the **Navigator**, right-click and click **New** > **Outsourcing Printer**.
2. In the **Create New Outsourcing Printer** dialog box, type a name for the outsourcing printer and click **OK**.
3. On the **Odin Editor Tools** tab, click the **Printer Selection** button.
4. In the **Printer Selection** dialog box, select a printer and click **OK**.
5. In the **Object Inspector** pane, modify the properties of the printer.
6. On the **Odin Editor Tools** tab, click the **Tray** button.
   Note During streaming, the **virtual** tray names in the stream file are defined at the document level. These are mapped to the actual trays of the respective output format or target printer in the corresponding (Compart-)Profile, when the stream is converted.

7. In the **Add Tray** dialog box, type a name for the tray and click **OK**.
8. In the **Object Inspector** pane, modify the properties of the tray.

Note All available forms display in the bottom pane of the **Editor** window.

Printer Properties

Properties of an Email Printer

Conversion

Property	Description
LicenseFile	Path and name of the license file for the Compart .NET-Toolkit.
ProfilePath	Path to the Compart profiles set in the respective input and output filters.
TraceFile	Path and name of the trace file for the Compart .NET-Toolkit.
TraceLevel	Trace level

Events

Property	Description
OnConvert	Script that executes while converting the document into an attachment. You can use this event to define the name of an attachment file. **Available context** OdinEmailConvert
OnEmail	Script that executes when sending an email. SMTP Server settings are stored in the email printer. Email settings such as sender or subject are defined in the script. **Available context** OdinEmail
OnError	Script that executes in the event of an error. **Available context** OnError

General

Property	Description
Description	Free text description.
ErrorAction	Action taken in the event of an error. Value: StopStackPrint — Description: All printers are deactivated. Value: StopStackPrintForPrinter — Description: The printer that throws an error is deactivated.
Priority	Priority of a printer.
VisibleForSelection	TRUE = The printer is visible and the user can select it. FALSE = The printer is not visible and the user cannot select it.

SMTP Settings

Property	Description
SmtpPickupDirectory	Directory to which emails are saved, if SmtpDeliveryMethod.SpecifiedPickupDirectory is set.
Smtp Username	User name for SMTP server authentication.
SmtpDeliveryMethod	Specifies how the email is sent. Possible values: OdinSmtpDeliveryMethod.Network — Description: Emails are sent through the configured SMTP server (SmptHost / SmtpPort). Possible values: OdinSmtpDeliveryMethod.SpecifiedPickupDirectory — Description: Emails are copied to the

		directory specified in the property SmtpPickupDirectory and must be sent by an external program.
	OdinSmtpDeliveryMethod.PickupDirectoryFromIis	Emails are copied to the directory configured in the Internet Information Server (IIS).
Smtp-Domain	Domain or computer where login information is verified.	
Smtp-Host	Host on which the SMTP server runs.	
Smtp-Password	User name for SMTP server authentication. Encoded password to authenticate to the SMTP server. **Note** To encode the password, use the tool Encoder.exe in the installation directory.	
Smtp-Port	Port of the SMTP server.	
UseSsl	TRUE = An encoded SSL connection is used to communicate with the SMTP server. FALSE = The connection used to communicate with the SMTP server is not encoded.	
UseDefaultCredentials	TRUE = Authenticate to the SMTP server using the logged on user. FALSE = Authenticate to the SMTP server using SmptUserName and SmtpPassword.	

Events

Property	Description
OnAfterPrint	Script that executes after printing the document. **Available context** OdinPrint

OnBeforePrint	Script that executes before the document is printed. **Available context** OdinPrint
OnError	Script that executes in the event of an error. **Available context** OnError
OnPrint	If a script is specified, it is responsible for the print and is called for each document. **Available context** OdinStreamPrint

General

Property	Description
Description	Free text description.
ErrorAction	Action taken in the event of an error. Value: StopStackPrint — All printers are deactivated. Value: StopStackPrintForPrinter — The printer that threw an error is deactivated.
Printer	Physical printer name. The drop-down list shows all printers available to the system.
PrinterOptions	Comma separated string with options. Only the option **Duplex_AddPage** is currently supported. Duplex_AddPage=TRUE = When the option UseOneSpoolFile is set to `True`, an empty page adds to a duplex document with an odd number of pages. Duplex_AddPage=FALSE = The last page is set to simplex mode. The default value is FALSE.
Priority	Priority of a printer.

VisibleForSelection	TRUE = The printer is visible and the user can select it. FALSE = The printer is not visible and the user cannot select it.

Properties of an Outsourcing Printer

Events

Property	Description
OnAfterPrint	Script that executes after the document is printed. **Available context** OdinPrint
OnBeforePrint	The specified script executes before the document is printed. **Available context** OdinPrint
OnError	Script that executes in the event of an error. **Available context** OnError
OnPrint	If a script is specified, it is responsible for the print and is called for each document. **Available context** OdinStreamPrint

General

Property	Description
Description	Free text description.
Printer	Physical printer name. The drop-down list shows all printers available to the system.
PrinterOptions	Comma separated string with options. Only the option **Duplex_AddPage** is currently supported.

	Duplex_AddPage=TRUE = When the option UseOneSpoolFile is set to `True`, an empty page adds to a duplex document with an odd number of pages. Duplex_AddPage=FALSE = The last page is set to simplex mode. The default value is FALSE.
Priority	Priority of a printer.

Tray Properties

Properties of an Email Printer Tray

Property	Description
BinType	Defines how the Odin document is used and converted.

Value	Description
OdinEmailPrinterBinType.Attachment	The Odin document is converted to the corresponding output format, for example PDF, and then added to the email as a separate attachment. You can define the name of the attachment in the event **OnConvert**. The name of the attachment is inserted into the Odin job table in the **ODJ_EMAIL_ATTACHMENT** field.
OdinEmailPrinterBinType.AttachmentCollect	All Odin documents are converted to an attachment file in the corresponding output format, which means several documents combine to a single file. The name of the attachment is inserted into the Odin job table in the **ODJ_EMAIL_ATTACHMENT** field.
OdinEmailPrinterBinType.Body	The Odin document is converted to text

	or HTML format and used as the body text of the email.
OdinEmailPrinterBinType.Subject	The Odin document is generally converted to text format and used as the subject text. This also enables a single line Odin document to be converted to an individual reference text of the email.

Forms	Specifies 1:n forms assigned to the tray. You can assign each form to only one tray of the printer. Forms selected for documents and stored in Odin job data are used to map a document to a specific tray. **Note** Select all forms you want to use.
Outputformat	Output format of the attachment. The drop-down list shows all the possible output formats as defined in the ModusSuite.Common.PropertyEditors.dll.config file. **Note** Possible values for the output filter depend on the corresponding Compart .NET-Toolkit license.

Properties of an Online Printer Tray

Property	Description
Duplex Mode	Specifies the duplex mode for the tray used to print the Odin document.

Value		Description
Simplex	OdinDuplexSetting.Simplex	No duplex
Duplex_Horizontal	OdinDuplexSetting.Horizontal	Duplex on the short edge
Duplex_Vertical	OdinDuplexSetting.Vertical	Duplex on the long edge

Form(s)	Specifies 1:n forms assigned to the tray. You can assign each form only to one tray of the printer. Forms selected for documents and stored in Odin job data are used to map a document to a specific tray. **Note** Select all forms you want to use.
NextInputBin	Specifies the (virtual) tray name for the following pages.
PaperSource	The physical tray used for printing. The drop-down list shows all trays of the physical printer driver selected.

Properties of an Outsourcing Printer Tray

Property	Description
Duplex Mode	Specifies the duplex mode for the tray used to print the Odin document.

Value		Description
Simplex	OdinDuplexSetting.Simplex	No duplex
Duplex_Horizontal	OdinDuplexSetting.Horizontal	Duplex on the short edge
Duplex_Vertical	OdinDuplexSetting.Vertical	Duplex on the long edge

Form(s)	Specifies 1:n forms assigned to the tray. You can assign each form only to one tray of the printer. Forms selected for documents and stored in Odin job data are used to map a document to a specific tray. **Note** Select all forms you want to use.

NextInputBin	Specifies the (virtual) tray name for the following pages.
PaperSource	The physical tray used for printing. The drop-down list shows all trays of the physical printer driver selected.

Versioning

About Versioning

Versioning is an optional component. You must have a Versioning license installed to use the versioning functionality.

Almost all object types administrated in Content Composer Studio can be versioned except the following object types.

- Folders
- Profiles
- Roles
- Role Mappings
- DB Aliases
- Transfer Jobs
- Systems

New objects

A new object always creates as a work version with the number 0.1.

New versions

You can create new versions only for editable objects.

Repositories without versioning

If versioning is not active, all objects receive the version number 1.0.

Opening versioned objects

The newest version of the object, the object with the highest version number, will open for editing.

You can open older versions only in the view mode. You cannot save any modifications to older versions until you have reactivated the version.

Renaming versioned objects

Renaming an object has no effect on the version number. All versions and references use the new object name.

Changes to versioned objects

Creating a new version of a version controlled object has no effect on the version number of parent or child objects.

Example

The document template **Partner_Application** contains the text block **Account_Overview**.**Account_Overview** contains the child text block **Partner_Accounts**. **Partner_Accounts** contains the child textblock **Partner_Account_InterestRate**.

Document template **Partner_Application**			
	Text Block **Account_Overview**		
		Text Block **Partner_Accounts**	
			Text Block **Partner_Account_InterestRate**

Modifications to the text block **Partner_Accounts** only result in a new version of this text block.

Neither the child text block **Partner_Account_InterestRate** nor the child text block **Account_Overview** receive a new version number.

The document template **Partner_Application** also does not receive a new version number.

Version Types

A distinction is made between two types of object versions:

Version Type	Example
Release Versions	1.0, 2.0, 3.0
Work Versions	0.1, 1.1, 1.2, 2.1, 3.1, 3.2

Only the highest object version is the active version, regardless of whether it is a work version or a release version. Only this version is used by other objects.

The version number increments automatically. A manual assignment of version numbers is not possible.

Creating a New Work Version from a Release Version

It is **not** possible to directly create a new release version. To save changes to a release version, you must first create a new work version.

To create a new work version from a release version, complete the following steps.

1. Open the object.
2. Make the required modifications.
3. On the Quick Access Toolbar, click the **Create New Version** button.
4. Optional. In the **Create New Version** dialog box, in the **Label** box, type a label.
5. Click **OK**.

Example

You have edited the version **2.0** of an object. The object now receives the version number **2.1**.

Creating a New Work Version from a Work Version

To create a new work version from a work version, complete the following steps.

1. Open the object.
2. Make the required changes.
3. On the Quick Access Toolbar, click the **Create New Version** button.
4. In the **Create New Version** dialog box, select **Work Version**.
5. Optional. In the **Label** box, type a label.
6. Click **OK**.

Example

You have edited the version **2.1** of an object. The object now receives the version number **2.2**.

Note You **cannot** transfer this version to another system.

Creating a New Release Version from a Work Version

Prerequisites

- You must possess the **Create Release Version** right.
- Changes to the object were saved in a valid way.

To create a new release version from a work version, complete the following steps.

1. Open the object.
2. Make the required changes.
3. On the Quick Access Toolbar, click the **Create New Version** button.
4. Optional. In the **Create New Version** dialog box, in the **Label** box, type a label.

5. In the **Create New Version** dialog box, select **Release Version**.
6. Click **OK**.

Example

You have edited the version **2.2** of an object. The object now receives version number **3.0**.

Note You can transfer this version to another system.

Modifying a Work Version Without Creating a New Version

To modify a work version without creating a new version, complete the following steps.

1. Open the object.
2. Make the required changes.
3. On the Quick Access Toolbar, click the **Save** button.

Example

You have edited the work version **2.1** of an object. After saving, the version number remains **2.1**.

Displaying the Object History

To display the history of an object, complete the following steps.

1. In the **Navigator**, right-click the object and click **Show Object History**.
2. Optional. To open an object version, in the **Object History** dialog box, select a version from the list and click the **Show Object** button.
   Note: To use this function, the user must possess the right **Open Historic Object Version (47)**.

Note You cannot generate or execute older versions of document templates, bundles and processes.

Reactivating an Older Version

Reactivating an object version is only available for editable objects. You cannot reactivate write-protected objects.

To reactivate an older version of an object, complete the following steps.

1. In the bottom pane of the **Navigator**, right-click the object and click **Show Object History**.
2. In the **Object History** dialog box, select an entry from the list and click the **Show Object** button.
   Note: To use this function, the user must possess the right **Open Historic Object Version (47)**.
3. On the **Quick Access Toolbar**, click the **Reactivate Version** button.
   Note This function creates a copy of the version opened as a new work version. For example, an object has the newest version **2.2**. You have opened and reactivated the version **1.1** of the object. A new work version with the number **2.3** is created as a copy of version **1.1**.

For more information, refer to Transfer Jobs.