StreamServe Persuasion SP5 Scripting Reference

Reference Manual

Rev C
Use of this software program is protected by copyright law, patent law, and international treaties. No part of this software product, associated documentation (including online help tools) may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Open Text Corporation. Information in this documentation is subject to change without notice. Open Text Corporation assumes no responsibility or liability for any errors or inaccuracies that may appear in this software program. All brands, product names and trademarks of other companies mentioned in this software program are used for identification purposes only and are acknowledged as property of the respective company. Companies, names and data used in examples in this software program are fictitious unless otherwise noted.

Open Text Corporation offers no guarantees and assumes no responsibility or liability of any type with respect to third party products and services, including any liability resulting from incompatibility between the third party products and services and the products and services offered by Open Text Corporation and its direct/indirect subsidiaries. By using Open Text Corporation software products and the third party products or services mentioned in this software product, you agree that you will not hold Open Text Corporation and its direct/indirect subsidiaries responsible or liable with respect to use of such third party products or services.

The trademarks, logos, brands, and service marks found in this software program are the property of Open Text Corporation or other third parties. You are not permitted to use such marks without the prior written consent of Open Text Corporation or the third party that owns the marks.

Use of any Open Text Corporation products or services with any third party products or services not mentioned in this documentation is entirely at your own risk.
Contents

Scripting in StreamServe ................................................................. 15
The script editor .................................................................................. 16
Checking the script syntax .............................................................. 16
Script execution .................................................................................. 18
Job execution phases ......................................................................... 18
Formatting phase ................................................................................ 19
Retrieved scripts ............................................................................... 20
Before scripts .................................................................................... 20
After scripts ....................................................................................... 21
StreamServe scripting language specifics ........................................ 22
Variables ............................................................................................ 22
Global variables ................................................................................ 23
Local variables .................................................................................. 25
Field references ................................................................................. 28
Array variables .................................................................................. 28
Multi dimensional arrays .................................................................. 29
Example of array usage .................................................................... 29
Operators ............................................................................................ 30
Delimiters ........................................................................................... 32
Expressions ........................................................................................ 32
Variables in numeric expressions ..................................................... 33
String Extraction Expression (PageIN only) ........................................ 33
Assignment statements ...................................................................... 34
Conditional statements ...................................................................... 34
The if statement ................................................................................ 35
The if...else statement ...................................................................... 36
The switch...case statement ............................................................ 36
The while statement ......................................................................... 37
The do...while statement .................................................................. 38
The for statement .............................................................................. 39
Comments ......................................................................................... 39
Script functions ................................................................................ 40
Numeric precision ............................................................................. 40
Escape sequences ............................................................................. 40
Built-in script functions .................................................................... 42
Function files .................................................................................... 43
When to use function files .............................................................. 44
User created script functions with arguments .................................. 44
Using local variable names as arguments to a function .................... 45
<table>
<thead>
<tr>
<th>Category</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Substitution tables</td>
<td>47</td>
</tr>
<tr>
<td><strong>Script functions reference</strong></td>
<td>49</td>
</tr>
<tr>
<td>General functions</td>
<td>50</td>
</tr>
<tr>
<td>Array handling functions</td>
<td>50</td>
</tr>
<tr>
<td>Block handling functions</td>
<td>50</td>
</tr>
<tr>
<td>Job ID handling functions</td>
<td>51</td>
</tr>
<tr>
<td>Job resource handling functions</td>
<td>51</td>
</tr>
<tr>
<td>Job status handling functions</td>
<td>52</td>
</tr>
<tr>
<td>Log file functions</td>
<td>52</td>
</tr>
<tr>
<td>Overlay handling functions</td>
<td>52</td>
</tr>
<tr>
<td>Page handling functions</td>
<td>53</td>
</tr>
<tr>
<td>PageIN specific functions</td>
<td>53</td>
</tr>
<tr>
<td>StreamIN specific functions</td>
<td>54</td>
</tr>
<tr>
<td>Ungrouped functions</td>
<td>54</td>
</tr>
<tr>
<td>Check digit calculation functions</td>
<td>57</td>
</tr>
<tr>
<td>COM functions</td>
<td>58</td>
</tr>
<tr>
<td>Date and time functions</td>
<td>59</td>
</tr>
<tr>
<td>Dispatcher functions</td>
<td>60</td>
</tr>
<tr>
<td>Document sorting and bundling functions</td>
<td>61</td>
</tr>
<tr>
<td>Email functions</td>
<td>65</td>
</tr>
<tr>
<td>File handling functions</td>
<td>66</td>
</tr>
<tr>
<td>LDAP functions</td>
<td>68</td>
</tr>
<tr>
<td>Connecting to the directory server</td>
<td>70</td>
</tr>
<tr>
<td>Non-persistent connections</td>
<td>70</td>
</tr>
<tr>
<td>Persistent connections</td>
<td>71</td>
</tr>
<tr>
<td>LDAP examples</td>
<td>71</td>
</tr>
<tr>
<td>Lotus Notes functions</td>
<td>75</td>
</tr>
<tr>
<td>Lotus Notes examples</td>
<td>76</td>
</tr>
<tr>
<td>Message traversing functions</td>
<td>77</td>
</tr>
<tr>
<td>Message traversing examples</td>
<td>79</td>
</tr>
<tr>
<td>Numeric operation functions</td>
<td>80</td>
</tr>
<tr>
<td>Position formatting functions</td>
<td>81</td>
</tr>
<tr>
<td>Preview functions</td>
<td>82</td>
</tr>
<tr>
<td>Preview examples</td>
<td>82</td>
</tr>
<tr>
<td>Project related functions</td>
<td>84</td>
</tr>
<tr>
<td>Queue and repository functions</td>
<td>85</td>
</tr>
<tr>
<td>Session handling functions</td>
<td>86</td>
</tr>
<tr>
<td>StoryTeller functions</td>
<td>87</td>
</tr>
<tr>
<td>String handling functions</td>
<td>89</td>
</tr>
<tr>
<td>Substitution table functions</td>
<td>90</td>
</tr>
<tr>
<td>Executing a substitution table function only once</td>
<td>91</td>
</tr>
<tr>
<td>Type and conversion functions</td>
<td>93</td>
</tr>
<tr>
<td>ODBC functions</td>
<td>94</td>
</tr>
<tr>
<td>ODBC startup argument</td>
<td>96</td>
</tr>
</tbody>
</table>
Script functions in alphabetical order ............................................................. 97

A ................................................................................................................ 97
  Abs ............................................................................................................ 97
  AddSubst ................................................................................................. 97
  Amount ..................................................................................................... 99
  AmountValue ........................................................................................... 99
  ArchiveBegin .......................................................................................... 100
  ArchiveDisable ....................................................................................... 100
  ArchiveEnable ......................................................................................... 100
  ArchiveEnd ............................................................................................. 100
  ArraySize ............................................................................................... 100
  AtJobEnd ................................................................................................. 102
  AttachmentBegin ..................................................................................... 102
  AttachmentEnd ......................................................................................... 103

B ................................................................................................................ 104
  Base64DecodeFile .................................................................................... 104
  Base64DecodeString ............................................................................... 105
  Base64DecodeStringUsingCodepage ....................................................... 105
  Base64EncodeFile ................................................................................... 106
  Base64EncodeString ............................................................................... 106
  Base64EncodeStringUsingCodepage ....................................................... 107
  BinAnd ..................................................................................................... 107
  BinOr ....................................................................................................... 108
  BinXOr ..................................................................................................... 108

C ................................................................................................................ 109
  CallBlock ............................................................................................... 109
  CallProc .................................................................................................. 109
  CancelJob ............................................................................................... 110
  Clear ....................................................................................................... 111
  ClearSubst .............................................................................................. 111
  ClearVars ............................................................................................... 111
  COMCreateObject .................................................................................. 112
  COMDestroyObject ................................................................................ 113
  COMInvoke ............................................................................................ 113
  COMSetProperty .................................................................................... 114
  COMGetProperty ..................................................................................... 115
  ConnectorPage ......................................................................................... 115
  ConnectorPageActual .............................................................................. 116
  ConnectorPages ...................................................................................... 116
  ConvCurrMsgToUC ................................................................................ 117
  CopyFile ................................................................................................ 117
  CreateGlobalSerNo ............................................................................... 118
  CurrDocProperty ................................................................................... 119
  CurrInFileName ...................................................................................... 119
  CurrInRealPath ...................................................................................... 120
  CurrInSavePath ...................................................................................... 120
  CurrJobOwner ......................................................................................... 121
  CurrJobProperty .................................................................................... 122
  CurrMetadata ......................................................................................... 123
  Curr ReposProperty ............................................................................... 124

D ................................................................................................................ 124
  DayOfWeek ............................................................................................. 124
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DayOfYear</td>
<td>125</td>
</tr>
<tr>
<td>DeclareMetadata</td>
<td>125</td>
</tr>
<tr>
<td>DeleteFile</td>
<td>126</td>
</tr>
<tr>
<td>DeleteJobResource</td>
<td>126</td>
</tr>
<tr>
<td>DelSubstKey</td>
<td>127</td>
</tr>
<tr>
<td>Dformat</td>
<td>128</td>
</tr>
<tr>
<td>DiffDate</td>
<td>129</td>
</tr>
<tr>
<td>Div</td>
<td>129</td>
</tr>
<tr>
<td>DocInsertedPages</td>
<td>130</td>
</tr>
<tr>
<td>DocPageActual</td>
<td>130</td>
</tr>
<tr>
<td>DocPage</td>
<td>131</td>
</tr>
<tr>
<td>DocPages</td>
<td>131</td>
</tr>
<tr>
<td>DtisoFormat</td>
<td>132</td>
</tr>
<tr>
<td>DumpVariables</td>
<td>133</td>
</tr>
<tr>
<td>EnableEnvelopeCheck</td>
<td>133</td>
</tr>
<tr>
<td>EndDocument</td>
<td>134</td>
</tr>
<tr>
<td>EndMessage</td>
<td>134</td>
</tr>
<tr>
<td>EraseSubst</td>
<td>135</td>
</tr>
<tr>
<td>Eval</td>
<td>136</td>
</tr>
<tr>
<td>Execute</td>
<td>137</td>
</tr>
<tr>
<td>Exist</td>
<td>138</td>
</tr>
<tr>
<td>FileClose</td>
<td>139</td>
</tr>
<tr>
<td>FileCopy</td>
<td>139</td>
</tr>
<tr>
<td>FileDelete</td>
<td>140</td>
</tr>
<tr>
<td>FileMove</td>
<td>141</td>
</tr>
<tr>
<td>FileOpen</td>
<td>142</td>
</tr>
<tr>
<td>FileReadLn</td>
<td>143</td>
</tr>
<tr>
<td>FileSize</td>
<td>143</td>
</tr>
<tr>
<td>FileWrite</td>
<td>144</td>
</tr>
<tr>
<td>FileWriteLn</td>
<td>145</td>
</tr>
<tr>
<td>FindInArray</td>
<td>145</td>
</tr>
<tr>
<td>FindStringExactInArray</td>
<td>146</td>
</tr>
<tr>
<td>FirstBlockInst</td>
<td>147</td>
</tr>
<tr>
<td>FirstBlockOnPage</td>
<td>148</td>
</tr>
<tr>
<td>FlushOutput</td>
<td>148</td>
</tr>
<tr>
<td>ForcePoll</td>
<td>149</td>
</tr>
<tr>
<td>FrameOverflow</td>
<td>149</td>
</tr>
<tr>
<td>GetAttachmentContentEncoding</td>
<td>150</td>
</tr>
<tr>
<td>GetAttachmentContentType</td>
<td>151</td>
</tr>
<tr>
<td>GetAttachmentCount</td>
<td>151</td>
</tr>
<tr>
<td>GetAttachmentFile</td>
<td>151</td>
</tr>
<tr>
<td>GetAttachmentOriginalFile</td>
<td>152</td>
</tr>
<tr>
<td>GetConnectorValue</td>
<td>152</td>
</tr>
<tr>
<td>GetCurrModuleinfo</td>
<td>153</td>
</tr>
<tr>
<td>GetCurrSegDoc</td>
<td>153</td>
</tr>
<tr>
<td>GetCurrX</td>
<td>154</td>
</tr>
<tr>
<td>GetCurrY</td>
<td>154</td>
</tr>
<tr>
<td>GetCurrBlockY</td>
<td>154</td>
</tr>
<tr>
<td>GetCurrObjY</td>
<td>155</td>
</tr>
<tr>
<td>GetDate</td>
<td>155</td>
</tr>
</tbody>
</table>
GetSubst... ................................................................. 186
GetSubstColCount .................................................. 187
GetSubstKey ............................................................ 188
GetSubstKeyCount .................................................. 189
GetTextFormattingInfo .......................................... 190
GetTime ................................................................. 192
GetTopJobId ............................................................ 193
GetXOffs ............................................................... 193
GetYOffs ............................................................... 193

H ............................................................................... 194
HexStr ........................................................................ 194

I .................................................................................. 194
In2Mm ................................................................. 194
In2Pt ................................................................. 195
Int ................................................................. 195
InConnectorName .................................................. 195
IncProcStatCounter ............................................. 196
InQueueName ........................................................ 196
InsertOverlay ........................................................ 197
InsertPage ........................................................... 197
IoErrText .............................................................. 198
IsamAdd ............................................................... 198
IsamClose ............................................................. 199
IsamCreate ........................................................... 199
IsamErase ............................................................. 200
IsamGet ............................................................... 200
IsamOpen ............................................................. 201
IsAmount .............................................................. 201
IsAmountValue ..................................................... 201
IsDate ................................................................. 202
IsFirstDocInEnvelope .......................................... 203
IsFirstPageInEnvelope ......................................... 203
IsFirstSegment ..................................................... 204
IsInsertingPage ..................................................... 204
IsLastDocInEnvelope ........................................... 205
IsLastPageInEnvelope .......................................... 205
IsLastSegment ...................................................... 206
IsNum ................................................................. 206
IsPreview ............................................................. 207

J ................................................................................ 207
JobBegin .............................................................. 207
JobInsertedPages ................................................ 207
JobEnd ................................................................. 208

L .................................................................................. 208
LastBlockInst ....................................................... 208
LastBlockOnPage .................................................. 209
LdapAddAttrBinaryValue ..................................... 209
LdapAddAttrValue ................................................ 210
LdapConnect ........................................................ 211
LdapConnectPersistent ....................................... 212
LdapConnectSSL .................................................. 214
LdapConnectSSLCCA ........................................... 215
MsgFrameGetValue .......................................................................................... 250
MsgFrameOpen ............................................................................................ 251
MsgFrameValueExist ...................................................................................... 252
MsgGetValue ................................................................................................... 254
MsgNextBlock .............................................................................................. 255
MsgNumFields .............................................................................................. 256
MsgOpen ........................................................................................................ 258
MsgProcCountId .......................................................................................... 259
MsgProcGetValue .......................................................................................... 260
MsgProcOpen ............................................................................................... 262
MsgProcValueExist ...................................................................................... 263
MsgValueExist ............................................................................................... 264
NextDoc ......................................................................................................... 266
NextPage ...................................................................................................... 266
NextProc ....................................................................................................... 267
NextSegment ................................................................................................. 267
NewDate ....................................................................................................... 267
NewJob .......................................................................................................... 268
NewPage ....................................................................................................... 268
NFormat ......................................................................................................... 270
NoFormFeed ................................................................................................. 270
Num .............................................................................................................. 271
NumberOfEvents .......................................................................................... 271
NumberOfEventsEx ....................................................................................... 272
OdbcBeginTrans .......................................................................................... 273
OdbcCommitTrans ........................................................................................ 273
OdbcConnect ............................................................................................... 274
OdbcConnectW ............................................................................................ 275
OdbcClearReorderText ................................................................................... 276
OdbcCloseQuery ........................................................................................... 276
OdbcDate2Date .............................................................................................. 276
OdbcDate2Time .............................................................................................. 277
OdbcDisconnect ............................................................................................ 277
OdbcExecute ................................................................................................. 278
OdbcExecuteEx ............................................................................................. 279
OdbcGetOne ................................................................................................. 279
OdbcGetFirst ............................................................................................... 280
OdbcGetNext ............................................................................................... 281
OdbcRollbackTrans ....................................................................................... 282
OdbcSetCodepage ......................................................................................... 283
OdbcSetReorderText ...................................................................................... 283
OutputFile .................................................................................................... 283
OutputLXFJobResource ................................................................................. 285
OutputLXFJobResourcePmOffs .................................................................... 285
OutputString ................................................................................................. 286
OverlayGetNumPages ..................................................................................... 287
Page .............................................................................................................. 287
PageOfPages ............................................................................................... 288
Pages ............................................................................................................ 288
PatternResult ............................................................................................... 289
PPDocCount ................................................................. 290
PreProc ................................................................. 290
PreProcLog ............................................................. 291
Pt2In ................................................................. 291
Pt2Mm ................................................................. 292
Q ................................................................. 292
QueuePage ........................................................... 292
QueuePages .......................................................... 292
R ................................................................. 293
ReadSubst ............................................................ 293
ReplaceJobIDAttribute ........................................ 294
ReplaceJobIDDDateAttribute ................................ 294
ReplaceJobIDNumAttribute ................................... 294
RGBcolor ............................................................. 294
Round ................................................................. 294
S ................................................................. 295
SaveCurrMetadata ............................................... 295
SavePPMetadata ................................................... 296
SaveSegMetadata .................................................. 296
SessionEnd ........................................................... 297
SessionGetVariable ............................................... 297
SessionSetVariable ................................................ 298
SessionStart .......................................................... 299
SetCopies ............................................................. 299
SetCurrMetadata .................................................... 300
SetCurrMetadataNum ............................................ 300
SetCurrX ............................................................. 301
SetCurrY ............................................................. 301
SetDestPath ........................................................... 302
SetExtJobId ............................................................ 303
SetFontProperties .................................................. 303
SetJobDescr ........................................................... 304
SetJobFailed .......................................................... 304
SetJobOwner ............................................................ 305
SetLanguage ........................................................... 305
SetMetaDataJob ....................................................... 306
SetPPMetadata ..................................................... 306
SetPPMetadataNum .................................................. 307
SetPrnXoffs ............................................................ 308
SetPrnYoffs ............................................................ 308
SetSegMetadata ....................................................... 309
SetSegMetadataNum .................................................. 310
SetSerNo ............................................................... 310
SetSubst ............................................................... 311
SetXoffs ............................................................... 312
SetYoffs ............................................................... 313
SinCount ............................................................... 313
SinVal ................................................................. 313
SinValExist ............................................................. 313
Skip ................................................................. 314
Sort ................................................................. 314
SortSegDoc ............................................................ 315
StartTimer ............................................................. 316
StEvalXPath ................................................................. 316
StGetParagraphPageXMM ........................................... 317
StGetParagraphPageXPt .............................................. 317
StGetParagraphPageYMM ........................................... 318
StGetParagraphPageYPt .............................................. 318
StGetParagraphXMM .................................................. 319
StGetParagraphXPt .................................................... 319
StGetParagraphYMM .................................................. 319
StGetParagraphYPt .................................................... 320
StGetProcessingProperty ......................................... 320
StGetProperty ............................................................ 321
StGetPropertyMm ....................................................... 324
StGetPropertyPt ........................................................ 326
StIsEmpty ............................................................... 327
StLanguageLookup ................................................... 327
StMsgAddNode .......................................................... 328
StMsgAttachXML ....................................................... 328
StMsgAttachXMLFile ............................................... 329
StMsgSaveToFile ....................................................... 330
StopTimer ................................................................. 330
StSetProperty ........................................................... 331
StSetPropertyMm ....................................................... 331
StSetPropertyPt ........................................................ 332
StoreJobIDAtribute ................................................... 332
StoreJobIDDateAttribute .......................................... 332
StoreJobIDNumAttribute ........................................... 332
str ................................................................. 333
StrBik ................................................................. 333
StrIdX ................................................................. 333
StrLadj ................................................................. 334
StrLen ................................................................. 335
StrQTok ................................................................. 335
StrrBik ................................................................. 336
StrrTok ................................................................. 336
StURIExist ............................................................. 337
Subst ................................................................. 337
SubArr ................................................................. 338
SubStr ................................................................. 339
SubStrRepl ............................................................ 340
T .................................................................................. 341
Terminate ............................................................ 341
ToLower .............................................................. 342
ToUpper .............................................................. 343
U ................................................................................ 343
UpdateDocStatus ................................................... 343
UpdateExternalCompletionStatus ......................... 344
UpdateJobStatus .................................................... 344
UpdatePPDocStatus ............................................... 345
UpdatePPJobStatus ............................................... 345
UpdateSegDocStatus ............................................. 346
UpdateSegJobStatus ............................................. 347
URLClose ............................................................ 348
URLExist ............................................................ 349
URLOpen.......................................................................................................................... 349
URLReadLn......................................................................................................................... 350
Utf8DecodeString ............................................................................................................. 351
Utf8EncodeString ............................................................................................................. 351
W...................................................................................................................................... 351
WeekOfYear ....................................................................................................................... 351
WriteSubst ........................................................................................................................... 352

Lotus Notes Fetch Wizard.................................................................................................... 355
Generating a code block with the Lotus Notes Fetch wizard ............................................ 357
   Information required by the Lotus Notes Fetch wizard ..................................................... 358
   Generating the code block ............................................................................................... 358
Editing a generated code block ......................................................................................... 359
Deleting a generated code block ....................................................................................... 360
Scripting in StreamServe

You can create scripts to use advanced StreamServer functionality. To create scripts, you must have some programming experience, and a good working knowledge of StreamServer applications and tools. It is also assumed that you are familiar with the StreamServe terminology.

In this chapter
- The script editor on page 16
- Script execution on page 18
- StreamServe scripting language specifics on page 22
- Built-in script functions on page 42
- Function files on page 43
- Substitution tables on page 47
The script editor

You create scripts in the script editor. The script editor user interface is described in the figure and table below:

![Script Editor Interface](image)

**Figure 1** The script editor.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The script view where you type in the scripts. You can enter as many scripts as you need in this view.</td>
</tr>
<tr>
<td>2</td>
<td>The syntax error view. All syntax error messages are displayed here. Each error message shows the type of error, and where to find it (which line) in the script view.</td>
</tr>
<tr>
<td>3</td>
<td>Syntax check. You can check the syntax manually (click Check Syntax) or select to always check the syntax before closing the script editor.</td>
</tr>
<tr>
<td>4</td>
<td>Before, After, or Retrieved – specifies when to trigger the scripts displayed in the script view. Note that the script view changes when you change the trigger.</td>
</tr>
</tbody>
</table>

Checking the script syntax

You should verify that a script is correct before you close the script editor. You can check the script syntax manually, or configure the script editor to always check the syntax before closing.

**To check the syntax manually**

1. Click **Check Syntax** in the Script Editor. If a script contains an error, it will be shown in the syntax error view.
2 Double-click the error notification. A red bullet is displayed at the beginning of the line where the syntax error is located.

**To check the syntax on exit**

1 Select **Check Syntax on exit**. If there is a syntax error when you try to close the script editor, you will be prompted to correct the error. The syntax error will be shown in the syntax error view.

2 Double-click the error notification. A red bullet is displayed at the beginning of the line where the syntax error is located.
Script execution

Scripts are run during different phases when the StreamServer processes the data. When to run a script depends on what you want to achieve. Scripts can be categorized as follows:

- Retrieved scripts – see Retrieved scripts on page 20
- Before scripts – see Before scripts on page 20
- After scripts – see After scripts on page 21

Script execution order

The figure below illustrates where scripts are applied to a job with one Message, one Event, and two Processes.

![Figure 2 - Script execution order.](image)

Job execution phases

Job processing is divided into three execution phases, first the input phase, then the formatting phase, and finally the output phase.

Input phase

The input phase starts when an input connector finds data, and ends when the complete input job is stored in the input queue.

Formatting phase

During the formatting phase, all transformations and formatting is done. All scripts are also run during this phase. Simply put, this is where StreamServer does the actual work and processes the job. The formatting phase starts when a job is picked from an input queue for processing, and ends when the complete input job has been processed, and all output jobs are successfully stored in the output queues. See Formatting phase on page 19.

Output phase

The output phase starts when all output jobs are stored in the output queues, and ends when all the output jobs are delivered.
Formatting phase

The primary tasks during the formatting phase are as follows:

- Detect documents (Events) in the input job.
- Extract information from the documents and create Messages.
- Run Processes to create the desired output.
- Store this output in output queues.

Conditional formatting using scripts is also applied during this phase. The formatting phase is separated into three phases:

- Collect phase
- Pre-process phase
- Process phase

Collect phase

In the collect phase, StreamServer analyzes the incoming data, detects documents (Events), and creates the Messages. Sorting of incoming documents is also done during the collect phase.

When StreamServer analyses the incoming data, and detects a document, the job begins internally in StreamServer. StreamServer will now start to scan the input for fields and new documents. All fields found are stored in a Message associated with the current Event. If a field is configured to create a variable, this is done at this stage. If a new Event is found, it will be added to a list of found Events, and any fields found after this will be associated with the Message for the new Event. This procedure continues until a list of all Events, with all fields, in the input job is created. The Messages are stored in sequence in a temporary storage.

As the last step in the collect phase, the stored Messages can be sorted. Sorting is based on the values of the script variables. To add more logic to sorting, a Retrieved script can be executed after each Event has been collected. At this stage, the variables for the Event are already created, and can be updated in the script, which in turn affects the result of sorting.

Pre-process phase

The pre-process phase starts when the collect phase ends. During the pre-process phase, all Events and Processes in the input job are executed without producing any output data. For example, during the pre-process phase, the number of pages are calculated, page breaks are located, overlays are added, etc.

The pre-processing order is as follows:

1. The first Event is pre-processed, followed by the Processes for this Event.
2. The next Event is pre-processed, followed by the Processes for this Event.

This continues until all Events are pre-processed.
StreamServer reads Messages stored in the temporary storage one Message at a time. For each Message, the internal Message structure is recreated in memory, and then all Processes and scripts related to that Event are executed. Each Message, including the information gathered during the pre-process phase, is again written to the temporary storage.

**Process phase**

The process phase starts when the pre-process phase ends. All information needed to create the desired output is available during this phase. The Processes and scripts are run as in the pre-process phase, but now the information retrieved during the pre-process phase is used to create the output.

Before the process phase starts, all variables are rolled back to the values they had before the pre-process step – this since all scripts in are run in the pre-process phase. The same applies to all external data sources the scripts have updated. For example, databases updated using ODBC script functions.

StreamServer reads Messages stored in the temporary storage one Message at a time. For each Message, the internal Message structure is recreated in memory, and then the Processes and scripts are executed. The output is sent from the Processes, and stored in the output queues.

**Retrieved scripts**

A retrieved script is applied per Event, and is used to act upon the data received from the source application. It is run after the Event – including fields, blocks, and variables – is created.

**Before scripts**

Before scripts can be used to assign values to variables, perform calculations, decide what the next action will be, etc.

**Before job scripts**

A Before job script is applied per Runtime job. It is run before the Events are created.

**Before Message scripts**

A Before Message script is applied per Message. It is run:

- After the Events are created, and after the data from the Events is sorted.
- Before the Processes are created.

**Before Process scripts**

A before Process script is applied per Process. It is run before the Process is created.
Before Process items scripts
You can apply Before scripts to Process items (page, field, frame, block, etc.). The script is run before the item is initiated.

After scripts
After scripts can be used for the same purposes as Before scripts, but since calculated values are known when the script is executed, After scripts can act upon calculated data.

After Process items scripts
You can apply After scripts to Process items (page, field, frame, block, etc.). The script is run after the item is initiated.

After Process scripts
An After Process script is applied per Process. It is run after the Process is created.

After Message scripts
An After Message script is applied per Message. It is run after all Processes are created.

After job scripts
An After job script is applied per Runtime job. It is run:

- After all Processes are created.
- Before the output is created.
StreamServe scripting language specifics

The StreamServe scripting language includes the following:

- Variables
- Field references
- Statements (if, while, switch)
- Functions

Script example

```plaintext
$copiesLeftToPrint = &copies;
Clear($copytext);
while (Num($copiesLeftToPrint) >= 0)
{
    if(Num($copiesLeftToPrint) != Num(&copies)) {
        $copytext = "COPY";
    }
    CallProc("procInvoice");
    $copiesLeftToPrint--;
}
```

In this section

- Variables on page 22
- Field references on page 28
- Array variables on page 28
- Operators on page 30
- Delimiters on page 32
- Expressions on page 32
- Assignment statements on page 34
- Conditional statements on page 34
- The while statement on page 37
- Comments on page 39
- Script functions on page 40
- Numeric precision on page 40
- Escape sequences on page 40

Variables

Variables in StreamServe can be global or local.
Global variables

Global variables are used in scripts, function files, and in the Design Center tools. For example, you can add global variables to a StoryTeller page.

The scope is within the execution of the job and the variables are cleared when the job is completed.

You can create field variables in Events. For example, you can assign the global variable $countryCode to the field CountryCode. You can also create global variables in scripts.

Examples

Example 1  Global variable $countryCode used in a script.

```java
if($countryCode = "USA")
{
  $overlay = "USA.lxf";
}
```

Example 2  Global variable $invoiceNumber used in the PageOUT Advanced Text tool.

```
Invoice number: $invoiceNumber
```

Example 3  Calculating the sum of two numbers.

In this example, the global variables $amount1 and $amount2 are created as field variables in an Event. The global variable $total in this script calculates the sum of $amount1 and $amount2.

$total = Num($amount1) + Num($amount2);

Result
If $amount1 = 10 and $amount2 = 12, $total = 22.

Example 4  Concatenating strings.

In this example, the global variables $firstName and $surName are created as field variables in an Event. The global variable $fullName in this script concatenates the strings $firstName and $surName.

$fullName = $firstName + " " + $surName;

Result
If $firstName = Mike and $surName = Jones, $fullName = Mike Jones.
StreamServe scripting language specifics

Scripting in StreamServe

Example 5  Assigning substrings to global variables.

In this example, the global variable $name = “Mr Mike Jones” is used as input to create the global variables $nameSur (should contain “Jones”) and $nameFirst (should contain “Mike”).

The global variable $name is a string variable containing 13 characters. The string and position of the characters are shown in the table below.

<table>
<thead>
<tr>
<th>String</th>
<th>M</th>
<th>r</th>
<th>M</th>
<th>i</th>
<th>k</th>
<th>e</th>
<th>J</th>
<th>o</th>
<th>n</th>
<th>e</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

The global variable $nameSur is a substring of $name, starting with the character in position 9 (J) and ending with the last character in the string (s):

$nameSur = $var1(9);  //$nameSur contains "Jones".

The global variable $nameFirst is a substring of $name, starting with the character in position 4 (M) and is four characters long:

$nameFirst = $var1(4,4);  //$nameFirst contains "Mike".

$ prefix

Global variable names must begin with $:

$<variable name>

Valid characters

A global variable name can consist of letters (A-Z, a-z), digits (0-9) and underscore (_). The name must begin with a letter or underscore.

Declaration of global variables

You do not have to declare a global variable before using it. By default, global variables are initiated to an empty string.

Field variables

You can create global variables ($<variable name>) for any field in an Event. For example, if the Event contains the field countryCode, you can create a field variable called $countryCode. When the data is processed, $countryCode will get the current value of the field countryCode. The field variable can be used in scripts as well as in the Design Center formatting tools.

Note that creating field variables affects performance, so you should only create field variables when necessary. If you only need to retrieve the field value in a script, you can use field references instead. See Field references on page 28.

Global variables within strings

You can use curly brackets to separate variables from adjacent text.
Example 6  Using global variables within strings.

$name = "Tom";
$typeof = "tri";
$result = Eval ("Dear $name, you may use the ${typeof}cycle");
$result will contain "Dear Tom, you may use the tricycle"
If $typeof has the value motor, $result will contain "Dear Tom, you may use the motorcycle"

Drawbacks of global variables

The following are issues that you could have when using global variables. To avoid it, one possibility is to use local variables. See Local variables on page 25.

Performance
To use a global variable in a numerical expression, or as an argument to a function that expects a numeric argument, you must convert the string type to a numerical value, which affects performance.

Project errors
To re-use a global variable within the scope of a job, you must first assign a value to the variable, or clear it, which are tasks that could introduce errors to your Project.

Function file portability
When you import a function file to your Project, the variables used within the function becomes part of the global scope. This introduces a risk that they interfere with variables used in existing scripts. Thus the portability of function files is limited.

Local variables

As opposed to global variables, which have a scope within the execution of a job, local variables have a scope that is limited to the compound statement where it is declared. A compound statement is a set of statements contained within curly brackets {}, or within begin/end.

You can use local variables in scripts and function files.

For reasons to use local variables, see Drawbacks of global variables on page 25.

No prefix
A local variable does not have the $ prefix.

Declaration
You must declare a local variable before any other statements in the compound statement, and you can not declare a variable more than once within the compound statement.
You declare a local variable with the following syntax:

```plaintext
type name-list;
```

- **type** Specifies the type of variable, **num** or **str**.
  - **num** corresponds to a numeric value, as in `num($variable)`, and can only be assigned numeric values.
  - **str** corresponds to a string value, just as a global $variable. The difference is the scope.

- **name-list** A comma separated list of the names of the declared variables.
  
  A name can consist of one or more letters, digits or underscore.
  
  Optionally, the name can be followed by one or more dimension specifications within square brackets, [], for example to declare an array variable.

  **Note:** The name is case insensitive, but must start with a letter.

**Example 7**  
**Declarations**

```plaintext
num i, j;
str name, company, country;
```

**Initialization**
Scalar variables (as opposed to e.g. array variables) can be declared and initialized simultaneously.

**Note:** If you do not initialize a local variable when you declare it, a numeric variable has value 0 and a string variable is empty, until you give them values.

**Example 8**  
**Declaration and initialization**

```plaintext
num i, j, count = 3;
strs name="Tom", company="OpenText";
```

**Reserved keywords**
You can not use the following keywords as variable names:

- and, begin, bool, break, case, continue, default, do, else, end, false, for, goto, if, int, not, num, return, str, struct, switch, true, while.
Nested compound statements

Compound statements of a script can be nested. A local variable is passed to all child compound statements of the compound statement where it is declared. You can re-initialize or re-declare a local variable in a child compound statement, but the variable will retain its previous state when the child compound statement goes out of scope.

Example 9  Nested compound statements

```plaintext
Script 1 "Event" //This is the outermost compound statement type where a local variable can exist, in this case a "retrieved" script.
str myStr = "Hello World!"; //Outermost declaration
{
  //comp. statement 1
  {//comp. statement 2, nested inside if 1.
   log(0, "myStr: " + myStr); // "myStr: Hello World!"
       // The value is passed on from where it is declared and initialized.
  }
}
{
  //comp. statement 3
  str myStr = "Hello Kitty!"; //local declaration in comp. statement 3
  log(0, "myStr: " + myStr); // "myStr: Hello Kitty!"
}
log(0, "myStr: " + myStr); // "myStr: Hello World!" again, because comp. statement 3 went out of scope
```

Using begin/end

As an alternative to `{}` you can use Begin/End to set limits on your compound statement.

Example 10  Begin / End

```plaintext
Begin
  num i, j=1;
  while (i++ < 10)
    log(j, "still in the loop");
End
```
Field references

If you only need to retrieve the value of a field in a script, you do not have to create a field variable. In this case you can use a field reference (&<field name>).

Examples

Example 11  Field reference &countryCode used in a script.

\[
\text{if(\&countryCode} = \text{"USA"} ) \\
\{ \\
\quad \$overlay = \text{"USA.lxf"}; \\
\}
\]

& prefix

Field references must begin with &:

&<field name>

Used in scripts only

You can only use field references in scripts, and not in the Design Center formatting tools.

Case sensitive

Field references are case sensitive. For example, if the field name in the Event is countryCode, the field reference must be &countryCode and not &Countrycode.

⚠️  In StoryTeller, you can not use a &<fieldname> reference. To retrieve the value of a field reference, use the StEvalXPath function.

For example, $value = StEvalXPath("fieldname");

Array variables

Instead of using several unique variables to store information, you can use an array variable. The elements in an array variable are numbered, starting with zero. The number (index) of an element is inside square brackets.

Examples

Example 12  Array variable $phonenumber.

\[
\text{
$phonenumber[0] = "031-90510";  \\
$phonenumber[1] = "020-222222";  \\
$phonenumber[2] = "031-183413";  
}\]
Multi dimensional arrays

You can create multi dimensional array variables, where each element contains one index per dimension.

**Examples**

*Example 13*  
Two dimensional array variable $arr.

$arr[0][0] = $tableValue1A;  
$arr[0][1] = $tableValue1B;  
$arr[1][0] = $tableValue2A;  
$arr[1][1] = $tableValue2B;

*Example 14*  
Indexing values in multidimensional arrays using explicit index values.

You can use explicit index values:

$array[0][0] = 1;

*Example 15*  
Indexing values in multidimensional arrays using calculated index values.

$x = 0;  
y = 0;  
$array[$x][$y] = 1;

**Example of array usage**

This script is an example on how to builds arrays. The following variables are used:

- $ca_no_dialled – at the time the script is called, this variable contains a phone number from a field in a phone bill.
- $ca_call_charge – contains the amount charged for the corresponding call.
- $arrtele – the array containing phone numbers.
- $arramt – the corresponding array of summed costs for calls placed to the phone number.
- $totalcharge – a sum that starts out empty and gets $ca_call_charge added to it each time the script is run. It ends up a sum of all the call charges in the phone bill.
- $subtotalcharge – a similar sum, but it is read and reset to 0 every now and then (each page) by a different script.
// Add charge for this call to sum.
$totalcharge = $totalcharge + $ca_call_charge;

// Add charge for this call to subtotal.
$subtotalcharge = $subtotalcharge + $ca_call_charge;

// See if the current phone number is already in the array of phone numbers.
$retno = FindInArray($arrtele, $ca_no_dialled);
if($retno == "-1") // If -1, then the phone number is a new one.
{
    // Determine current size of array, so that we know where to append.
    $retsize = ArraySize($arrtele);
    /* Add the new phone number to our phone numbers array, and put the corresponding
     * cost in the $arramt array. */
    $arrtele[$retsize] = $ca_no_dialled; // The new phone number
    $arramt[$retsize] = $ca_call_charge; // amount charged
}
else
    /* The current phone number was found in the array. Add the amount charged to
     * the corresponding $arramt element. */
    $arramt[$retno] = $arramt[$retno] + $ca_call_charge;

---

**Operators**

In the StreamServe scripting language, you can use the operators described in the table below.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>$var = expression1 - expression2</td>
</tr>
<tr>
<td>+</td>
<td>Addition</td>
<td>$var = expression1 + expression2</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>$var = expression1 * expression2</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>$var = expression1 / expression2</td>
</tr>
<tr>
<td>++</td>
<td>Incrementation</td>
<td>$var++</td>
</tr>
<tr>
<td>Operator</td>
<td>Description</td>
<td>Syntax</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>--------</td>
</tr>
<tr>
<td>--</td>
<td>Decrementation</td>
<td>$\text{var}--$</td>
</tr>
<tr>
<td>-=</td>
<td>Subtraction Assignment</td>
<td>$\text{var} -= \text{expression}$</td>
</tr>
<tr>
<td>+=</td>
<td>Addition Assignment</td>
<td>$\text{var} += \text{expression}$</td>
</tr>
<tr>
<td>=</td>
<td>Assignment</td>
<td>$\text{var} = \text{expression}$</td>
</tr>
<tr>
<td>==</td>
<td>Equality</td>
<td>\text{expression1} == \text{expression2}</td>
</tr>
</tbody>
</table>
StreamServe scripting language specifics

Scripting in StreamServe

### Delimiters

The StreamServe scripting language includes the delimiters described in the table below.

<table>
<thead>
<tr>
<th>Delimiter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>;</td>
<td>Terminates statements.</td>
</tr>
<tr>
<td>&quot; &quot;</td>
<td>Delimiter for text strings.</td>
</tr>
<tr>
<td>,</td>
<td>Separates items in lists.</td>
</tr>
<tr>
<td>.</td>
<td>Decimal separator.</td>
</tr>
<tr>
<td>( )</td>
<td>Surrounds expressions, and parameter lists in functions.</td>
</tr>
<tr>
<td>{ }</td>
<td>Surrounds consecutive statements, making them appear as one simple statement. Equivalent to <code>begin ... end</code>.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Surrounds indexes in an array.</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Surrounds hexadecimal numbers, and lists of hexadecimal numbers.</td>
</tr>
</tbody>
</table>

### Expressions

Expressions are created using constants, variables, functions calls, and operators. The evaluation of an expression always results in a value.

#### Examples

**Example 16**  
*Numeric expression.*

```plaintext
$\text{value} = \text{Num}(\text{amount}) - 1;
```
StreamServe scripting language specifics

Scripting in StreamServe

Example 17  String expression.

\[ $\text{fullName} = $\text{firstName} + "-" + $\text{surName}; \]

Example 18  Boolean expression.

\[
\text{if} \ (\text{Num}($\text{cost}) < 100)
\{
   \text{value} = \text{Num}($\text{amount}) - 1;
\}
\]

Variables in numeric expressions

You must use the script function \text{Num} to assign numeric values to variables in numeric expressions. If you do not, the variables will be treated as strings.

Examples

Example 19  Using the Num script function.

\[
\text{a} = 20;
\text{b} = \text{Num}($\text{a}) + 30;
\text{Result: } \text{b}=50
\]

Example 20  Not using the Num script function.

\[
\text{a} = 20;
\text{b} = $\text{a} + 30;
\text{Result: } \text{b}=2030
\]

String Extraction Expression (PageIN only)

For PageIN data, there is a special expression to extract information from an input ASCII file:

\text{column row length}

For example, to extract the string in column 37 to 43 on row 14 the expression would be:

\[ 37 14 7; \]

This string extraction expression is often used as the right-hand side in assignment statements, for example:

\[ $\text{strg} = 37 14 7; \]
Assignment statements

Assignment statements are used to assign values to variables.

Syntax

```plaintext
variable=expression;
```

Examples

**Example 21** Assigning 1 to $counter.

```plaintext
$counter = 1;
```

**Example 22** Assigning “Hello” to $message.

```plaintext
$message = "Hello";
```

Conditional statements

There are several types of conditional statements:

- The if statement
- The if...else statement
- The switch...case statement
- The while statement
- The do...while statement
- The for statement
The if statement

Syntax

```plaintext
if (expression)
{
    statement1;
    [statement2;]
    ...
    [statementN;]
}
```

Examples

Example 23  if statement.

```plaintext
if($countryCode = "USA")
{
    $overlay = "invoiceUSA.lxf";
}
```
The if...else statement

Syntax

```c
if (expression)
{
    statement1;
    [statement2;]
    ...
    [statementN;]
} else
{
    statementN+1;
    [statementN+2;]
    ...
    [statementN+N;]
}
```

Examples

**Example 24**  if...else statement.

```c
if($countryCode = "USA")
{
    $overlay = "invoiceUSA.lxf";
}
else
{
    $overlay = "invoiceDefault.lxf";
}
```

The switch...case statement

Use the `switch...case` statement when there are several different statements, or groups of statements, where it would be confusing to use a large number of "ifs" and "elses".
Syntax

```java
switch (expression)
{
    case value1:
        statement1;
        [break;]
    case value2:
        statement2;
        [break;]
    case valueN:
        statementN;
        [break;]
    [default:
        defaultStatement;]
}
```

The `expression` is tested, and depending on the value, the corresponding `case` is executed. If there is no corresponding case, the `default` case is executed.

A common error when using the `switch...case` statement is to forget that the execution starts in the selected `case`, and continues until `break`. If you leave out `break`, the execution will continue into the next `case`.

Examples

**Example 25** `switch...case` statement.

```java
switch ($countryCode)
{
    case "USA":
        $overlay = "invoiceUSA.lxf"
        break;
    case "Canada":
        $overlay = "invoiceCanada.lxf";
        break;
    case "UK":
        $overlay = "invoiceUK.lxf"
        break;
    default:
        $overlay = "invoiceDefault.lxf"
}
```

The while statement

A while statement is executed as long as the expression is true.
Syntax

```
while(expression)
{
    statement1;
    [break;]
    [statement2;]
    [continue;]
    [statement3;]
}
```

Examples

**Example 26 While statement.**

```
$copiesLeftToPrint = &copies;
Clear($copytext);
while (Num($copiesLeftToPrint) >= 0)
{
    if(Num($copiesLeftToPrint) != Num(&copies)) {
        $copytext = "COPY";
    }
    CallProc("procInvoice");
    $copiesLeftToPrint--;}
```

The do...while statement

Syntax

```
do
{
    statement[s];
}
while (expression);
```

Examples

**Example 27 do...while statement.**

```
{
    num i;
do
        {log(1, "in the loop");
i++;
    }
while (i <=4);
```
The for statement

**Syntax**

```plaintext
for (statement1; expression; statement2) statement3
```

**Examples**

**Example 28** for loop statement.

```plaintext
{
    num max = 5, i;
    for (i=0; i < max; i++)
        log(1, str(i));
}
```

**Comments**

You can comment out a line, or part of a line, using double slash (//). To comment out a whole section in a script, you can use /* to start the section and */ to end the section to comment out.

**Examples**

**Example 29** Line comments.

```plaintext
if($ourref = "Mike Jones")     //Condition
{
    $tray = "Lower tray";      //Statement
}
//Log(1,$tray);
```
Example 30  Section comment

```plaintext
if($ourref = "Mike Jones")
{
    $tray = "Lower tray";
}
/*if($ourref = "Bella Houdini")
{
    $tray = "Upper tray";
}
Log(1,$tray);*/
```

Script functions

You can use pre-defined script functions in your scripts. A script function is a portion of code, which performs a specific task and is relatively independent of the remaining code.

Built-in script functions

There are several built-in script functions that you can use. See Built-in script functions on page 42.

User created script functions

You can also create your own script functions. See Function files on page 43.

Numeric precision

The numeric precision is limited to 15 digits. This limitation is set by the hardware. The StreamServer uses the host’s double precision floating point datatype (double), which has 15 digits of precision.

Escape sequences

The less-than sign (<) is a reserved character. To use this character in a string, you must either enclose the whole string within escape sequences `< [ and ] >`, or use the ASCII code for the < character.

Examples

Example 31  Escaping the < character using <[ and ]> escape sequences

```plaintext
$select_2 = "<[Select list.questions from list where list.id < ]>" + $num;
If $num=2, this gives the following result:
Select list.questions from list where list.id < 2;
```
Example 32  **Escaping the < character using the ASCII code for the < character**

```plaintext
$select_2 = "Select list.questions from list where list.id <3c>" + $num;

If $num=2, this gives the same result as above:
Select list.questions from list where list.id < 2;
```

Example 33  **Using the Log function to append a string containing a < character to the log file**

```plaintext
Log (0, "<[Value < 5]>")
```
Built-in script functions

A script function is a portion of code, which performs a specific task and is relatively independent of the remaining code. There are several built-in script functions that you can use. You can also create your own script functions. See Function files on page 43.

Covers a wide range of use

The built-in script functions cover a wide range of use, such as:

• Calculation.
• String Manipulation.
• Output Customization.

General syntax

A script function call always begins with the function name followed by parentheses - with or without arguments:

FunctionName([arg1,][arg2,][argN,]);

Script functions reference

All built-in script functions are described in Script functions reference on page 49.
Function files

A function file is a text file containing user created script functions. The example below shows a function file with four user created script functions. Each script function in this example returns a specific RGB-value.

```plaintext
CodePage UTF8

//Returns dark orange (RGB 255,101,0)
func OrangeDark ()
{
    return RGBcolor(255,101,0);
}

//Returns light orange (RGB 255,154,0)
func OrangeLight ()
{
    return RGBcolor(255,154,0);
}

//Returns dark blue (RGB 0,0,255)
func BlueDark ()
{
    return RGBcolor(0,0,255);
}

//Returns light blue (RGB 99,207,255)
func BlueLight ()
{
    return RGBcolor(99,207,255);
}
```

Creating a function file

A function file is a resource that you must add to the appropriate resource set. This means you can create the function file directly from the resource set, or import an existing function file.

Calling a user created script function

You call a user created script function the same way as you call a built-in script function (SetDestPath, CallProc, etc.). For example, if you have a script function called setArea, you can call this function from a script:

```plaintext
(area = setArea($length, $width);
```

Functions returning values

You can create functions that return values. The function example below returns the RGB value 255,154,0:

```plaintext
func OrangeLight ()
{
    return RGBcolor(255,154,0);
}
```

This function can be called from a script:

```plaintext
/color = OrangeLight ()
```
Functions not returning values

You can also create functions that do not return values. The function example below includes the `SetDestPath` script function:

```plaintext
func setInvoicePath ()
{
    SetDestPath(&invoiceNumber + "_" + &yourReference);
}
```

This function can be called from a script:

```plaintext
setInvoicePath ();
```

When to use function files

Reusing scripts

You should use function files when you need to reuse scripts. If you do so, you will have one source for the script, and you do not have to create or edit the same script several times. For example, if you have two Processes, and need to add the same Before Process script to both Processes, you can create a script function that includes the script. Then, at Before Process for both Processes, you call the script function instead of entering the same script twice.

Gathering related scripts

You can also use function files to gather related scripts in the same text file.

User created script functions with arguments

You can create script functions that need arguments in the input. In the script function, you use `#1` to retrieve the first argument in the input, `#2` to retrieve the second argument, and so on.

You can also declare arguments to a function as local variable names. See Using local variable names as arguments to a function on page 45.

Example 34  Function with two input arguments

```plaintext
Function
func FileNameString ()
{
    $namestring = #1 + "_" + #2;
    return $namestring;
}
```

In this example, the script function `FileNameString` uses two input arguments to create the string “customer name_invoice number”.

Function call

```plaintext
$nameAndNumber = FileNameString (&yourReference, &invoiceNumber);
```

Result

```plaintext
```

If \&yourReference is Mike Jones, and \&invoiceNumber is 1020, the value of \$nameAndNumber is “Mike Jones_1020”.

---

**Example 35  Function with substrings**

**Function**

```
func ChangeDateFormat ()
{
    $month = #1(1,2);  //Character 1 and 2 in input string
    $day = #1(4,2);    //Character 4 and 5 in input string
    $year = #1(7,4);   //Character 7 to 10 in input string
    $newformat = $year + "-" + $month + "-" + $day;
    return $newformat;
}
```

In this example, the script function `ChangeDateFormat` changes the date format `mm/dd/yyyy` to `yyyy-mm-dd`.

**Function call**

```
$dateFormatSWE = ChangeDateFormat (&invoiceDate);
```

**Result**

If \&invoiceDate is 05/21/2008, the value of \$dateFormatSWE is 2008-05-21.

---

**Using local variable names as arguments to a function**

You can declare arguments to a function as local variable names. Variables declared in the function declaration, or in the function body, are local to the scope of the function call. Compared to when all variables were global, there is no risk of overwriting global variables, thus making custom functions more portable.

Such a function declaration has the following form:

```
func name([type name,])
{
    [statements]
    [return value]
}
```

**Example 36  Function declaration**

```
func myfunc(str firstname, str lastname, num age)
{
    if (age > 29)
        return firstname + lastname;
    else
        return 0;
```
Using the #N syntax to read arguments

It is still possible to use the #N syntax to retrieve the arguments to a function even when you use variable names as arguments.

Example 37  #N syntax when arg variables are declared

```plaintext
{   
    log(1, echo("hello!"));   
} 

func echo(str string)   
{   
    return #1;   
}
```

- Array variables are currently not supported in function arguments or return values.
- You can not initialize argument variables in the function declaration. This must be done in the function body.
Substitution tables

There are a number of built-in script functions that can be used to retrieve or modify values in substitution tables. See Substitution table functions on page 90 for more information.

A substitution table contains several entries (rows), and each entry contains two or more columns. The first column is the key, which is a unique identifier for each entry. The remaining columns contain the values for the entry (value 0, value 1, ..., value N).

Example 38 Substitution table.

In the substitution table below, alfred is the key and hitchcock and 1899 the value of the entry. The value consists of two columns, 0 and 1.

```plaintext
//multicolumn!
//key       value 0     value 1
alfred    hitchcock 1899
```
The script functions reference contains descriptions of all built-in script functions.

In this chapter

- General functions on page 50
- Check digit calculation functions on page 57
- COM functions on page 58
- Date and time functions on page 59
- Dispatcher functions on page 60
- Document sorting and bundling functions on page 61
- Email functions on page 65
- File handling functions on page 66
- LDAP functions on page 68
- Lotus Notes functions on page 75
- Message traversing functions on page 77
- Numeric operation functions on page 80
- Position formatting functions on page 81
- Preview functions on page 82
- Project related functions on page 84
- Queue and repository functions on page 85
- Session handling functions on page 86
- StoryTeller functions on page 87
- String handling functions on page 89
- Substitution table functions on page 90
- Type and conversion functions on page 93
- ODBC functions on page 94
- Script functions in alphabetical order on page 97
General functions

In this section
- Array handling functions on page 50
- Block handling functions on page 50
- Job ID handling functions on page 51
- Job resource handling functions on page 51
- Job status handling functions on page 52
- Log file functions on page 52
- Overlay handling functions on page 52
- Page handling functions on page 53
- PageIN specific functions on page 53
- StreamIN specific functions on page 54
- Ungrouped functions on page 54

Array handling functions

The table below contains a short description of each script function. For information about syntax, examples, etc., see the full description of each script function.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArraySize</td>
<td>Counts the number of elements stored in an array.</td>
</tr>
<tr>
<td>FindInArray</td>
<td>Searches for a value in an array. Returns the position of the array element that contains the specified value or superset of the specified value.</td>
</tr>
<tr>
<td>FindStringExactInArray</td>
<td>Searches for a value in an array. Returns the position of the array element that contains exactly the specified value.</td>
</tr>
</tbody>
</table>

Block handling functions

The table below contains a short description of each script function. For information about syntax, examples, etc., see the full description of each script function.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallBlock</td>
<td>Calls a Free Block.</td>
</tr>
<tr>
<td>FirstBlockInst</td>
<td>Determines if the current block is the first of its kind in a frame.</td>
</tr>
</tbody>
</table>
### General functions

### Script functions reference

#### Job ID handling functions

The table below contains a short description of each script function. For information about syntax, examples, etc., see the full description of each script function.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetExtJobId</td>
<td>Returns the external ID of the current job.</td>
</tr>
<tr>
<td>GetIntJobId</td>
<td>Returns the internal ID of the current job.</td>
</tr>
<tr>
<td>GetTopJobId</td>
<td>Returns the internal ID of the input job (top job).</td>
</tr>
<tr>
<td>SetExtJobId</td>
<td>Sets the external ID of the current job.</td>
</tr>
</tbody>
</table>

#### Job resource handling functions

The table below contains a short description of each script function. For information about syntax, examples, etc., see the full description of each script function.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeleteJobResource</td>
<td>Marks job resources to be removed. The marked resources are deleted when all references to the job resource are released.</td>
</tr>
<tr>
<td>GetJobResourceIndex</td>
<td>Retrieves the index of a job sent to a Job Resource output connector.</td>
</tr>
<tr>
<td>OutputLXFJobResource</td>
<td>Adds an LXF overlay from a job resource to a PageOUT Process.</td>
</tr>
<tr>
<td>OutputLXFJobResourcePrnOffs</td>
<td>Adds an LXF overlay from a job resource to a PageOUT Process, and enables the use of SetPrnXOffs and SetPrnYOffs to adjust the overlay offset.</td>
</tr>
</tbody>
</table>
General functions

Script functions reference

Job status handling functions

The table below contains a short description of each script function. For information about syntax, examples, etc., see the full description of each script function.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetJobStatus</td>
<td>Returns the status of the current job.</td>
</tr>
<tr>
<td>SetJobFailed</td>
<td>Specifies the current job status as failed.</td>
</tr>
</tbody>
</table>

Log file functions

The table below contains a short description of each script function. For information about syntax, examples, etc., see the full description of each script function.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log</td>
<td>Appends a string to the log file.</td>
</tr>
<tr>
<td>PreProcLog</td>
<td>During the pre-process phase, appends a string to the log file.</td>
</tr>
</tbody>
</table>

Overlay handling functions

The table below contains a short description of each script function. For information about syntax, examples, etc., see the full description of each script function.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OutputLXFJobResource</td>
<td>Adds an LXF overlay from a job resource to a PageOUT Process.</td>
</tr>
<tr>
<td>OutputLXFJobResourcePrnOffs</td>
<td>Adds an LXF overlay from a job resource to a PageOUT Process, and enables the use of SetPrnXOffs and SetPrnYOffs to adjust the overlay offset.</td>
</tr>
<tr>
<td>OverlayGetNumPages</td>
<td>Returns the total number of pages in a multi-page overlay.</td>
</tr>
</tbody>
</table>
Page handling functions

The table below contains a short description of each script function. For information about syntax, examples, etc., see the full description of each script function.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectorPage</td>
<td>Returns the number of pages (in the current job) that have been processed so far on the current connector.</td>
</tr>
<tr>
<td>ConnectorPages</td>
<td>Returns the total number of pages (in the current job) that will be processed on the current connector.</td>
</tr>
<tr>
<td>DocPage</td>
<td>Returns the current page number of the current document.</td>
</tr>
<tr>
<td>DocPages</td>
<td>Returns the total number of pages in the current document.</td>
</tr>
<tr>
<td>FrameOverflow</td>
<td>Generates a new page. If called from a Before Script, the current block is the first block on the next page; if called from an After Script, the current block is the last block on the current page.</td>
</tr>
<tr>
<td>NewPage</td>
<td>Generates a new page according to the context in which it is called.</td>
</tr>
</tbody>
</table>

PageIN specific functions

The table below contains a short description of each script function. For information about syntax, examples, etc., see the full description of each script function.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mat</td>
<td>Copies characters from the input matrix. Removes leading and trailing blanks.</td>
</tr>
<tr>
<td>MatBlk</td>
<td>Copies characters from the input matrix. Keeps leading and trailing blanks.</td>
</tr>
<tr>
<td>MaxCol</td>
<td>Returns the number of columns in the input matrix.</td>
</tr>
<tr>
<td>MaxRow</td>
<td>Returns the number of rows in the input matrix.</td>
</tr>
<tr>
<td>PatternResult</td>
<td>Used to check for a pattern in a Message or in a block.</td>
</tr>
</tbody>
</table>
StreamIN specific functions

The table below contains a short description of each script function. For information about syntax, examples, etc., see the full description of each script function.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MatchCol</td>
<td>Returns the content of a specified column. Only available within scripts in StreamIN description files.</td>
</tr>
<tr>
<td>MatchPos</td>
<td>Returns the content of the data from a specified position. Only available within scripts in StreamIN description files.</td>
</tr>
<tr>
<td>Page</td>
<td>Returns the number of the current page.</td>
</tr>
<tr>
<td>PageOfPages</td>
<td>Returns a string specifying the current page number and the total number of pages.</td>
</tr>
<tr>
<td>Pages</td>
<td>Returns the number of pages in the Process.</td>
</tr>
</tbody>
</table>

Ungrouped functions

The table below contains a short description of each script function. For information about syntax, examples, etc., see the full description of each script function.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AtJobEnd</td>
<td>Triggers a specific Event at the end of a job. Typically used to produce a closing page showing the number of invoices printed so far, or the total sum of all invoices.</td>
</tr>
<tr>
<td>CallProc</td>
<td>Invokes a Process in the Message from which it is called.</td>
</tr>
<tr>
<td>CancelJob</td>
<td>Cancels the current job. The cancellation is executed after pre-processing, i.e. all Events are pre-processed before the job is interrupted.</td>
</tr>
<tr>
<td>Clear</td>
<td>Clears the specified variables.</td>
</tr>
<tr>
<td>ClearVars</td>
<td>Clears all variables in a job.</td>
</tr>
<tr>
<td>CreateGlobalSerNo</td>
<td>Creates a serial number key in the repository.</td>
</tr>
<tr>
<td>CurrJobOwner</td>
<td>Returns the current job owner.</td>
</tr>
<tr>
<td>DumpVariables</td>
<td>Writes all variables that have values to a file.</td>
</tr>
<tr>
<td>EndDocument</td>
<td>Forces a Document Begin at the end of the Process from which the script function is called.</td>
</tr>
<tr>
<td>Function name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EndMessage</td>
<td>Used in a Retrieved script to set Event Order Last for the Event.</td>
</tr>
<tr>
<td>Eval</td>
<td>Evaluates all variables in the argument string.</td>
</tr>
<tr>
<td>Execute</td>
<td>Executes an external program, script, bat file or process.</td>
</tr>
<tr>
<td>Exist</td>
<td>Checks if a directory or file exists.</td>
</tr>
<tr>
<td>FlushOutput</td>
<td>Used during the processing of a job. Sends portions of a large output job to the printer, even though processing of the job has not yet been completed.</td>
</tr>
<tr>
<td>ForcePoll</td>
<td>Signals an input connector to poll data.</td>
</tr>
<tr>
<td>GetConnectorValue</td>
<td>Returns the value of a named field from a connector.</td>
</tr>
<tr>
<td>GetCurrModuleinfo</td>
<td>Returns the current Event or Process status, depending on where the script is executed.</td>
</tr>
<tr>
<td>GetGlobalSerNo</td>
<td>Calculates the next counter value in a number sequence according to a specified key stored in the repository.</td>
</tr>
<tr>
<td>GetGlobalSerNoRange</td>
<td>Reserves a specified range of counter values and calculates the next counter value in the number sequence according to a specified key stored in the repository.</td>
</tr>
<tr>
<td>GetHTTPHeaderValue</td>
<td>Returns the value of a named field in the HTTP header of a job created by a request on an HTTP input connector.</td>
</tr>
<tr>
<td>GetMetaDataMessage</td>
<td>Returns the current value of a metadata.</td>
</tr>
<tr>
<td>GetSerNo</td>
<td>Calculates the next counter value in a number sequence according to a specified key stored in a file.</td>
</tr>
<tr>
<td>GetTextFormattingInfo</td>
<td>Enables splitting of LXF text into separate areas, for example to create multi-column output.</td>
</tr>
<tr>
<td>GetXoffs</td>
<td>Returns the current X offset value in millimeters.</td>
</tr>
<tr>
<td>GetYoffs</td>
<td>Returns the current Y offset value in millimeters.</td>
</tr>
<tr>
<td>InConnectorName</td>
<td>Returns the name of the current input connector.</td>
</tr>
<tr>
<td>IncProcStatCounter</td>
<td>Increases the value of the argument with 1.</td>
</tr>
<tr>
<td>NewJob</td>
<td>Replaces the JobBegin and JobEnd functions, and is used to split large input jobs into smaller jobs.</td>
</tr>
<tr>
<td>NoFormFeed</td>
<td>Suppresses the form feed sequence output at end of page.</td>
</tr>
<tr>
<td>NumberOfEvents</td>
<td>Counts the number of Events in a job.</td>
</tr>
<tr>
<td>NumberOfEventsEx</td>
<td>Returns the number of the specified Event.</td>
</tr>
<tr>
<td>Function name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>OutputString</code></td>
<td>Outputs a line of text directly to the output buffer of a StreamOUT or XMLOUT Process.</td>
</tr>
<tr>
<td><code>PreProc</code></td>
<td>Determines whether or not StreamServer is running in pre-process phase.</td>
</tr>
<tr>
<td><code>RGBcolor</code></td>
<td>Specifies an RGB color. For example, the color of a font or an area.</td>
</tr>
<tr>
<td><code>StartTimer</code></td>
<td>Starts a timer in the StreamServer to measure processing time. See also <code>StopTimer</code> on page 330.</td>
</tr>
<tr>
<td><code>StopTimer</code></td>
<td>Stops a timer in the StreamServer, and if required displays the elapsed time in the log and/or writes it to a file. See also <code>StartTimer</code> on page 316.</td>
</tr>
<tr>
<td><code>SetCopies</code></td>
<td>Sets the number of copies of the current Process.</td>
</tr>
<tr>
<td><code>SetDestPath</code></td>
<td>Sets a destination path, and optionally, a destination file.</td>
</tr>
<tr>
<td><code>SetFontProperties</code></td>
<td>Specifies a font. For example, when using a variable alias for font formatting, you use this scripting function to assign a font to the font variable.</td>
</tr>
<tr>
<td><code>SetJobDescr</code></td>
<td>Sets the description of the current job. The job description is used in the log and as the title of spooled job (Windows/NetWare).</td>
</tr>
<tr>
<td><code>SetJobOwner</code></td>
<td>Sets the owner of the print job created by the Spool output connector.</td>
</tr>
<tr>
<td><code>SetLanguage</code></td>
<td>Sets the language to be used for the current job. This setting is used with the StreamServe Language Set (*.sls) file.</td>
</tr>
<tr>
<td><code>SetSerNo</code></td>
<td>Sets the value of a serial number in a number sequence.</td>
</tr>
<tr>
<td><code>Skip</code></td>
<td>Skips the current operation, i.e. the operation to which the script is associated. It can also be used to skip Events, Processes, blocks and fields.</td>
</tr>
<tr>
<td><code>Sort</code></td>
<td>Used in a Before Process script to assign one or more sort keys to a Message. StreamServe sorts the Messages in the job according to the specified sort keys.</td>
</tr>
<tr>
<td><code>Terminate</code></td>
<td>Makes the StreamServer shutdown gracefully when all jobs being processed have been completed, including the one that called the Terminate function.</td>
</tr>
</tbody>
</table>
Check digit calculation functions

The StreamServer supports the Mod10, Mod10L, and Mod11 check digit calculation schemes. Please read the descriptions carefully before you use the functions. Relying only on the function names can sometimes be misleading. For example, Mod11 does not only do a modulo 11 calculation, but uses weight values as well.

Arguments and return values are strings. Arguments can consist of moderately long digit sequences and return values can consist of non-digit values. Currently, the functions do not check that the input string contains digits. Spaces are skipped.

If the input string contains many digits, you should consider using multiple check digits.

Check digit calculation functions

The table below contains a short description of each script function. For information about syntax, examples, etc., see the full description of each script function.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mod10</td>
<td>Performs a Pad 10-based check digit calculation with the alternating weights 2 and 1, starting at the end of the string. The digits of the values are summed. This check digit calculation method is known as Luhn’s algorithm.</td>
</tr>
<tr>
<td>Mod10L</td>
<td>Calculates a length check digit and a Luhn check digit. First the length check digit is calculated (length plus 2, mod 10). Then, to calculate the check digit, the length check digit is added at the end of the string and a pad 10-based check digit calculation (same as for Mod10) is performed. Both the length check digit (first) and the pad 10-based check digit are included in the returned result.</td>
</tr>
<tr>
<td>Mod11</td>
<td>Performs a Pad 11-based check digit calculation with the cycled weights 7, 9, 5, 3, 2, 4, 6, 8, starting at the end of the string. Weighted values are summed. A resulting 0 is replaced by a 5, and a resulting 10 is replaced by a 0.</td>
</tr>
</tbody>
</table>
COM functions can be used to call COM components from StreamServe scripts.

Limitations
- COM functions are only available in Windows environments.
- Only COM components that support the IDispatch interface are supported.
- The following Variant Types are supported:
  - VT_I2
  - VT_I4
  - VT_R4
  - VT_R8
  - VT_CY
  - VT_DATE
  - VT_BSTR
  - VT_BOOL
  - VT_I1
  - VT_UI1
  - VT_I2
  - VT_UI4
  - VT_INT
  - VT_UINT
  - VT_VARIANT

For more information about the IDispatch interface and Variant Types, see [http://msdn.microsoft.com](http://msdn.microsoft.com)

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMCreateObject</td>
<td>Loads the COM component with the specified ProgID and associates it with the specified handle.</td>
</tr>
<tr>
<td>COMDestroyObject</td>
<td>Unloads the COM component associated with the specified handle.</td>
</tr>
<tr>
<td>COMInvoke</td>
<td>Invokes the specified method for the COM component associated with the specified handle.</td>
</tr>
<tr>
<td>COMSetProperty</td>
<td>Sets the specified property for the COM component associated with the specified handle.</td>
</tr>
<tr>
<td>COMGetProperty</td>
<td>Retrieves the value of the specified property for the COM object associated with the specified handle.</td>
</tr>
</tbody>
</table>
Date and time functions

The table below contains a short description of each script function. For information about syntax, examples, etc., see the full description of each script function.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DayOfWeek</strong></td>
<td>Returns the number of the day in a week. Monday returns 1 and Sunday returns 7.</td>
</tr>
<tr>
<td><strong>DayOfYear</strong></td>
<td>Returns the number of the day in a year (1-365) where 1 is January 1st.</td>
</tr>
<tr>
<td><strong>Dformat</strong></td>
<td>Formats a date according to the specified date format. If the date is invalid, no formatting is done, and an empty string is returned.</td>
</tr>
<tr>
<td><strong>DiffDate</strong></td>
<td>Calculates the number of days between two dates.</td>
</tr>
<tr>
<td><strong>DtisoFormat</strong></td>
<td>Converts date and time to ISO 8601 format. (<em>YYYY-MM-DDTHH:MM:SS</em>). The StreamStudio Collector web application requires this date and time format.</td>
</tr>
<tr>
<td><strong>GetDate</strong></td>
<td>Returns the system date in the format <code>yyyymmdd</code>.</td>
</tr>
<tr>
<td><strong>GetDateTime</strong></td>
<td>Returns the system date and time in the format specified by the given picture clause, or in ISO format including milliseconds if empty string is given as argument.</td>
</tr>
<tr>
<td><strong>GetTime</strong></td>
<td>Returns the system time in the format <code>hhmmss</code> (24-hour time format).</td>
</tr>
<tr>
<td><strong>IsDate</strong></td>
<td>Checks if a string expression is a date string.</td>
</tr>
<tr>
<td><strong>NewDate</strong></td>
<td>Calculates a date from a given date and a negative or positive number of days.</td>
</tr>
<tr>
<td><strong>WeekOfYear</strong></td>
<td>Returns the year and the number of a week within the year.</td>
</tr>
</tbody>
</table>
Dispatcher functions

The table below contains a short description of each script function. For information about syntax, examples, etc., see the full description of each script function.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetDistributionChannelsForRole</td>
<td>Returns the distribution channels for a role, for a specific document type.</td>
</tr>
<tr>
<td>GetDistributionChannelsForUser</td>
<td>Returns the distribution channels for a user, for a specific document type.</td>
</tr>
<tr>
<td>GetRolesForUser</td>
<td>Returns the roles associated with a user.</td>
</tr>
</tbody>
</table>
## Document sorting and bundling functions

The table below contains a short description of each script function. For information about syntax, examples, etc., see the full description of each script function.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectorPageActual</td>
<td>Returns the number of logical pages sent to the output connector, including inserts but excluding skipped pages.</td>
</tr>
<tr>
<td>CurrDocProperty</td>
<td>Returns the specified document property for the current document.</td>
</tr>
<tr>
<td>CurrJobProperty</td>
<td>Returns the specified job property.</td>
</tr>
<tr>
<td>CurrMetadata</td>
<td>Returns the value of the specified metadata for the current document.</td>
</tr>
<tr>
<td>CurrReposProperty</td>
<td>Returns the specified repository property.</td>
</tr>
<tr>
<td>DeclareMetadata</td>
<td>Declares metadata that can be used in post-processor scripting. If the specified metadata exists in the post-processor repository, the metadata is retrieved with the documents when they are retrieved from the repository.</td>
</tr>
<tr>
<td>DocInsertedPages</td>
<td>Returns the number of pages inserted in the output queue within the current document.</td>
</tr>
<tr>
<td>DocPage</td>
<td>Returns the current page number of the current document.</td>
</tr>
<tr>
<td>DocPageActual</td>
<td>Returns the number of document pages sent to the output connector, including inserts but excluding skipped pages.</td>
</tr>
<tr>
<td>EnableEnvelopeCheck</td>
<td>Enables or disables a check for an inconsistent number of sheets when using enveloping.</td>
</tr>
<tr>
<td>GetCurrSegDoc</td>
<td>Returns an index that identifies the current document, in the current segment, in the set of documents sent to the output connector.</td>
</tr>
<tr>
<td>GetDocActualInserts</td>
<td>Returns the inserts assigned to the document.</td>
</tr>
<tr>
<td>GetDocEnvelopeCount</td>
<td>Returns the number of envelopes for the current document.</td>
</tr>
<tr>
<td>GetDocInsertCount</td>
<td>Returns the number of inserts for the current document.</td>
</tr>
<tr>
<td>GetDocInsertEquival</td>
<td>Returns the insert equivalent count for the current document.</td>
</tr>
<tr>
<td>GetDocSheetCount</td>
<td>Returns the number of sheets within the current document.</td>
</tr>
<tr>
<td>GetEnvelNr</td>
<td>Returns the envelope number of the current document.</td>
</tr>
<tr>
<td>GetFirstPPDoc</td>
<td>Returns a counter index that identifies the first document, in a set of documents sent to the output connector in a Post-processor job.</td>
</tr>
<tr>
<td>GetFirstSegDoc</td>
<td>Returns the index that identifies the first document in the current segment, in the set of documents sent to the output connector.</td>
</tr>
<tr>
<td>Function name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GetMailMachine</td>
<td>Returns the name of the enveloping machine to which the current document was assigned.</td>
</tr>
<tr>
<td>GetNextPPDoc</td>
<td>Returns a counter index that identifies the next document in a set of documents in the post-processor job sent to the output connector.</td>
</tr>
<tr>
<td>GetNextSegDoc</td>
<td>Returns an index that identifies the next document, in the current segment, in the set of documents sent to the output connector.</td>
</tr>
<tr>
<td>GetPPDocId</td>
<td>Returns the document ID of a document stored in a Post-processor repository.</td>
</tr>
<tr>
<td>GetPPDocProperty</td>
<td>Returns the value of the specified property for the specified document.</td>
</tr>
<tr>
<td>GetPPError</td>
<td>Returns the last error that occurred during the Post-processor job.</td>
</tr>
<tr>
<td>GetPPJobId</td>
<td>Returns the job ID of a job stored in a Post-processor repository.</td>
</tr>
<tr>
<td>GetPPJobProperty</td>
<td>Returns the value of the specified property for the job containing the specified document.</td>
</tr>
<tr>
<td>GetPPMetadata</td>
<td>Returns the value of the specified metadata for the specified document.</td>
</tr>
<tr>
<td>GetPPReposProperty</td>
<td>Returns the value of the specified repository property storing the specified document.</td>
</tr>
<tr>
<td>GetSegDocProperty</td>
<td>Returns the specified document property for the specified document.</td>
</tr>
<tr>
<td>GetSegJobProperty</td>
<td>Returns the specified job property for the specified document.</td>
</tr>
<tr>
<td>GetSegment</td>
<td>Returns the current segment number. This is the same number that replaces the %{SEGMENT} variable in the output file name.</td>
</tr>
<tr>
<td>GetSegMetadata</td>
<td>Returns the value of the specified metadata for the document in the specified segment.</td>
</tr>
<tr>
<td>GetSegReposProperty</td>
<td>Returns the specified repository property for the specified document.</td>
</tr>
<tr>
<td>InsertOverlay</td>
<td>Adds an overlay to the current page.</td>
</tr>
<tr>
<td>InsertPage</td>
<td>Inserts a new page in the output stream.</td>
</tr>
<tr>
<td>IsFirstDocInEnvelope</td>
<td>Checks if the current document is the first document in the envelope.</td>
</tr>
<tr>
<td>IsFirstPageInEnvelope</td>
<td>Checks if the current page is the first page in the envelope.</td>
</tr>
<tr>
<td>IsFirstSegment</td>
<td>Checks if the current segment is the first segment of post-processor job.</td>
</tr>
<tr>
<td>IsInsertingPage</td>
<td>Checks if the current page is being inserted to the output queue.</td>
</tr>
<tr>
<td>IsLastDocInEnvelope</td>
<td>Checks if the current document is the last document in the envelope.</td>
</tr>
<tr>
<td>IsLastPageInEnvelope</td>
<td>Checks if the current page is the last page in the envelope.</td>
</tr>
<tr>
<td>IsLastSegment</td>
<td>Checks if the current segment is the last segment of post-processor job.</td>
</tr>
<tr>
<td>Function Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>JobInsertedPages</td>
<td>Returns the number of pages inserted to the output queue within the current job.</td>
</tr>
<tr>
<td>NextDoc</td>
<td>Skips the current document and processes the next document.</td>
</tr>
<tr>
<td>NextPage</td>
<td>Skips the current page and processes the next page.</td>
</tr>
<tr>
<td>NextProc</td>
<td>Skips the rest of the current Process, and processes the next Process.</td>
</tr>
<tr>
<td>NextSegment</td>
<td>Skips the rest of the current segment, and processes the next segment.</td>
</tr>
<tr>
<td>PPDocCount</td>
<td>Returns the number of documents sent to the output connector.</td>
</tr>
<tr>
<td>SaveCurrMetadata</td>
<td>Saves the metadata for the current document in the repository.</td>
</tr>
<tr>
<td>SavePPMetadata</td>
<td>Saves the metadata for the document specified in the <code>doc_handle</code> parameter returned by either the <code>GetFirstPPDoc</code> or <code>GetNextPPDoc</code> function.</td>
</tr>
<tr>
<td>SaveSegMetadata</td>
<td>Saves the metadata for the document specified in the <code>doc_handle</code> parameter returned by either the <code>GetFirstSegDoc</code>, <code>GetCurrSegDoc</code>, or <code>GetNextSegDoc</code> function.</td>
</tr>
<tr>
<td>SetCurrMetadata</td>
<td>Sets string metadata for the current document.</td>
</tr>
<tr>
<td>SetCurrMetadataNum</td>
<td>Sets numeric metadata for the current document.</td>
</tr>
<tr>
<td>SetSegMetadata</td>
<td>Sets string metadata for the document specified in the <code>doc_handle</code> parameter returned by either the <code>GetFirstSegDoc</code>, <code>GetCurrSegDoc</code>, or <code>GetNextSegDoc</code> function.</td>
</tr>
<tr>
<td>SetSegMetadataNum</td>
<td>Sets numeric metadata for the document specified in the <code>doc_handle</code> parameter returned by either the <code>GetFirstSegDoc</code>, <code>GetCurrSegDoc</code>, or <code>GetNextSegDoc</code> function.</td>
</tr>
<tr>
<td>SortSegDoc</td>
<td>Sorts the documents in the current segment according to the specified sort keys.</td>
</tr>
<tr>
<td>UpdateDocStatus</td>
<td>Updates status of the current document in the repository.</td>
</tr>
<tr>
<td>UpdateJobStatus</td>
<td>Updates the status of the current post-processor job in the Post-processor repository.</td>
</tr>
<tr>
<td>UpdatePPDocStatus</td>
<td>Updates repository status of the document specified in the <code>doc_handle</code> parameter returned by either the <code>GetFirstPPDoc</code> or <code>GetNextPPDoc</code> function.</td>
</tr>
<tr>
<td>UpdatePPJobStatus</td>
<td>Updates status of the current post-processor job in the repository.</td>
</tr>
<tr>
<td>UpdateSegDocStatus</td>
<td>Updates repository status of the document specified in the <code>doc_handle</code> parameter returned by either the <code>GetFirstSegDoc</code>, <code>GetCurrSegDoc</code>, or <code>GetNextSegDoc</code> function.</td>
</tr>
<tr>
<td><strong>Function name</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td><em>UpdateSegJobStatus</em></td>
<td>Updates repository status of the post-processor job that contains the document specified in the <code>doc_handle</code> parameter returned by either the <code>GetFirstSegDoc</code>, <code>GetCurrSegDoc</code>, or <code>GetNextSegDoc</code> function.</td>
</tr>
</tbody>
</table>
Email functions

The table below contains a short description of each script function. For information about syntax, examples, etc., see the full description of each script function.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AttachmentBegin</td>
<td>This script function is normally called from a Before Process script on the first Process to include in an attachment.</td>
</tr>
<tr>
<td>AttachmentEnd</td>
<td>This script function is called from an After Process script on the last Process. It stops the AttachmentBegin function.</td>
</tr>
<tr>
<td>GetAttachmentFile</td>
<td>Returns the file name of an attachment saved to disk via an EmailIN connector.</td>
</tr>
<tr>
<td>GetAttachmentCount</td>
<td>Returns the number of attachments in the current email that have been saved to disk via an EmailIN connector.</td>
</tr>
<tr>
<td>GetAttachmentOriginalFile</td>
<td>Returns the original file name of an attachment saved to disk via an EmailIN connector.</td>
</tr>
<tr>
<td>GetAttachmentContentEncoding</td>
<td>Returns the encoding of an attachment saved to disk via an EmailIN connector.</td>
</tr>
<tr>
<td>GetAttachmentContentType</td>
<td>Returns the content-type of an attachment saved to disk via an EmailIN connector.</td>
</tr>
</tbody>
</table>
File handling functions

The table below contains a short description of each script function. For information about syntax, examples, etc., see the full description of each script function.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base64DecodeFile</td>
<td>Decodes a base64 encoded file and stores the result in an output file.</td>
</tr>
<tr>
<td>Base64EncodeFile</td>
<td>Encodes an input file to base64 format. The encoded data is stored in an output file.</td>
</tr>
<tr>
<td>CurrInFileName</td>
<td>Returns the name of the input file, if there is one, else an empty string.</td>
</tr>
<tr>
<td>CurrInRealPath</td>
<td>Returns the temporary file name of the input file, if there is one.</td>
</tr>
<tr>
<td>CurrInSavePath</td>
<td>Input files retrieved via a Directory input connector can be copied to backup directory. All files are given unique names when copied to the backup directory. The CurrInSavePath function is used to retrieve the new file names.</td>
</tr>
<tr>
<td>FileClose</td>
<td>Closes the specified file.</td>
</tr>
<tr>
<td>FileCopy</td>
<td>Copies the specified file, and leaves the original file unchanged.</td>
</tr>
<tr>
<td>FileDelete</td>
<td>Deletes the specified file.</td>
</tr>
<tr>
<td>FileMove</td>
<td>Moves the specified file from one location to another.</td>
</tr>
<tr>
<td>FileOpen</td>
<td>Opens the specified file for reading or writing according to mode specified.</td>
</tr>
<tr>
<td>FileReadLn</td>
<td>Reads a line from the specified file until a new line or an end of file is reached, and returns the line to a specified variable.</td>
</tr>
<tr>
<td>FileSize</td>
<td>Returns the size of the specified file.</td>
</tr>
<tr>
<td>FileWrite</td>
<td>Writes data to the specified file.</td>
</tr>
<tr>
<td>FileWriteLn</td>
<td>Writes data to the specified file, and adds a new line character at the end. You must open the file with FileOpen before calling FileWriteLn, and use the FileClose function to close the file.</td>
</tr>
<tr>
<td>IoErrText</td>
<td>Returns the error text for IO functions.</td>
</tr>
<tr>
<td>MkDir</td>
<td>Creates a new directory using the specified path.</td>
</tr>
<tr>
<td>OutputFile</td>
<td>Used in a Process to send the contents of a file directly to an output connector, without changing or modifying the contents. It can also be used for concatenating or encapsulating already existing data.</td>
</tr>
<tr>
<td>URLClose</td>
<td>Closes the file at the specified URL.</td>
</tr>
<tr>
<td>URLExist</td>
<td>Checks if a file exists at the specified URL.</td>
</tr>
<tr>
<td>URLOpen</td>
<td>Opens a file at the specified URL for reading.</td>
</tr>
</tbody>
</table>
### Function name: URLReadLn

**Description**: Reads a line from a file at the specified URL until a new line or an end of file is reached, and returns the line to a specified variable.
LDAP functions

There are a number of built-in LDAP script functions that can be used to connect to a directory server, retrieve information from a user directory, add entries to a user directory, etc..

![Diagram](https://via.placeholder.com/150)

**Figure 3  Connection to a directory server.**

**Escape sequences**

The less-than sign (<) is a reserved character and must be escaped if used in a string. See *Escape sequences* on page 40.

**LDAP functions**

The table below contains a short description of each script function. For information about syntax, examples, etc., see the full description of each script function.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LdapAddAttrBinaryValue</td>
<td>Takes the entryID (returned by the LdapGetEntry function), the attribute name, and a filename, and adds the contents of the file to the given attribute.</td>
</tr>
<tr>
<td>LdapAddAttrValue</td>
<td>Adds the specified value to the specified attribute. The changes are not written to the directory until the LdapUpdateEntry function is called.</td>
</tr>
<tr>
<td>LdapConnect</td>
<td>Connects to the directory server.</td>
</tr>
<tr>
<td>LdapConnectPersistent</td>
<td>Connects to the directory server.</td>
</tr>
<tr>
<td>LdapConnectSSL</td>
<td>Connects to the directory server using an encrypted connection.</td>
</tr>
<tr>
<td>Function name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LdapConnectSSLCCA</td>
<td>Connects to the directory server using an encrypted connection. A Client Certificate Authentication is provided instead of user/password authentication.</td>
</tr>
<tr>
<td>LdapConnectSSLCCAPersistent</td>
<td>Connects to the directory server using an encrypted connection. A Client Certificate Authentication is provided instead of user/password authentication.</td>
</tr>
<tr>
<td>LdapConnectSSLPersistent</td>
<td>Connects to the directory server using an encrypted connection.</td>
</tr>
<tr>
<td>LdapDisconnect</td>
<td>Disconnects the from the directory server.</td>
</tr>
<tr>
<td>LdapFind</td>
<td>Finds entries in the LDAP directory using an LDAP query.</td>
</tr>
<tr>
<td>LdapGetAttrValue</td>
<td>Retrieves a value from a specific attribute on an entry.</td>
</tr>
<tr>
<td>LdapGetAttrValueCount</td>
<td>Returns the number of values of a specific attribute on an entry.</td>
</tr>
<tr>
<td>LdapGetEntry</td>
<td>Retrieves a specific entry from a result set (the result of a query).</td>
</tr>
<tr>
<td>LdapGetEntryCount</td>
<td>Returns the number of entries associated with a resultID.</td>
</tr>
<tr>
<td>LdapGetObjectByDN</td>
<td>Searches the LDAP directory for a specific object.</td>
</tr>
<tr>
<td>LdapGetUser</td>
<td>Searches the LDAP directory for a specific user.</td>
</tr>
<tr>
<td>LdapNewEntry</td>
<td>Creates a new entry in the LDAP directory. Changes are not written to the directory until the LdapUpdateEntry function is called.</td>
</tr>
<tr>
<td>LdapReleaseNewEntries</td>
<td>To decrease the memory usage, you can call this function which releases memory allocated for all new entries created by LdapNewEntry function calls.</td>
</tr>
<tr>
<td>LdapReleaseResultSet</td>
<td>To decrease the memory usage, you can call this function which frees result sets that are returned from ldapFind, ldapGetUser, and ldapGetObjectByDN function calls.</td>
</tr>
<tr>
<td>LdapReplaceAttrBinaryValue</td>
<td>Takes the entryID (as returned by LdapGetEntry), the attribute name, and a filename, and adds the contents of the file to the given attribute.</td>
</tr>
<tr>
<td>LdapReplaceAttrValue</td>
<td>Replaces the value(s) of an attribute with another value.</td>
</tr>
<tr>
<td>LdapSort</td>
<td>Sorts entries in a result set based on one or more entry attributes.</td>
</tr>
<tr>
<td>LdapUpdateEntry</td>
<td>Writes any changes that have been made to an entry back to the LDAP server.</td>
</tr>
</tbody>
</table>
Connecting to the directory server

Before you retrieve data from, or add data to, the user directory, you must connect to the directory server. You can use different LDAP script functions for this purpose.

Non-persistent connections

You can use the following script functions to establish a non-persistent connection to the directory server:

- LdapConnect
- LdapConnectSsl
- LdapConnectSslCca

The connection stays open until the job ends. You can also use the LdapDisconnect script function to disconnect from the directory server.

Non-encrypted connections

The LdapConnect function is used when the connection is not encrypted. The syntax is as follows:

```plaintext
LdapConnect(connName, hostName, user, password)
```

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A name for the connection.</td>
</tr>
<tr>
<td>The host to connect to. The</td>
</tr>
<tr>
<td>default port is 389, but can</td>
</tr>
<tr>
<td>be overridden using hostName:port.</td>
</tr>
<tr>
<td>The user ID to log on to the</td>
</tr>
<tr>
<td>server.</td>
</tr>
<tr>
<td>The password to log on to the</td>
</tr>
<tr>
<td>server.</td>
</tr>
</tbody>
</table>

Leaving the user and password fields blank implies an anonymous logon. This normally results in a limited read only access to the user directory.

Encrypted connections

The LdapConnectSsl and LdapConnectSslCca are used when the connection is encrypted. LdapConnectSsl uses username and password for authentication, and LdapConnectSslCca uses a client certificate for authentication. See the online help for more information on these script functions.
Persistent connections

You can use the following script functions to establish a persistent connection to the directory server:

- `LdapConnectPersistent`
- `LdapConnectSslPersistent`
- `LdapConnectSslCcaPersistent`

The connection stays open until you call the `LdapDisconnect` script function. The syntax and functionality are the same as for the script function used for non-persistent connections.

LDAP examples

The following examples show how the LDAP functions can be used:

**Example 39  Reading a user profile (scenario 1)**

The script reads a user profile, and queries the fax number of the user.

```plaintext
// Connect to the LDAP server on localhost. Give the name "con" to the connection
ldapconnect("con", "localhost", "","");
// Retrieve information about the user specified in $ourref
$theuser = ldapgetuser("con", "dc=streamserve,dc=com",$ourref, "", "");
/* Retrieve the sn and givenname attributes of the user. Concatenate the two values with a comma in between, and store the string in $refNam.*/
$refNam = ldapgetattrvalue($theuser, "sn", 0) +", " +
ldapgetattrvalue($theuser, "givenname", 0);
// Retrieve the user’s fax number and store it in $ourfax.
$ourfax = ldapgetattrvalue($theuser,"facsimiletelephonenumber", 0);
/* Here you either proceed with further queries or call LdapDisConnect to terminate the connection to the LDAP server.*/
```
Example 40  Reading a user profile (scenario 2)

This script identifies members of a group, and queries the attributes (email address and common name) of each member.

```perl
/* Connect to the LDAP server on localhost. Give the name "con" to the connection. The username used is "Directory Manager" and password is "streamserve"*/
ldapconnect("con", "localhost", "cn=Directory Manager", "streamserve");
/* Find the group entry $ourgroup. The first argument is the connection ID, the second specifies the root of the search tree. The third argument is the common name to search for.*/
$grouplist = ldapfind("con", "ou=Groups, dc=streamserve,dc=com", "cn=" + $ourgroup);
// Get the first entry (0 based index).
$thegroup = ldapgetentry($grouplist, 0);
// Get the number of values associated to attribute "uniquemember"
$iterator = ldapgetattrvaluecount($thegroup, "uniquemember") - 1;
// For each value of the attribute (each member of the group), do...
while (num($iterator) > -1)
{
    /* The values of the "uniquemember" attribute are distinguished names of different users who are members of the group. Fetch them one by one*/
    $groupmember = ldapgetobjectbyname("con", ldapgetattrvalue($thegroup, "uniquemember", Num($iterator)));
    /* Get the value of the "mail" attribute for the current group member. Store it in $email*/
    $email = ldapgetattrvalue($groupmember, "mail", 0);
    // Get another of the users attributes...
    $commonname = ldapgetattrvalue($groupmember, "cn", 0);
    // Decrement $iterator to get the next group member.
    $iterator -= 1;
    // Send it to the block
    CallBlock("Free_Block");
}
/* Here you either proceed with further queries or call LdapDisConnect to terminate the connection to the LDAP server.*/
```
Example 41  

Reading a user profile (scenario 3)

The following example is a variation of Example 40 on page 72. This script identifies members of a group, and queries the attributes (email address) of each member. It then creates a string with concatenated email addresses for a mass mailing (an email is sent to every member of this group).

/* Connect to the LDAP server on localhost. Give the name "con" to the connection. The username is "Directory Manager" and password "streamserve". */
$ourrefs="";
$ourgroup = "Administrator Group";
ldapconnect("con", "localhost", "cn=Directory Manager","streamserve");
/* Find the group entry $ourgroup. The first argument is the connection ID, the second specifies the root of the search tree. The third argument is the common name to search for. */
$grouplist = ldapfind("con", "ou=StreamServe Groups,dc=streamserve,dc=com", "(cn=" + $ourgroup + ")");
// Get the first entry (0 based index).
$theuser = ldapgetentry($grouplist, 0);
// Get the number of values associated to attribute "uniquemember"
$iterator = ldapgetattrvaluecount($theuser, "uniquemember")-1;
log(0,"iterator="+$iterator);
// For each value of the attribute (each member of the group), do...
while (num($iterator) >-1)
{
    /* The values of the "uniquemember" attribute are distinguished names of different users who are members of the group. Fetch them one by one. */
    $groupuser = ldapgetobjectbydn("con", ldapgetattrvalue($theuser, "uniquemember", num($iterator)) );
    /* Get the value of the "mail" attribute for the current group member. Concatenate the value (e-mail address) with a semicolon between the e-mail addresses of different users and store the string in $ourrefs. */
    $ourrefs = $ourrefs + ldapgetattrvalue($groupuser, "mail", 0) + ";";
    // Decrement $iterator to get the next group member.
    $iterator -= 1;
}
log(0,"ourref="+$ourrefs);
/* Here you either proceed with further queries or call LdapDisConnect to terminate the connection to the LDAP server. */
Example 42  Retrieving and updating attributes

This script retrieves one attribute (fax number) and updates another attribute (email address).

/* Connect to the LDAP server on localhost. Give the name "con" to the
connection. The username used is "Directory Manager" and password is
"dmanager"*/
ldapconnect("con", "localhost", "cn=Directory Manager", "dmanager");
/* Find the user entry $ourref. The first argument is the connection ID, the
second specifies the root of the search tree. The third argument is the common
name to search for.*/
$theuserlist = ldapfind("con", "dc=streamserve,dc=com", "cn=" + $ourref);
// Find the first (and hopefully only) entry in the result.
$theuser = ldapgetentry($theuserlist, 0);
// Get the user's fax number
$ourfax = ldapgetattrvalue($theuser, "facsimiletelephonenumber", 0);
// Update the e-mail address
ldapreplaceattrvalue($theuser, "mail", $ourref_email);
//Write the changes made to $theuser back to the LDAP server.
ldapupdateentry($theuser);
/* Here you either proceed with further queries or call LdapDisConnect to
terminate the connection to the LDAP server.*/
Lotus Notes functions

The Lotus Notes script functions can be used to retrieve data from a Lotus Notes database to the StreamServer, and to update the Lotus Notes database.

You can retrieve data directly to a Process from a Lotus Notes database at runtime, for example an address or a phone number. You use the Lotus Notes Fetch Wizard to generate a code block in a script in the Process in which you want to include the data.

Lotus Notes functions

The table below contains a short description of each script function. For information about syntax, examples, etc., see the full description of each script function.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LotusNotesAttachFile</td>
<td>Attaches a file to the current Lotus Notes Note. If a file already is</td>
</tr>
<tr>
<td></td>
<td>attached to the Lotus Notes Note it is replaced by the new file.</td>
</tr>
<tr>
<td>LotusNotesAddNote</td>
<td>Adds a Lotus Notes Note to a Lotus Notes database.</td>
</tr>
<tr>
<td>LotusNotesConnect</td>
<td>Establishes a connection to a Lotus Notes database on a Lotus Domino server.</td>
</tr>
<tr>
<td>LotusNotesGet</td>
<td>Retrieves data from a field in a Lotus Notes Note.</td>
</tr>
<tr>
<td>LotusNotesDisconnect</td>
<td>Disconnects an existing connection to a Lotus Notes database.</td>
</tr>
<tr>
<td>LotusNotesFind</td>
<td>Searches the database for Lotus Notes Notes that match the specified search</td>
</tr>
<tr>
<td></td>
<td>criteria.</td>
</tr>
<tr>
<td>LotusNotesFirst</td>
<td>If the LotusNotesFind function finds more than one Lotus Notes Note that</td>
</tr>
<tr>
<td></td>
<td>matches the search criteria it creates a list of Lotus Notes Notes. The</td>
</tr>
<tr>
<td></td>
<td>LotusNotesFirst function can be used to return the first Lotus Notes Note in</td>
</tr>
<tr>
<td></td>
<td>the list.</td>
</tr>
<tr>
<td>LotusNotesLast</td>
<td>If the LotusNotesFind function finds more than one Lotus Notes Note that</td>
</tr>
<tr>
<td></td>
<td>matches the search criteria it creates a list of Lotus Notes Notes. The</td>
</tr>
<tr>
<td></td>
<td>LotusNotesLast function can be used to return the last Lotus Notes Note in</td>
</tr>
<tr>
<td></td>
<td>the list.</td>
</tr>
<tr>
<td>LotusNotesNext</td>
<td>If the LotusNotesFind function finds more than one Note that matches the</td>
</tr>
<tr>
<td></td>
<td>search criteria it creates a list of Lotus Notes Notes. The LotusNotesNext</td>
</tr>
<tr>
<td></td>
<td>function can be used to return the next Lotus Notes Note in the list.</td>
</tr>
<tr>
<td>LotusNotesPrev</td>
<td>If the LotusNotesFind function finds more than one Note that matches the</td>
</tr>
<tr>
<td></td>
<td>search criteria it creates a list of Lotus Notes Notes. The LotusNotesPrev</td>
</tr>
<tr>
<td></td>
<td>function can be used to return the previous Lotus Notes Note in the list.</td>
</tr>
</tbody>
</table>
### Lotus Notes functions

**Script functions reference**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LotusNotesSetDateTime</td>
<td>Sets the value of an item in a Note to the date and, optionally, the time specified.</td>
</tr>
<tr>
<td>LotusNotesSetNumber</td>
<td>Sets the value of an item in a Lotus Notes Note to the specified value.</td>
</tr>
<tr>
<td>LotusNotesSetText</td>
<td>Updates an item in a Lotus Notes Note with the specified text.</td>
</tr>
</tbody>
</table>

### Lotus Notes examples

**Example 43  Connecting to a Notes database and attaching a document to a file**

This script is used to connect to a Notes database and attach a document to a file.

```plaintext
// Domino Server
$server_name = "StuNt01/streamserve";
// The Lotus Notes database where you want to create a new document
$database_file = "AGFxax.nsf";
// Your password
$password = "streamserve";
// Create a connection to Lotus Notes
LotusNotesConnect($server_name, $database_file, $password);
// Create a new document in the database.
LotusNotesAddNote();
// Set the field NFaxno to 123456789 in the new document.
LotusNotesSetText("NFaxno", "123456789");
// This will only be done once, i.e in Runtime mode not in Preproc
if (preproc()=0) {
    $ret = LotusNotesAttachFile("C:\Files\My_file.pdf", "True");
}
LotusNotesdisconnect();
```
Message traversing functions

Message traversing script functions enable you to retrieve field values directly from the Message, and include the value via a variable in a Process. For example, you can use Message traversing script functions to search for a field in a block, and add the field value at root level to a PageOUT Process. You cannot do this by just using field variables defined in the Event.

What is Message traversing?
Message traversing means you can walk through (i.e. traverse) the Message tree, and locate the appropriate blocks and fields. To locate a field, you do not have to explicitly select the field. You can specify where in the Message tree to start searching, and use a search string that points to the field.

Functions traversing the entire Message
You can use script functions that traverses the entire Message to locate blocks and fields. The table below contains a short description of each script function. For information about syntax, examples, etc., see the full description of each script function.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MsgCountId</td>
<td>Counts the number of instances of the specified block in the Message.</td>
</tr>
<tr>
<td>MsgGetValue</td>
<td>Returns the value of the specified field.</td>
</tr>
<tr>
<td>MsgValueExist</td>
<td>Checks if the specified field contains a value.</td>
</tr>
</tbody>
</table>

Functions traversing the current Process
You can use script functions that traverses the current Process (not applicable to PageOUT) to locate blocks and fields. The table below contains a short description of each script function. For information about syntax, examples, etc., see the full description of each script function.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MsgProcCountId</td>
<td>Counts the number of instances of the specified block in the current Process.</td>
</tr>
<tr>
<td>MsgProcGetValue</td>
<td>Returns the value of a specified field within the current Process.</td>
</tr>
<tr>
<td>MsgProcValueExist</td>
<td>Checks if a specific field exists in the current Process.</td>
</tr>
</tbody>
</table>
Functions traversing the current frame

You can use script functions that traverses the current frame (only applicable to PageOUT) to locate blocks and fields. The table below contains a short description of each script function. For information about syntax, examples, etc., see the full description of each script function.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MsgFrameCountId</td>
<td>Counts the number of instances of the specified block in the current frame.</td>
</tr>
<tr>
<td>MsgFrameGetValue</td>
<td>Returns the value of the specified field.</td>
</tr>
<tr>
<td>MsgFrameValueExist</td>
<td>Checks if the specified field contains a value.</td>
</tr>
</tbody>
</table>

Functions traversing sections of a Message

You can use script functions that traverses sections of the Message. A script function (MsgOpen, MsgProcOpen, or MsgFrameOpen) defines where to start traversing the Message, and how many levels in the Message tree to traverse. This script function returns a handle that identifies the specific traversing pass. The handle is then used as argument in other script functions that locate blocks and fields in the specified section of the Message. The table below contains a short description of each script function. For information about syntax, examples, etc., see the full description of each script function.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MsgOpen</td>
<td>Defines where to start traversing the Message.</td>
</tr>
<tr>
<td>MsgProcOpen</td>
<td>Defines where in the current Process to start traversing the Message.</td>
</tr>
<tr>
<td></td>
<td>This function cannot be used in PageOUT.</td>
</tr>
<tr>
<td>MsgFrameOpen</td>
<td>Defines where in the current frame to start traversing the Message.</td>
</tr>
<tr>
<td></td>
<td>This function can only be used in PageOUT.</td>
</tr>
<tr>
<td>MsgBlockId</td>
<td>Returns the name of the current block.</td>
</tr>
<tr>
<td>MsgNumFields</td>
<td>Returns the number of fields in the current block.</td>
</tr>
<tr>
<td>MsgFieldId</td>
<td>Returns the name of a field in the current block.</td>
</tr>
<tr>
<td>MsgFieldValue</td>
<td>Returns the value of a field in the current block.</td>
</tr>
<tr>
<td>MsgNextBlock</td>
<td>Moves to the next block.</td>
</tr>
<tr>
<td>MsgClose</td>
<td>Closes the Message after it has been traversed.</td>
</tr>
</tbody>
</table>
Message traversing examples

Example 44   Message traversing example

The script in this example browses a frame and retrieving the values of the field "ManualRow_Type" from all blocks within the frame.

```plaintext
$handle=MsgFrameOpen("", 0);
$returnValue=MsgNextBlock($handle);
//Get value from sub-block and check if correct blockname from XML
$block_name=MsgBlockId($handle);
if($block_name="Order_Manuell_row")
{
    //Get number of fields in Order_Manuell_row block and initiate counter.
    $NoOfblockFields=MsgNumFields($handle);
    $NoOfBlockFields_I=1;
    //find field "ManualRow_Type" and extract value
    while (num($NoOfblockFields_I)<=num($NoOfBlockFields))
    {
        $field_id=MsgFieldId($handle, $NoOfblockFields_I);
        if($field_id="ManualRow_type")
        {
            //Get value from field "ManualRow_type"
            $Type_value=MsgFieldValue($handle, $NoOfBlockFields_I);
        }
        $NoOfBlockFields_I++;
    }
    //Check that the same block is examined
    while(num($returnValue)!=0 and $block_name!="Order_Row")
    {
        //Return value if next OrderRow or all OrderRows examined=0
        $returnValue=MsgNextBlock($handle);
        //Get value from sub-block and check if correct block name from XML
        $block_name=MsgBlockID($handle);
        if($block_name="Order_Manuell_row")
        {
            //Get number of fields in Order_Manuell_row block and initiate counter.
            $NoOfblockFields=MsgNumFields($handle);
            $NoOfBlockFields_I=1;
            //find field "ManualRow_Type" and extract value
            while (num($NoOfblockFields_I)<=num($NoOfBlockFields))
            {
                $field_id=MsgFieldId($handle, $NoOfblockFields_I);
                if($field_id="ManualRow_type")
                {
                    //Get value from field "ManualRow_type"
                    $Type_value=MsgFieldValue($handle, $NoOfBlockFields_I);
                }
                $NoOfBlockFields_I++;
            }
        }
    }
}
```

The table below contains a short description of each script function. For information about syntax, examples, etc., see the full description of each script function.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Abs</em></td>
<td>Returns the absolute value of a number.</td>
</tr>
<tr>
<td><em>BinAnd</em></td>
<td>Performs an AND operation on the binary representation of the complement integer values of the two arguments.</td>
</tr>
<tr>
<td><em>BinOr</em></td>
<td>Performs an OR operation on the binary representation of the complement integer values of the two arguments.</td>
</tr>
<tr>
<td><em>BinXOr</em></td>
<td>Returns the exclusive OR on the binary representation of the complement integer values of the two arguments.</td>
</tr>
<tr>
<td><em>Div</em></td>
<td>Divides the arguments and returns the floor of the result of that division.</td>
</tr>
<tr>
<td><em>In2Mm</em></td>
<td>Returns inches to millimeter conversion (as a floating point value) of the argument.</td>
</tr>
<tr>
<td><em>In2Pt</em></td>
<td>Returns inches to millimeter conversion (as a floating point value) of the argument. A point is 1/72 of an inch.</td>
</tr>
<tr>
<td><em>Int</em></td>
<td>Returns truncation-to-integer (as a floating point value) of the argument.</td>
</tr>
<tr>
<td><em>Mm2In</em></td>
<td>Returns millimeters to inches conversion (as a floating point value) of the argument.</td>
</tr>
<tr>
<td><em>Mm2Pt</em></td>
<td>Returns millimeters to points conversion (as a floating point value) of the argument.</td>
</tr>
<tr>
<td><em>Mod</em></td>
<td>Returns a number that is the modulus of the arguments.</td>
</tr>
<tr>
<td><em>Pt2In</em></td>
<td>Returns points to inches conversion (as a floating point value) of the argument. A point is 1/72 of an inch.</td>
</tr>
<tr>
<td><em>Pt2Mm</em></td>
<td>Returns points to millimeters conversion (as a floating point value) of the argument.</td>
</tr>
<tr>
<td><em>Round</em></td>
<td>Returns the first argument rounded to the number of decimals specified in the second argument.</td>
</tr>
</tbody>
</table>
Position formatting functions

The table below contains a short description of each script function. For information about syntax, examples, etc., see the full description of each script function.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetCurrX</td>
<td>Returns the current position of a PageOUT object along the X axis in millimeters.</td>
</tr>
<tr>
<td>GetCurrBlockY</td>
<td>Returns the current position of a PageOUT block along the Y axis (i.e. the position of the upper left corner of the block) in millimeters.</td>
</tr>
<tr>
<td>GetCurrObjY</td>
<td>Returns the current position of a PageOUT object along the Y axis (i.e. the position of the upper left corner of the object) in millimeters.</td>
</tr>
<tr>
<td>SetCurrX</td>
<td>Sets a new X position for an object in PageOUT, such as a field, line, barcode, or text.</td>
</tr>
<tr>
<td>SetCurrY</td>
<td>Sets a new Y position for an object in PageOUT.</td>
</tr>
<tr>
<td>SetPrnXoffs</td>
<td>Adjusts X positions for overlay files (*.prn).</td>
</tr>
<tr>
<td>SetPrnYoffs</td>
<td>Adjusts Y positions for overlay files (*.prn).</td>
</tr>
<tr>
<td>SetXoffs</td>
<td>Adjusts X positions for all output fields, pictures, and overlays towards the right.</td>
</tr>
<tr>
<td>SetYoffs</td>
<td>Adjusts Y positions for all output fields, pictures, and overlays downwards.</td>
</tr>
</tbody>
</table>
Preview functions

Preview functions are used to check if a job is a preview job or not. If the job is a preview job, a script function can be used to select an appropriate connector for the preview Process.

The table below contains a short description of each script function. For information about syntax, etc., see the full description of each script function.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsPreview</td>
<td>Checks if the current job is a preview job or not.</td>
</tr>
<tr>
<td>GetPreviewType</td>
<td>Returns a number, indicating whether a job is a preview job or not. If the job is a preview job, the number indicates from where the preview job was invoked.</td>
</tr>
<tr>
<td>GetRequestedPreviewContentType</td>
<td>Only available for preview calls from Ad Hoc Correspondence, Correspondence Reviewer, or Composition Center.</td>
</tr>
<tr>
<td></td>
<td>Returns a string containing an optionally requested content type for a response to a preview request. The returned value can be used to select an appropriate connector for the preview Process.</td>
</tr>
</tbody>
</table>

Preview examples

Example 45  Selecting output connector and Process depending on preview type

A Message is configured with two Processes and four output connectors. At runtime, the output connector and Process are selected depending on the type of preview job. If the preview is invoked from a web service, the content type is also considered.

Processes:
- MyPageOUT
- MyXMLOUT

In the Runtime Process settings, the Processes are set up to be triggered using the CallProc scripting function (that is, the Select automatically option is cleared).

Output connectors:
- PDF
- JPG
- XML
- HTTP response
The Message is configured to be invoked through service requests. If the preview is invoked from a web service, the expected content type (if any) from the web service request is also considered when selecting the connector for the preview Process.

The following script is run as a Before Message script.

```plaintext
$selectedConnector = "PDF";
$selectedProcess = "MyPageOUT";

if(IsPreview())
{
    preproclog(0, "In preview, checking type and requested connector");

    if(GetPreviewType() = 1)
    {
        preproclog(0, "Web service preview");

        $requestedCT = GetRequestedPreviewContentType();
        if($requestedCT = "application/pdf")
            $selectedConnector = "PDF";
        else if($requestedCT = "application/jpg" || $requestedCT = "application/jpeg")
            $selectedConnector = "JPG";
        else if($requestedCT = "application/xml" || $requestedCT = "text/xml")
            {
                $selectedConnector = "XML";
                $selectedProcess = "MyXMLOUT";
            }
    }
    else if(GetPreviewType() = 2)
    {
        $selectedConnector = "HTTP response";
    }
    else
    {
        preproclog(0, "Other preview");
    }
}

preproclog(0, "Process: " + $selectedProcess);
preproclog(0, "Connector: " + $selectedConnector);

CallProc($selectedProcess);
```
## Project related functions

The table below contains a short description of each script function. For information about syntax, examples, etc., see the full description of each script function.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>GetDesignCenterVersion</code></td>
<td>Returns the Design Center version that was used to create the Project.</td>
</tr>
<tr>
<td><code>GetPhysicalPlatform</code></td>
<td>Returns the name of the physical platform layer deployed in Control Center.</td>
</tr>
<tr>
<td><code>GetProjectName</code></td>
<td>Returns the name of the Design Center Project.</td>
</tr>
<tr>
<td><code>GetProjectRevision</code></td>
<td>Returns the revision of the Design Center Project. The revision is the version control system label defined when issuing the Create Release command in Design Center.</td>
</tr>
<tr>
<td><code>GetSender</code></td>
<td>Returns the name of the sender of the job. You specify the sender in the Runtime configuration in Design Center.</td>
</tr>
</tbody>
</table>
Queue and repository functions

The table below contains a short description of each script function. For information about syntax, examples, etc., see the full description of each script function.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetMetaDataDocument</td>
<td>This script function is used for retrieving the value of the specified metadata item of the current incoming document.</td>
</tr>
<tr>
<td>GetMetaDataJob</td>
<td>Returns the value of a metadata attribute associated with the input job. You use the SetMetaDataJob function to create the metadata attribute.</td>
</tr>
<tr>
<td>SetMetaDataJob</td>
<td>Creates a new metadata attribute for the input job.</td>
</tr>
<tr>
<td>IsamAdd</td>
<td>Adds a record to the Isam file path. The key part of the record must be placed according to the key specified in IsamCreate.</td>
</tr>
<tr>
<td>IsamClose</td>
<td>Closes the specified Isam file.</td>
</tr>
<tr>
<td>IsamCreate</td>
<td>Creates an Isam file with the specified record length and the specified key position and length.</td>
</tr>
<tr>
<td>IsamErase</td>
<td>Removes the specified Isam file.</td>
</tr>
<tr>
<td>IsamGet</td>
<td>Retrieves a record from the specified Isam file.</td>
</tr>
<tr>
<td>IsamOpen</td>
<td>Opens the specified Isam file.</td>
</tr>
</tbody>
</table>
Session handling functions

The table below contains a short description of each script function. For information about syntax, examples, etc., see the full description of each script function.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SessionEnd</td>
<td>Ends a session in the StreamServer, and removes all associated variables.</td>
</tr>
<tr>
<td>SessionGetVariable</td>
<td>Retrieves the value of a given variable in a given session.</td>
</tr>
<tr>
<td>SessionSetVariable</td>
<td>Specifies a value for a given variable in a given session. If the variable does not exist, it will be created.</td>
</tr>
<tr>
<td>SessionStart</td>
<td>Starts a new session in the StreamServer, and specifies the length of the session (minutes). The session will be removed if it has not been used for the specified number of minutes.</td>
</tr>
</tbody>
</table>
**StoryTeller functions**

The following functions with the ‘St’ prefix are to be used in StoryTeller tool only. The table contains a short description of each script function. For information about syntax, examples, etc., see the full description of each script function.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StEvalXPath</td>
<td>Evaluates an XPath and returns a string.</td>
</tr>
<tr>
<td>StGetParagraphPageXMm</td>
<td>Returns the paragraph X coordinate in millimeters relative to page.</td>
</tr>
<tr>
<td>StGetParagraphPageXPt</td>
<td>Returns the paragraph X coordinate in points relative to page.</td>
</tr>
<tr>
<td>StGetParagraphPageYMm</td>
<td>Returns the paragraph Y coordinate in millimeters relative to page.</td>
</tr>
<tr>
<td>StGetParagraphPageYPt</td>
<td>Returns the paragraph Y coordinate in points relative to page.</td>
</tr>
<tr>
<td>StGetParagraphXMm</td>
<td>Returns the paragraph X coordinate in millimeters relative to text area.</td>
</tr>
<tr>
<td>StGetParagraphXPt</td>
<td>Returns the paragraph X coordinate in points relative to text area.</td>
</tr>
<tr>
<td>StGetParagraphYMm</td>
<td>Returns the paragraph Y coordinate in millimeters relative to text area.</td>
</tr>
<tr>
<td>StGetParagraphYPt</td>
<td>Returns the paragraph Y coordinate in points relative to text area.</td>
</tr>
<tr>
<td>StGetProcessingProperty</td>
<td>Returns the value of the specified processing property.</td>
</tr>
<tr>
<td>StGetProperty</td>
<td>Returns the value of a non-dimensional property.</td>
</tr>
<tr>
<td>StGetPropertyMm</td>
<td>Returns the value in millimeters of a dimensional property.</td>
</tr>
<tr>
<td>StGetPropertyPt</td>
<td>Returns the value in points of a dimensional property.</td>
</tr>
<tr>
<td>StIsEmpty</td>
<td>Checks if an object is empty.</td>
</tr>
<tr>
<td>StLanguageLookup</td>
<td>Returns a translated value from .s1s lookup tables.</td>
</tr>
<tr>
<td>StMsgAddNode</td>
<td>Appends a node <em>name</em> to a Message.</td>
</tr>
<tr>
<td>StMsgAttachXML</td>
<td>Appends an XML DOM node – loaded from an XML string – to a Message node.</td>
</tr>
<tr>
<td>StMsgAttachXMLFile</td>
<td>Appends an XML DOM node – loaded from an XML file – to a Message node.</td>
</tr>
<tr>
<td>StMsgSaveToFile</td>
<td>Saves Message data for an Event into an XML file.</td>
</tr>
<tr>
<td>StSetProperty</td>
<td>Sets a value on a non-dimensional property.</td>
</tr>
<tr>
<td>StSetPropertyMm</td>
<td>Sets a value in millimeters on a dimensional property.</td>
</tr>
<tr>
<td>StSetPropertyPt</td>
<td>Sets a value in points on a dimensional property.</td>
</tr>
<tr>
<td>StURIExist</td>
<td>Checks if a specified URI exists, including StoryTeller protocols.</td>
</tr>
</tbody>
</table>
## String handling functions

The table below contains a short description of each script function. For information about syntax, examples, etc., see the full description of each script function.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Base64DecodeString</code></td>
<td>Decodes a base64 encoded text string.</td>
</tr>
<tr>
<td><code>Base64DecodeStringUsingCodepage</code></td>
<td>Decodes a base64 encoded text string that was encoded with the <code>Base64EncodeStringUsingCodepage</code> function.</td>
</tr>
<tr>
<td><code>Base64EncodeString</code></td>
<td>Encodes a text string to base64 format.</td>
</tr>
<tr>
<td><code>Base64EncodeStringUsingCodepage</code></td>
<td>Encodes a text string to base64 format. Before encoding the text string, it is converted to the specified codepage.</td>
</tr>
<tr>
<td></td>
<td>This function is used instead of <code>Base64EncodeString</code> if the receiving system does not handle Unicode.</td>
</tr>
<tr>
<td><code>Str</code></td>
<td>Converts a numerical value to a string.</td>
</tr>
<tr>
<td><code>StrBlk</code></td>
<td>Removes leading white space (spaces and tabs) in a string.</td>
</tr>
<tr>
<td><code>StridX</code></td>
<td>Locates the first occurrence of a sub-string in a string.</td>
</tr>
<tr>
<td><code>StrLadj</code></td>
<td>Left-aligns and truncates (or extends) a string to a specified length.</td>
</tr>
<tr>
<td><code>StrLen</code></td>
<td>Returns the length of a string.</td>
</tr>
<tr>
<td><code>StrQTok</code></td>
<td>Takes four strings and a one-dimensional array and returns a numeric value indicating the number of tokens passed into the array.</td>
</tr>
<tr>
<td><code>StrrBlk</code></td>
<td>Removes trailing white spaces (spaces and tabs) from the input string.</td>
</tr>
<tr>
<td><code>StrTok</code></td>
<td>Takes two strings and a one-dimensional array and returns a numeric value indicating the number of tokens passed into the array.</td>
</tr>
<tr>
<td><code>SubStr</code></td>
<td>Returns a sub-string of the input string.</td>
</tr>
<tr>
<td><code>SubStrRepl</code></td>
<td>Replaces a search string with a sub-string in an input string.</td>
</tr>
<tr>
<td><code>ToLower</code></td>
<td>Converts all characters in a string to lower case.</td>
</tr>
<tr>
<td><code>ToUpper</code></td>
<td>Converts all characters in a string to upper case.</td>
</tr>
<tr>
<td><code>Utf8DecodeString</code></td>
<td>Decodes an UTF8 encoded text string.</td>
</tr>
<tr>
<td><code>Utf8EncodeString</code></td>
<td>Encodes a text string to UTF8 format.</td>
</tr>
</tbody>
</table>
Substitution table functions

There are a number of built-in script functions that can be used to retrieve or modify values in substitution tables. A substitution table contains several entries (rows), and each entry contains two or more columns. The first column is the key, which is a unique identifier for each entry. The remaining columns contain the values for the entry (value 0, value 1,..., value N).

Example 46  Substitution table.

The substitution table below contains two entries. Each entry contains two values

```plaintext
//!CodePage UTF8!
//!multicolumn!
//key     value 0     value 1
agatha   christie   1890
alfred    hitchcock  1899
```

Creating a substitution table

You can create a substitution table as a table resource in any resource set in your Project. Each entry must contain two or more columns, where the first column is the key (a unique identifier for the entry) and the remaining columns contain the values for the entry.

You can use space or tab as delimiter for the columns. Note that keys and values must be within double quotes (") if they contain spaces.

Referencing a substitution table

When you use a substitution table script function, you must specify the path to the substitution table to work with. If you create the substitution table as a table resource, the substitution table file will be exported and deployed to:

```plaintext
../data/tables/<filename>
```

Example 47  Using the Subst function.

In this example, the substitution table `author.tbl` is created as a table resource. The script function `Subst` is used to retrieve the first value of the substitution table entry with the key `agatha`.

```plaintext
Subst("../data/tables/author.tbl", "agatha");
```
Removing substitution tables from memory after use
A substitution table is kept in memory until the script function *EraseSubst* is called for that table. This means StreamServer performance will be affected if several tables are created in a job, and if the tables are never removed from memory during the job using the *EraseSubst* function.

For example, if you use variables in the table names, you must make sure there are a limited number of tables in the memory. In this case you must call *EraseSubst* for each table after it is used.

Substitution table functions
The table below contains a short description of each script function. For information about syntax, examples, etc., see the full description of each script function.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddSubst</td>
<td>Adds a value to the specified value column of the specified substitution</td>
</tr>
<tr>
<td></td>
<td>table entry.</td>
</tr>
<tr>
<td>ClearSubst</td>
<td>Clears the contents of the specified substitution table, but does not remove</td>
</tr>
<tr>
<td></td>
<td>the actual table from memory.</td>
</tr>
<tr>
<td>DelSubstKey</td>
<td>Deletes the specified entry from the specified substitution table.</td>
</tr>
<tr>
<td>EraseSubst</td>
<td>Removes the specified substitution table from memory.</td>
</tr>
<tr>
<td>GetSubst</td>
<td>Returns the value from the specified value column in the specified</td>
</tr>
<tr>
<td></td>
<td>substitution table entry.</td>
</tr>
<tr>
<td>GetSubstColCount</td>
<td>Returns the number of columns in the specified entry in the specified</td>
</tr>
<tr>
<td></td>
<td>substitution table.</td>
</tr>
<tr>
<td>GetSubstKey</td>
<td>Returns a key from a substitution table.</td>
</tr>
<tr>
<td>GetSubstKeyCount</td>
<td>Returns the number of entries in a substitution table.</td>
</tr>
<tr>
<td>ReadSubst</td>
<td>Re-reads a substitution table from disk.</td>
</tr>
<tr>
<td>SetSubst</td>
<td>Assigns a value to an entry in a substitution table.</td>
</tr>
<tr>
<td>Subst</td>
<td>Returns the first value (value column 0) in the specified substitution</td>
</tr>
<tr>
<td></td>
<td>table entry.</td>
</tr>
<tr>
<td>SubstArr</td>
<td>Looks up a key in a substitution table, and creates an array containing the</td>
</tr>
<tr>
<td></td>
<td>entry values.</td>
</tr>
<tr>
<td>WriteSubst</td>
<td>Writes a substitution table to a specified path.</td>
</tr>
</tbody>
</table>

Executing a substitution table function only once
When you use substitution functions to modify substitution tables, you must be aware of the fact that StreamServer runs the scripts twice:
• Once during the preprocess phase
• Once during the process phase

To execute the substitution functions only once, you can use the PreProc function. The PreProc function returns 1 when the server is in the preprocess phase.

**Example 48 Using the PreProc function.**

```c
/*Check if the server runs in preprocess phase. If it does, perform the substitution.*/
if (PreProc() = 1)
{
    $nb = AddSubst("../data/tables/author.tbl", "alfred", 1, "100");
}
```
Type and conversion functions

The table below contains a short description of each script function. For information about syntax, examples, etc., see the full description of each script function.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AmountValue</code></td>
<td>Converts a string to a numerical value using a specified format.</td>
</tr>
<tr>
<td><code>ConvCurrMsgToUC</code></td>
<td>Converts the encoding of input data from the specified code page to UCS-2.</td>
</tr>
<tr>
<td><code>HexStr</code></td>
<td>Copies a string and interprets numbers in angle brackets as two-digit</td>
</tr>
<tr>
<td></td>
<td>hexadecimal numbers (useful for Escape sequences). Other characters</td>
</tr>
<tr>
<td></td>
<td>are returned unchanged.</td>
</tr>
<tr>
<td><code>IsAmountValue</code></td>
<td>Checks if a string expression is in ‘amount’ format. ‘Amount’ format is a</td>
</tr>
<tr>
<td></td>
<td>string in the format commonly used to represent e.g. monetary amounts,</td>
</tr>
<tr>
<td></td>
<td>for example 12,345,678.90</td>
</tr>
<tr>
<td><code>IsNum</code></td>
<td>Checks if a string expression consists of digits.</td>
</tr>
<tr>
<td><code>NFormat</code></td>
<td>Converts a number to a string, formatting the string according to the</td>
</tr>
<tr>
<td></td>
<td>specified format syntax.</td>
</tr>
<tr>
<td><code>Num</code></td>
<td>Converts a string expression to a numerical value.</td>
</tr>
</tbody>
</table>
# ODBC functions

StreamServe ODBC functions enable interfacing with ODBC compliant data sources. You can connect to any data source that has an ODBC driver configured for the platform that StreamServe is operating on.

To be able to use the StreamServe ODBC functions, you must be familiar with ODBC and data source concepts, SQL, scripting in StreamServe, and the operating platforms that both StreamServe and the data sources to be used will be running on.

## Requirements

To use the ODBC script functions, you must

- **Configure an ODBC data source**
  
  You must configure an ODBC data source for each data source you want to use. Once configured, the data sources will be accessible for all ODBC functionality, not just with StreamServe.

  On UNIX, you must configure the `odbc.ini` file in `<StreamServe installation>/Platform` to comply with your environment.

- **Use a reliable ODBC driver**

  The ODBC functions provide an interface to ODBC and not to a specific data source. You must choose a reliable and thoroughly tested ODBC driver for your data source. The StreamServe ODBC module does not check how your ODBC driver executes the SQL statements.

  On UNIX, you must configure the `odbcinst.ini` files in `<StreamServe installation>/Platform` if you use any other driver than DataDirect.

- **Specify the appropriate ODBC startup arguments**

  You must specify the appropriate ODBC startup arguments before you export the StreamServe Project. See *ODBC startup argument* on page 96.

## Escape sequences

The less-than sign (`<`) is a reserved character and must be escaped if used in a string, see *Escape sequences* on page 40.

## ODBC functions

The table below contains a short description of each script function. For information about syntax, examples, etc., see the full description of each script function.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>OdbcBeginTrans</em></td>
<td>Begins a new transaction, which opens two possibilities:</td>
</tr>
<tr>
<td></td>
<td>- To commit the transaction, call the <em>ODBCCommitTrans</em> function.</td>
</tr>
<tr>
<td></td>
<td>- To end the transaction, call the <em>ODBCRollbackTrans</em> function.</td>
</tr>
<tr>
<td>Function name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>OdbcCommitTrans</td>
<td>Commits a transaction that you have opened with ODBCBeginTrans. The changes within the transaction are saved to the database and the transaction is ended.</td>
</tr>
<tr>
<td>OdbcConnect</td>
<td>Connects the user to the data source.</td>
</tr>
<tr>
<td>OdbcConnectW</td>
<td>Connects the user to the data source. If the data retrieved with ODBCGetFirst, ODBCGetNext, or ODBCGetOne is of Unicode type, you must use the OdbcConnectW (instead of OdbcConnect) when connecting to the data source.</td>
</tr>
<tr>
<td>OdbcCloseQuery</td>
<td>Closes any open queries (statements) in the data source. An open query exists when OdbcGetFirst has been executed and all rows in the result set have not been retrieved with OdbcGetNext.</td>
</tr>
<tr>
<td>OdbcDate2Date</td>
<td>Extracts the date from a date and time string.</td>
</tr>
<tr>
<td>OdbcDate2Time</td>
<td>Extracts the time from a date and time string.</td>
</tr>
<tr>
<td>OdbcDisconnect</td>
<td>Disconnects the server from the data source.</td>
</tr>
<tr>
<td>OdbcExecute</td>
<td>Executes SQL statements. Any result sets that the statement generates are ignored. This function can be used, for example, to update record fields in the data source.</td>
</tr>
<tr>
<td>OdbcExecuteEx</td>
<td>Same as OdbcExecute. The difference is that this function can be configured to accept that a CREATE TABLE statement fails.</td>
</tr>
<tr>
<td>OdbcGetOne</td>
<td>Executes an SQL statement and copies the values from the first row in the result set to script variables.</td>
</tr>
<tr>
<td>OdbcGetFirst</td>
<td>Executes an SQL statement and copies the values from the first row in the result set to the specified script variables. The OdbcGetNext function is then used to retrieve the remaining parts of the result set.</td>
</tr>
<tr>
<td>OdbcGetNext</td>
<td>Retrieves the values of the columns in the second row of a result set. This function can only be called after OdbcGetFirst has been called to retrieve the first row in the result set. OdbcGetNext can then be called repeatedly to process all the remaining rows.</td>
</tr>
<tr>
<td>OdbcRollbackTrans</td>
<td>Rolls back and ends a transaction that you have opened with ODBCBeginTrans.</td>
</tr>
<tr>
<td>OdbcSetCodepage</td>
<td>Sets the code page.                                                                arsi</td>
</tr>
</tbody>
</table>
**ODBC startup argument**

You can specify the following ODBC startup argument before you export the StreamServe Project.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-odbcTimeOut value</td>
<td>Specifies the time-out value (seconds) StreamServer will use when connecting to data sources. If no value is entered, the default value (30 seconds) will be used.</td>
</tr>
</tbody>
</table>
Script functions in alphabetical order

This section contains detailed descriptions of all StreamServe script functions. The script functions are listed in alphabetical order.

A

Abs

Syntax

Abs(num);

<table>
<thead>
<tr>
<th>num</th>
<th>A numeric expression.</th>
</tr>
</thead>
</table>

Description

Returns the absolute value of a number.

Returns

The absolute value of the num argument.

Example

The following example illustrates the use of the Abs function.

$c = \text{Abs}(\text{Num}(a) - \text{Num}(b))$;

In this example, the num argument is a subtraction of Num(b) from Num(a). If b is 40 and a is 20, then $c$ will be the absolute value of -20, i.e. 20.

AddSubst

Syntax

AddSubst(tbl_file, str_key, num_col, str_value);

<table>
<thead>
<tr>
<th>tbl_file</th>
<th>A string specifying the path and filename of the substitution table.</th>
</tr>
</thead>
<tbody>
<tr>
<td>str_key</td>
<td>A string specifying the key of the substitution table entry to search for.</td>
</tr>
<tr>
<td>num_col</td>
<td>A number specifying the value column. The first value column number is 0.</td>
</tr>
</tbody>
</table>
### Description

Adds \( \text{str\_value} \) to the value in \( \text{num\_col} \) of the specified substitution table entry. It does not affect the physical file stored on disk, only the representation of the table in the memory. The substitution table file is specified by \( \text{tbl\_file} \), and the substitution table entry is specified by \( \text{str\_key} \). If the key or column does not exist, it will be added.

### Returns

Returns the sum of the old value retrieved from the substitution table, and the value given in \( \text{str\_value} \).

### Example

In this examples, the following substitution table (\texttt{author.tbl}) is used.

```plaintext
//!CodePage UTF8!
//!multicolumn!
agatha christie  1890
alfred hitchcock 1899
```

**Example 49**  
**Adding a value.**

In the following script, the variable \$century \ gets the value \( 1899 + 100 \).

```plaintext
$century = AddSubst("../data/tables/author.tbl","alfred",1,"100");
```

**Example 50**  
**Adding a new substitution table entry.**

The following script adds the substitution table entry \texttt{astrid}:

```plaintext
SetSubst("../data/tables/author.tbl","astrid",0,"lindgren");
AddSubst("../data/tables/author.tbl","astrid",1,"1907");
WriteSubst("../data/tables/author.tbl");
```
After the script is executed, `author.tbl` is changed as follows:

```plaintext
//!CodePage UTF8!
//!multicolumn!
agatha christie  1890
alfred hitchcock  1899
astrid lindgren   1907
```

**Notes**
- A substitution table is kept in memory until the script function `EraseSubst` is called for that table.

**Amount**

The `Amount` function is replaced by `AmountValue`.

**AmountValue**

**Syntax**

```
AmountValue(str_amount, thousand_sep, decimal_sep);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>str_amount</code></td>
<td>A string specifying the amount.</td>
</tr>
<tr>
<td><code>thousand_sep</code></td>
<td>A string specifying the thousand separator.</td>
</tr>
<tr>
<td><code>decimal_sep</code></td>
<td>A string specifying the decimal separator.</td>
</tr>
</tbody>
</table>

**Description**

Converts a string to a numerical value using a specified format.

The thousand separator is only allowed every three digits, except in the beginning or at the end of the number. There cannot be any thousand separator after the decimal separator.

For example `amountvalue("1,200.65", ",", ".")`; For comparison, `amountvalue("1,2000.65", ",", ".")`; is not a valid conversion.

If you leave the separator arguments empty, the separators are not checked. For example, if you leave the thousand separator argument empty, as in `amountvalue("1200.65", ",", ".")`; the number does not need to contain a thousand separator.

The numerical value can contain a minus sign (-) both at the beginning and at the end of the value. A plus sign (+) is only allowed in the beginning.
**Returns**
A numeric expression.

**Example**
$a="1,200.65";$
$b="1 '765,15";$
$c=amountvalue($a,"",".");$d=amountvalue($b," ",");

**Result:**
$c=1200.65$
$d=1765.15$

**ArchiveBegin**

ArchiveBegin() is an old function and is removed.

**ArchiveDisable**

ArchiveDisable() is an old function and is removed.

**ArchiveEnable**

ArchiveEnable() is an old function and is removed.

**ArchiveEnd**

ArchiveBegin() is an old function and is removed.

**ArraySize**

**Syntax**

ArraySize(array_name);

<table>
<thead>
<tr>
<th>array_name</th>
<th>A string specifying the name of the array.</th>
</tr>
</thead>
</table>

**Description**
Counts the number of elements stored in an array.
Returns

| num | The number of elements stored in the array. |

Examples

**Example 51**  Counting the number of items in a one dimensional array.

In this example, ArraySize is used to count the numbers in the array $arr.$

```
$arr[0] = "Mike Jones";
$arr[1] = "Bill Carlson";
$arr[2] = "Eve Smith";
$numberOfItems = ArraySize($arr);
```

When the script is run, $numberOfItems gets the value 3.

**Example 52**  Counting the number of items in a one dimensional array.

In this example, ArraySize is used to count the numbers in the array $arr.$

```
$arr[0][0] = "Mike";
$arr[0][1] = "Jones";
$arr[1][0] = "Bill";
$arr[1][1] = "Carlson";
$arr[2][0] = "Eve";
$arr[2][1] = "Smith";
$numberOfItems = ArraySize($arr);
```

When the script is run, $numberOfItems gets the value 3, i.e. the same result as in the previous example.

**Example 53**  Adding items to arrays.

This example has two arrays:

- $arrTele containing phone numbers.
- $arrAmount containing call charges.

If a phone number for a specific call is found in $arrTele, the charge for the call should be added to the corresponding position in $arrAmount.

If the phone number is not found, the new phone number should be added to $arrTele and the new charge to $arrAmount. The ArraySize function specifies the position where to add the new number and charge at the end of the arrays.

```
$retIndex = FindInArray($arrTele, $noDialled);
if ($retIndex == "-1")
{

```
AtJobEnd

**Syntax**

AtJobEnd(eventname);

<table>
<thead>
<tr>
<th>eventname</th>
<th>A string specifying the name of the Event.</th>
</tr>
</thead>
</table>

**Description**

Triggers a specific Event at the end of a job. Typically used to produce a closing page showing the number of invoices printed so far, or the total sum of all invoices. This scripting function is not supported in Post-processor scripting.

**Returns**

A number indicating whether the Event was found and will be triggered at the end of the job.

<table>
<thead>
<tr>
<th>1</th>
<th>The Event will be triggered.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The Event will not be triggered.</td>
</tr>
</tbody>
</table>

**Example**

AtJobEnd("invoice");

AttachmentBegin

**Syntax**

AttachmentBegin();

**Description**

The **AttachmentBegin** and **AttachmentEnd** script functions define the beginning and end of an attachment:

- The **AttachmentBegin** script function is normally called from a Before Process script on the first Process to include in an attachment.
- The **AttachmentEnd** script function ends the **AttachmentBegin** function, and is called from an After Process script on the last Process to include in an attachment.
The attachment can be added to an email sent via a email output connector, for example an SMTP (MIME) output connector.

To be able to add the attachment to the email, the attachment must be finished before the email is sent via the email output connector. When the attachment has to be finished depends on the output mode defined for the email connector.

<table>
<thead>
<tr>
<th>Output mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process</td>
<td>The mail is sent when the Process that invokes the email connector is completed. Any attachment intended for the current email must be created before that time, i.e. all attachments from other Processes must be run before the email connector Process is run.</td>
</tr>
<tr>
<td>Document</td>
<td>The email is sent when the document ends, i.e. the document containing the Process that invoked the email connector. Any attachment intended for the current email must belong to a Process run before the email connector Process, or after the email connector Process but before the document trigger changes value.</td>
</tr>
<tr>
<td>Job</td>
<td>Any attachment from Processes within the same job as the email connector Process can be attached to the email connector output.</td>
</tr>
</tbody>
</table>

**Example**

Before Process script defining the beginning of the attachment:

```plaintext
$attachmentfile = AttachmentBegin();
```

After Process script defining the end of the attachment:

```plaintext
AttachmentEnd($attachmentfile);
```

To add the attachment to an email, `$attachmentfile` is used as file name.

**AttachmentEnd**

**Syntax**

```plaintext
AttachmentEnd(filename);
```

**Description**

The `AttachmentBegin` and `AttachmentEnd` script functions define the beginning and end of an attachment:

- The `AttachmentBegin` script function is normally called from a Before Process script on the first Process to include in an attachment.
- The `AttachmentEnd` script function ends the `AttachmentBegin` function, and is called from an After Process script on the last Process to include in an attachment.
The attachment can be added to an email sent via a email output connector, for example an SMTP (MIME) output connector.

To be able to add the attachment to the email, the attachment must be finished before the email is sent via the email output connector. When the attachment has to be finished depends on the output mode defined for the email connector.

<table>
<thead>
<tr>
<th>Output mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process</td>
<td>The mail is sent when the Process that invokes the email connector is completed. Any attachment intended for the current email must be created before that time, i.e. all attachments from other Processes must be run before the email connector Process is run.</td>
</tr>
<tr>
<td>Document</td>
<td>The email is sent when the document ends, i.e. the document containing the Process that invoked the email connector. Any attachment intended for the current email must belong to a Process run before the email connector Process, or after the email connector Process but before the document trigger changes value.</td>
</tr>
<tr>
<td>Job</td>
<td>Any attachment from Processes within the same job as the email connector Process can be attached to the email connector output.</td>
</tr>
</tbody>
</table>

**Example**

Before Process script defining the beginning of the attachment:

```javascript
$attachmentfile = AttachmentBegin();
```

After Process script defining the end of the attachment:

```javascript
AttachmentEnd($attachmentfile);
```

To add the attachment to an email, `$attachmentfile` is used as file name.

**Base64DecodeFile**

**Syntax**

```javascript
Base64DecodeFile(InputFile, OutputFile);
```

<table>
<thead>
<tr>
<th>InputFile</th>
<th>Full path and name of the input file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>OutputFile</td>
<td>Full path and name of the output file with decoded data. If the file already exists, it will be overwritten.</td>
</tr>
</tbody>
</table>
Description
Decodes a base64 encoded file and stores the result in an output file.

Returns
A number indicating whether the decoding was successful.

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Decoding succeeded.</td>
</tr>
<tr>
<td>0</td>
<td>Decoding failed.</td>
</tr>
</tbody>
</table>

Example

```csharp
base64DecodeFile("C:\Data\ToDecode.txt","C:\Data\Decoded.txt");
```

Base64DecodeString

**Syntax**

```csharp
Base64DecodeString(InputStr);
```

<table>
<thead>
<tr>
<th>InputStr</th>
<th>Input string to be decoded.</th>
</tr>
</thead>
</table>

**Description**

Decodes a base64 encoded text string.

**Returns**

The decoded string. If decoding fails an empty string is returned.

**Example**

```csharp
$DecodedString = base64DecodeString("string to be decoded");
```

Base64DecodeStringUsingCodepage

**Syntax**

```csharp
Base64DecodeStringUsingCodepage(InputStr);
```

<table>
<thead>
<tr>
<th>InputStr</th>
<th>Input string to be decoded.</th>
</tr>
</thead>
</table>

**Description**

Decodes a base64 encoded text string that was encoded with the Base64EncodeStringUsingCodepage function.

**Returns**

The decoded string. If decoding fails an empty string is returned.
Example
$decoded_string = base64DecodeStringUsingCodepage($string);

Base64EncodeFile

Syntax
Base64EncodeFile(InputFile, OutputFile);

<table>
<thead>
<tr>
<th>InputFile</th>
<th>Full path and name of input file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>OutputFile</td>
<td>Full path and name of output file with encoded data. If the file already exists it will be overwritten.</td>
</tr>
</tbody>
</table>

Description
Encodes an input file to base64 format. The encoded data is stored in an output file.

Returns
A number indicating whether the encoding was successful.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Encoding succeeded.</td>
</tr>
<tr>
<td>0</td>
<td>Encoding failed.</td>
</tr>
</tbody>
</table>

Example
base64EncodeFile("C:\Data\ToBeEncoded.txt","C:\Data\Encoded.txt");

Base64EncodeString

Syntax
Base64EncodeString(InputStr);

<table>
<thead>
<tr>
<th>InputStr</th>
<th>Input string to be base64 encoded.</th>
</tr>
</thead>
</table>

Description
Encodes a text string to base64 format.

Returns
The encoded string. If encoding fails an empty string is returned.

Example
$Base64_EncodedString = base64EncodeString("string to be encoded");
Base64EncodeStringUsingCodepage

**Syntax**

Base64EncodeStringUsingCodepage(InputStr, Codepage);

<table>
<thead>
<tr>
<th>InputStr</th>
<th>Input string to be base64 encoded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Codepage</td>
<td>The codepage to convert the input string to before encoding it to base64 format.</td>
</tr>
</tbody>
</table>

**Description**

Encodes a text string to base64 format. Before encoding the text string, it is converted to the specified codepage. This function is used instead of Base64EncodeString if the receiving system does not handle Unicode.

**Returns**

The encoded string. If encoding fails an empty string is returned.

**Example**

```
$encoded_string = base64EncodeStringUsingCodepage($string, "iso 8859-1");
```

BinAnd

**Syntax**

BinAnd(num_1, num_2)

<table>
<thead>
<tr>
<th>num_1</th>
<th>Numeric expression.</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_2</td>
<td>Numeric expression.</td>
</tr>
</tbody>
</table>

**Description**

Performs an AND operation on the binary representation of the complement integer values of the two arguments (Int(num_1) and Int(num_2)).

**Returns**

A number.

**Examples**

```
BinAnd(0,1)
Result: 0
BinAnd(2,1)
Result: 0
BinAnd(10,2)
Result: 2
```
BinOr

Syntax
BinOr(num_1, num_2)

<table>
<thead>
<tr>
<th>num_1</th>
<th>Numeric expression.</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_2</td>
<td>Numeric expression.</td>
</tr>
</tbody>
</table>

Description
Performs an OR operation on the binary representation of the complement integer values of the two arguments (Int(num_1) and Int(num_2)).

Returns
A number.

Examples
BinOr(0,1)
Result: 1
BinOr(2,1)
Result: 3
BinOr(10,2)
Result: 10

BinXOr

Syntax
BinXOr(num_1, num_2)

<table>
<thead>
<tr>
<th>num_1</th>
<th>Numeric expression.</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_2</td>
<td>Numeric expression.</td>
</tr>
</tbody>
</table>

Description
Returns the exclusive OR on the binary representation of the complement integer values of the two arguments (Int(num_1) and Int(num_2)).

Returns
A number.

Examples
BinXOr(0,1)
Result: 1
BinXOr(2,1)
Result: 3
BinXOr(10,2)
Result: 8

C

CallBlock

Syntax
CallBlock(free_block);

| free_block | A string specifying the name of the Free Block. |

Description
Calls a Free Block.

Returns
N/A

Example
callblock("frBlkLocalCalls");

CallProc

Syntax
CallProc(str_proc_name);

| str_proc_name | A string specifying the name of the Process. |

Description
Invokes a Process in the Message from which it is called. For example, you can
deselect the Select automatically option in the Process Runtime settings, and use
the CallProc function to select a Process based on a variable in the input data.
The Process invoked must be included in the Message configuration from which
it is called.

The CallProc function can be used in Before/After Message or Before/After
Process scripts. It is not supported in Post-processor scripting.

Returns
A number indicating whether the Process was invoked successfully.

| 0 | Ok. The Process was invoked successfully. |
Example

callproc("invoice_process");

CancelJob

Syntax

CancelJob(event_id);

| event_id | A string specifying the EventID, or an asterisk (*). If an EventID is specified, the corresponding Event will be executed and can be used to report the interruption. There is no input Message available for this Event, which means that all input data must be provided via variables.
|          | If no reporting is desired, an asterisk (*) should be used instead of the EventID. |

Description

Cancels the current job. The cancellation is executed after pre-processing, i.e. all Events are pre-processed before the job is interrupted (for more information on the pre-process phase, _Formatting phase_ on page 19). This scripting function is not supported in Post-processor scripting.

Returns

N/A

Example

if (getcurrobjy() > 75 and $joberr = "")
{
    $joberr = "too many items in block xx"
}
canceljob("joberror");

Result:

If the current Y position in the frame exceeds 75 millimeters and the $joberr variable has not been assigned a value, the $joberr variable is assigned the string value _too many items in block xx_, and the canceljob("joberror") script will be executed. This script cancels the job and calls the joberror Event, which uses the $joberr variable.
Clear

Syntax

```plaintext
clear(var_1 [, var_2, var_3, ...var_N]);
```

| var_1, var_2, ... | A list of variables to be cleared. |

Description

Clears the specified variables.

Returns

N/A

Example

```plaintext
clear($a, $b, $c);
```

Result:

```
$a= "", $b= "", $c= ""
```

ClearSubst

Syntax

```plaintext
ClearSubst(tbl_file);
```

| tbl_file | A string specifying the path and filename of the substitution table. |

Description

Clears the contents of the specified substitution table. It does not affect the physical file stored on disk, only the representation of the table in the memory.

The ClearSubst function is similar to EraseSubst:

- ClearSubst clears the table contents but keeps the actual table in the memory.
- EraseSubst removes the table from the memory.

If you experience performance problems when using substitution tables, you should use the EraseSubst function instead of ClearSubst. A substitution table is created in memory the first time it is used, and added to a global list of tables. The whole list must be searched to find the correct table, and if several tables are created for each job, the StreamServer will successively get slower if the tables are never removed from memory during the job.

If you use ClearSubst, and then access the substitution table with for example GetSubst, an empty string will be returned. If you access the table after using EraseSubst, the table will be re-read back from disk, and a value will be returned.
Table cleared

Example
In this example, the following substitution table (author.tbl) is used.

```plaintext
//!CodePage UTF8!
//!multicolumn!
agatha christie 1890
alfred hitchcock 1899
```

The following script clears the physical substitution table stored on disk (author.tbl):

```plaintext
ClearSubst ("../data/tables/author.tbl");
WriteSubst ("../data/tables/author.tbl");
```

ClearVars

**Syntax**

```plaintext
ClearVars();
```

**Description**

Clears all variables in a job. If the layout for controlling Processes within an Event depends on the variables being empty at the beginning of each Event, `ClearVars` can be called in a script after the Event.

**Note:** Do not use `ClearVars` in a Before Script before an Event as the assigning of variables is done before a Before Script.

**Returns**

N/A

**COMCreateObject**

**Syntax**

```plaintext
COMCreateObject(strObjHandl, strProgId);
```

<table>
<thead>
<tr>
<th>strObjHandl</th>
<th>The handle to be associated with the component.</th>
</tr>
</thead>
<tbody>
<tr>
<td>strProgId</td>
<td>The ProgID of the component to be loaded.</td>
</tr>
</tbody>
</table>
Description
Loads the COM component with the specified ProgID and associates it with the specified handle. Only components that support the IDispatch interface can be loaded using the COMCreateObject function.

Returns
A number indicating whether the component was successfully loaded.

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The component was successfully loaded.</td>
</tr>
<tr>
<td>-1</td>
<td>Failed to load the component.</td>
</tr>
</tbody>
</table>

Example

```javascript
$iRetVal=COMCreateObject("objTest", "COMCall.ITest");
```

COMDestroyObject

Syntax

```javascript
COMDestroyObject(strObjHandl);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strObjHandl</td>
<td>The handle associated with the component to be unloaded.</td>
</tr>
</tbody>
</table>

Description
Unloads the COM component associated with the specified handle.

Returns
A number indicating whether the component was successfully unloaded.

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The component was successfully unloaded.</td>
</tr>
<tr>
<td>-1</td>
<td>Failed to unload component.</td>
</tr>
</tbody>
</table>

Example

```
$iRetVal=COMDestroyObject("objTest");
```

COMInvoke

Syntax

```javascript
COMInvoke(strObjHandle, strMethodName, var_1 [, var_2, var_3, ... var_N], ReturnVar);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strObjHandle</td>
<td>The handle associated with the component.</td>
</tr>
<tr>
<td>strMethodName</td>
<td>The method to be invoked.</td>
</tr>
</tbody>
</table>
**Script functions in alphabetical order**

**Script functions reference**

<table>
<thead>
<tr>
<th>COMInvoke</th>
<th>Description</th>
<th>Invokes the specified method for the COM component associated with the specified handle. The COMInvoke function can take several variables as arguments. All the variables must be defined in StreamServe scripts. Explicit values, or return values from other functions can not be passed directly to the COMInvoke function.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Returns</strong></td>
<td>A number indicating whether the method was successfully invoked.</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>The method was successfully invoked.</td>
<td></td>
</tr>
<tr>
<td>num&lt;0</td>
<td>Failed to invoke the method.</td>
<td></td>
</tr>
</tbody>
</table>

**Example**

```javascript
$iRetVal=COMInvoke("objTest", "DocProcessed", $job_id, $doc_id, $nr_of_docs);
$iRetVal=COMInvoke("objTest", "FireNewJob");
```

### COMSetProperty

**Syntax**

```javascript
COMSetProperty(strObjHandl, strPropertyName, VarList);
```

| strObjHandl | The handle associated with the component. |
| strPropertyName | The property to be set. |
| VarList | One or more variable from StreamServe scripts to be used as arguments. |

**Description**

Sets the specified property for the COM component associated with the specified handle. The COMSetProperty function can take several variables as arguments. All the variables must be defined in StreamServe scripts. Explicit values, or return values from other functions can not be passed directly to the COMSetProperty function.
Returns
A number indicating whether the property was successfully set.

<table>
<thead>
<tr>
<th>num</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The property was successfully set.</td>
</tr>
<tr>
<td>num&lt;0</td>
<td>Failed to set the property.</td>
</tr>
</tbody>
</table>

Example
iRetVal=COMSetProperty("objTest", "Long", $longval);

COMGetProperty

Syntax
COMGetProperty(strObjHandl, strPropertyName, VarList);

strObjHandl The handle associated with the component.
strPropertyName The property to be retrieved.
VarList One or more variable from StreamServe scripts to be used as arguments.

Description
Retrieves the value of the specified property for the COM object associated with the specified handle. The COMGetProperty function can take several variables as arguments. All the variables must be defined in StreamServe scripts. Explicit values, or return values from other functions can not be passed directly to the COMGetProperty function.

Returns
A number indicating whether the property was successfully retrieved.

<table>
<thead>
<tr>
<th>num</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The property was successfully retrieved.</td>
</tr>
<tr>
<td>num&lt;0</td>
<td>Failed to retrieve the property.</td>
</tr>
</tbody>
</table>

Example
$iRetVal=COMGetProperty("objTest", "Long", $LongVal);

ConnectorPage

Syntax
ConnectorPage();
Script functions in alphabetical order

Script functions reference

**Description**

Returns the number of pages (in the current job) that have been processed so far on the current connector. This function replaces the `QueuePage` function.

**Returns**

A number.

**Example**

```plaintext
$cp=connectorpage();
log(1,"Number of pages: "+$cp);
```

**ConnectorPageActual**

**Syntax**

`ConnectorPageActual();`

**Description**

Returns the number of logical pages sent to the output connector, including inserts but excluding skipped pages.

**Returns**

<table>
<thead>
<tr>
<th>num</th>
<th>The number of logical pages.</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Error.</td>
</tr>
</tbody>
</table>

**Example**

```plaintext
$page_count = ConnectorPageActual();
Result:
$page_count = 5
```

**ConnectorPages**

**Syntax**

`ConnectorPages();`

**Description**

Returns the total number of pages (in the current job) that will be processed on the current connector. This function replaces the `QueuePages` function.

**Note:** This function cannot be used in the pre-process phase.

**Returns**

A number.
Example
$cp=connectorpages();
if(num($cp)<10)
{
   $text="Smaller than 10";
}

ConvCurrMsgToUC

Syntax
ConvCurrMsgToUC(codepage, text_order);

<table>
<thead>
<tr>
<th>codepage</th>
<th>A string (case sensitive) that specifies the code page used to encode input data.</th>
</tr>
</thead>
<tbody>
<tr>
<td>text_order</td>
<td>0 (logical order) or 1 (visual order). Specifies whether bidirectional input text is logically or visually ordered. Set this argument to 1 only if the input data contains visually ordered bidirectional text (Arabic or Hebrew).</td>
</tr>
</tbody>
</table>

Description
Converts the encoding of input data from the specified code page to UCS-2. You can use ConvCurrMsgToUC under the following conditions:

- The incoming data is represented by single byte characters.
- The script is triggered as a Retrieved script.
- No code page is specified for the input connector. If you specify a code page for the input connector, ConvCurrMsgToUC will be ignored.

Returns
A number indicating whether the data was successfully converted.

| 1 | OK. The data was successfully converted. |
| 0 | Failed. The data was NOT successfully converted. |

Example
ConvCurrMsgToUC("ISO 8859-1", 0);

CopyFile

CopyFile is an old name for the FileCopy function.
CreateGlobalSerNo

**Syntax**

CreateGlobalSerNo(key, initial, min, max);

<table>
<thead>
<tr>
<th>key</th>
<th>A string uniquely identifying a serial number key, representing for example a document type.</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial</td>
<td>The initial value of the serial number. The first call to GetGlobalSerNo or GetGlobalSerNoRange will return this number, for subsequent calls it will be incremented by one. See GetGlobalSerNo on page 164, and GetGlobalSerNoRange on page 165.</td>
</tr>
<tr>
<td>min</td>
<td>The minimum value of the created serial numbers. Specify -1 to not use a minimum value. If a maximum value is specified and if that maximum value is reached, the serial number counter will start again from the minimum value. If no minimum value is used, the counter will start again from 0.</td>
</tr>
<tr>
<td>max</td>
<td>The maximum value of the created serial numbers. If this value is reached, the counter will start again from the minimum serial number or 0, depending on the min value. Specify -1 to not use a maximum value.</td>
</tr>
</tbody>
</table>

**Description**

Creates a new serial number key in the repository. This key can be accessed by all StreamServer applications running in a specific application domain.

**Returns**

A number indicating whether the serial number key was created successfully or not, or if it already exists.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The serial number key was created successfully.</td>
</tr>
<tr>
<td>1</td>
<td>The serial number key already exists in the repository,</td>
</tr>
<tr>
<td>-1</td>
<td>Error.</td>
</tr>
</tbody>
</table>

**Example**

createglobalserno("invoices", 1000, 1000, 2000);  
This function creates the following serial number key entry in the repository table:  
invoices 1000 1000 2000  
Where:  
First column – the key  
Second column – initial value
Third column – minimum value
Fourth – maximum value

**Note:** The min/max columns can be omitted by specifying -1 for their arguments.

### CurrDocProperty

**Syntax**

```plaintext
CurrDocProperty(propertyName);
```

<table>
<thead>
<tr>
<th><code>propertyName</code></th>
<th>The document property to return. Can be one of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ppdocid</td>
<td>the Post-processor document ID.</td>
</tr>
<tr>
<td>ppjobid</td>
<td>the Post-processor job ID.</td>
</tr>
<tr>
<td>status</td>
<td>the document status.</td>
</tr>
<tr>
<td>priority</td>
<td>the document priority.</td>
</tr>
<tr>
<td>creationtime</td>
<td>the document creation time.</td>
</tr>
<tr>
<td>procsetime</td>
<td>the document process time.</td>
</tr>
<tr>
<td>error</td>
<td>the document error.</td>
</tr>
<tr>
<td>pages</td>
<td>the number of logical pages in the document.</td>
</tr>
<tr>
<td>resrepository</td>
<td>the resource repository.</td>
</tr>
<tr>
<td>resserver</td>
<td>the resource repository server.</td>
</tr>
</tbody>
</table>

**Description**

Returns the specified document property for the current document.

**Returns**

The specified document property.

**Example**

```plaintext
$docid = CurrDocProperty("ppdocid");
```

Result:

```plaintext
$docid = 123
```

### CurrInFileName

**Syntax**

```plaintext
CurrInFileName();
```

**Description**

Returns the name of the input file, if there is one, else an empty string.
Enables the transfer of file information to the StreamServer where the file name can hold information about which Event to activate, printer, and number of copies. A file could have the following name:

HP_OP_PL1.ORDERSW.3.DSI

In this file name, the first portion (up to the first period) specifies a printer, the second specifies which Event to be activated, and the third specifies the number of copies to be printed.

This function can also be used, for example, on a flyleaf to display the file that has been printed.

See also CurrInRealPath.

**Returns**
A string.

**Example**

```plaintext
$file=CurrInFileName();
```

**CurrInRealPath**

**Syntax**

```plaintext
CurrInRealPath();
```

**Description**

Returns the temporary file name of the input file, if there is one.
If a file is found when scanning a directory, the file is renamed with a temporary file name. The temporary file name is the one that is used in StreamServe. The function cannot be used for queues.

See also CurrInFileName.

**Returns**
A string.

**Example**

```plaintext
$temp_file=CurrInRealPath();
```

**CurrInSavePath**

**Syntax**

```plaintext
CurrInSavePath();
```
Description
When using a Directory input connector, you can copy the input files to backup directory. You use the Save Path connector setting to specify the backup directory. All files are given unique names when copied to the backup directory. The CurrInSavePath function is used to retrieve the new file names.

Returns
A string.

Example
Specify the following Save Path for the Directory input connector:
C:\safekeep
Then use the CurrInSavePath function to retrieve the new file name:
$keptfile = CurrInSavePath();
log (1, "SavePath" + $keptfile);
$keptfile may contain for example
"C:\safekeep\ds2A4FE1D09210310251010018510.bak"

CurrJobOwner

Syntax
CurrJobOwner();

Description
Returns the current job owner. See also SetJobOwner.

Returns
A string.

Example
$owner=CurrJobOwner();
CurrJobProperty

Syntax

CurrJobProperty(propertyName);

<table>
<thead>
<tr>
<th>propertyName</th>
<th>The job property to return. Can be one of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• ppjobid – the Post-processor job ID.</td>
</tr>
<tr>
<td></td>
<td>• status – the job status.</td>
</tr>
<tr>
<td></td>
<td>• priority – the job priority.</td>
</tr>
<tr>
<td></td>
<td>• creationtime – the job creation time.</td>
</tr>
<tr>
<td></td>
<td>• procsertime – the job process time.</td>
</tr>
<tr>
<td></td>
<td>• error – the job error.</td>
</tr>
<tr>
<td></td>
<td>• pages – the number of logical pages in the job.</td>
</tr>
<tr>
<td></td>
<td>• name – the name of the job.</td>
</tr>
<tr>
<td></td>
<td>• strsid – the StreamServe job ID.</td>
</tr>
<tr>
<td></td>
<td>• user – the user who submitted the job.</td>
</tr>
<tr>
<td></td>
<td>• description – the description of the job.</td>
</tr>
</tbody>
</table>

Description

Returns the specified job property. The function can be used at the following post-processing scripting levels:

Before and after document, process and page.

Returns

The specified job property.

Example

$job_id = CurrJobProperty("ppjobid");

Result:

$job_id = 1
CurrMetadata

Syntax

CurrMetadata(metadataName);

<table>
<thead>
<tr>
<th>metadataName</th>
<th>A metadata name connected to the current document. You can use metadata specified in the Project or the following predefined metadata:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• MailMach – the mailing machine used for current document.</td>
</tr>
<tr>
<td></td>
<td>• EnvelNr – the envelope number for the current document.</td>
</tr>
<tr>
<td></td>
<td>• NumSheets – the number of sheets in the current document.</td>
</tr>
<tr>
<td></td>
<td>• NumEnvelopes – the number of envelopes for the current document.</td>
</tr>
<tr>
<td></td>
<td>• NumInserts – the number of inserts for the current document.</td>
</tr>
<tr>
<td></td>
<td>• NumInsertEquival – the number of insert equivalents, i.e. the number of normal sheets that the weight of the inserts</td>
</tr>
<tr>
<td></td>
<td>corresponds to.</td>
</tr>
<tr>
<td></td>
<td>• ActualInserts – the insert numbers assigned to the current document.</td>
</tr>
<tr>
<td></td>
<td>• SplitPositions – the page numbers of the first page in each part of the document when it is split into more than one</td>
</tr>
<tr>
<td></td>
<td>envelope.</td>
</tr>
</tbody>
</table>

Description

Returns the value of the specified metadata for the current document. The function can be used at the following post-processing scripting levels:

Before and after document, process and page.

Returns

The value (string) of the specified metadata. If the specified metadata is not found, an empty string is returned.

Example

```$insertsActual = CurrMetadata("ActualInserts");
Result;
$insertsActual ="3;6;7"
$num_of_tokens=StrTok($insertsActual,",",$arr1);
Result:
arr1[0]=3
arr1[1]=6
arr2[2]=7```
CurrReposProperty

Syntax

CurrReposProperty(propertyName);

<table>
<thead>
<tr>
<th>propertyName</th>
<th>The name of the repository property. Can be one of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td>server</td>
<td>the FO server name.</td>
</tr>
<tr>
<td>dbname</td>
<td>the FO database name.</td>
</tr>
<tr>
<td>physicalserver</td>
<td>the physical server for the FO server.</td>
</tr>
<tr>
<td>dblogicalname</td>
<td>the database logical name.</td>
</tr>
</tbody>
</table>

Description

Returns the specified repository property.

Returns

The specified repository property.

Example

$db_server = CurrReposProperty("server");

D

DayOfWeek

Syntax

DayOfWeek([yy] yymmdd);

<table>
<thead>
<tr>
<th>[yy] yymmdd</th>
<th>A date string.</th>
</tr>
</thead>
</table>

Description

Returns the number of the day in a week. Monday returns 1 and Sunday returns 7.

Returns

A number (string) between 1-7.

Example

$a="19981209";
$b=DayOfWeek($a);
Result: $b="3"
DayOfYear

Syntax
DayOfYear([yy]yymmdd);

[yy]yymmdd | A date string.

Description
Returns the number of the day in a year (1-365) where 1 is January 1st.

Returns
A number (string) between 1-365.

Example
$a="980212";$
$b=dayofyear($a);
Result: $b="43"

DeclareMetadata

Syntax
DeclareMetadata(metadataName[, type, sort_order, behaviour, defaultValue]);

<table>
<thead>
<tr>
<th>metadataName</th>
<th>A string specifying the metadata name.</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>A string specifying one of the following types:</td>
</tr>
<tr>
<td></td>
<td>S – String (default)</td>
</tr>
<tr>
<td></td>
<td>N – Numeric</td>
</tr>
<tr>
<td>sort_order</td>
<td>A string specifying sorting order if metadata is used to sort documents. You can specify one of the following:</td>
</tr>
<tr>
<td></td>
<td>A – Ascending (default)</td>
</tr>
<tr>
<td></td>
<td>D – Descending</td>
</tr>
<tr>
<td>behaviour</td>
<td>A string specifying the result of the DeclareMetadata function if metadata is not found for the documents. You can specify one of the following:</td>
</tr>
<tr>
<td></td>
<td>F – Fail (default)</td>
</tr>
<tr>
<td></td>
<td>E – Empty</td>
</tr>
<tr>
<td></td>
<td>D – Default. See defaultValue argument.</td>
</tr>
<tr>
<td>defaultValue</td>
<td>Specifies the default value used when behaviour is set to default.</td>
</tr>
</tbody>
</table>
Description
Declares metadata that can be used in post-processor scripting. If the specified metadata exists in the post-processor repository, the metadata is retrieved with the documents when they are retrieved from the repository. You can use the declared metadata to get and set metadata on documents.

This function can only be used when enveloping and/ or sorting is enabled. The function must be used at Job Begin scripting level. You cannot use it at the post-processing Job Begin scripting level.

Returns

<table>
<thead>
<tr>
<th>num&lt;0</th>
<th>Error</th>
</tr>
</thead>
</table>

Example
Job Begin script:
```
$ret = DeclareMetadata("zip", "N", "A");
```

Post-processor Job Begin script:
```
$doc_id = GetFirstSegDoc();
while(num($doc_id) > 0)
{
    $zip = 41260
    SetSegMetaData($doc_id, "zip", $zip);
    $odc_id = GetNextSegDoc($doc_id);
}
$ret = SortSegDoc("zip");
```

DeleteFile
DeleteFile is an old name for the FileDelete function.

DeleteJobResource

Syntax
DeleteJobResource(num_jobID, str_name, num_index);

<table>
<thead>
<tr>
<th>num_jobID</th>
<th>The ID of the job in which the LXF job resource was created.</th>
</tr>
</thead>
<tbody>
<tr>
<td>str_name</td>
<td>The name of the job resource you want to delete. If str_name is empty, all job resources within the specified job will be deleted.</td>
</tr>
</tbody>
</table>
**DeleteJobResource**

**Syntax**

```plaintext
DeleteJobResource (job_id, resource_name, index);
```

**Description**

Marks job resources to be removed. The marked resources will be deleted when all references to the job resource have been released. This function is not supported in Post-processor scripting.

See also the *Job Resource output connector settings* and the `OutputLXFJobResource` and `GetJobResourceIndex` functions.

**Returns**

A number indicating whether the job resources have been successfully marked and will be deleted.

<table>
<thead>
<tr>
<th>num_index</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;0</td>
<td>OK. The job resources have been successfully marked and will be deleted.</td>
</tr>
<tr>
<td>0</td>
<td>Failure.</td>
</tr>
</tbody>
</table>

**Example**

To delete a job resource created in job 122, named `MyLXFResourceConnector`, with index 0:

```plaintext
DeleteJobResource (122, "MyLXFResourceConnector", 0);
```

To delete all job resources created in job 122:

```plaintext
DeleteJobResource (122, ",", -1);
```

To delete all job resources named `MyLXFResourceConnector` created in job 122:

```plaintext
DeleteJobResource (122, "MyLXFResourceConnector", -1);
```

**DelSubstKey**

**Syntax**

```plaintext
DelSubstKey(tbl_file, str_key);
```

**Description**

Deletes an entry specified by `str_key` from the substitution table `tbl_file`. It does not affect the physical file stored on disk, only the representation of the table in the memory.

<table>
<thead>
<tr>
<th>tbl_file</th>
<th>A string specifying the path and filename of the table.</th>
</tr>
</thead>
<tbody>
<tr>
<td>str_key</td>
<td>The key of the entry to delete.</td>
</tr>
</tbody>
</table>
Script functions in alphabetical order

Script functions reference

Returns
A number indicating whether the entry was deleted successfully.

<table>
<thead>
<tr>
<th></th>
<th>Entry deleted successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Example
In this example, the following substitution table (author.tbl) is used.

```plaintext
//!CodePage UTF8!
//!multicolumn!
agatha christie 1890
alfred hitchcock 1899
```

The following script deletes an entry from the physical substitution table stored on disk (author.tbl):

```plaintext
DelSubstKey("../data/tables/author.tbl","alfred");
WriteSubst("../data/tables/author.tbl");
```

After the script is executed, author.tbl is changed as follows:

```plaintext
//!CodePage UTF8!
//!multicolumn!
agatha christie 1890
```

Notes
- A substitution table is kept in memory until the script function `EraseSubst` is called for that table.

Dformat

Syntax

```plaintext
Dformat(str_dateformat, [yy]yymmdd);
```

<table>
<thead>
<tr>
<th>str_dateformat</th>
<th>A string specifying the date format.</th>
</tr>
</thead>
<tbody>
<tr>
<td>[yy]yymmdd</td>
<td>A number specifying the a date, where:</td>
</tr>
<tr>
<td></td>
<td>• yyyy – year in four digits, e.g. 2005.</td>
</tr>
<tr>
<td></td>
<td>• yy – year in two digits, e.g. 05.</td>
</tr>
<tr>
<td></td>
<td>• mm – month.</td>
</tr>
<tr>
<td></td>
<td>• dd – day of the month</td>
</tr>
</tbody>
</table>
Description
Formats a date according to the specified date format. If the date is invalid, no formatting is done, and an empty string is returned.

Returns
A formatted date.

Example
$d = dformat("mon mm year yyy", 980923);
Result: $d="mon 09 year 1998"

DiffDate
Syntax
DiffDate([yy]yymmdd, [yy]yymmdd);

Description
Calculates the number of days between two dates.

Returns
A string indicating the number of days.

Note: If the second date is later than the first date, the result is negative.

Example
$a="19980512";
$b="980427";
$c=diffdate($a,$b);
Result: $c="15"

Div
Syntax
Div(num_1, num_2)

d_{1} | A numeric expression.
d_{2} | A numeric expression.

Description
Divides the arguments and returns the floor of the result of that division.
Returns
A number.

Example
div(106,10)
Result: 10
div(-1,2)
Result: -1
div(-106,10)
Result: -11

DocInsertedPages

Syntax
DocInsertedPages();

Description
Returns the number of pages inserted in the output queue within the current
document. Pages are inserted either by using the InsertPage() function, or by
adding an address sheet when splitting the document into more than one envelope.
The function can be used at the following post-processing scripting levels:
Before and after document, process and page.

Returns

<table>
<thead>
<tr>
<th>num</th>
<th>The number of pages inserted.</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Error</td>
</tr>
</tbody>
</table>

Example
$inserted_pages = DocInsertedPages();

DocPageActual

Syntax
DocPageActual();

Description
Returns the number of document pages sent to the output connector, including
inserts but excluding skipped pages. The function can be used at the following
post-processing scripting levels:
Before and after document, process and page.
Returns

<table>
<thead>
<tr>
<th>num</th>
<th>The number of pages sent to the output connector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Error</td>
</tr>
</tbody>
</table>

Example

```php
$page_count = DocPageActual();
```

Result:
```
$page_count = 5
```

DocPage

**Syntax**

```php
DocPage();
```

**Description**

Returns the current page number of the current document.

**Returns**

<table>
<thead>
<tr>
<th>num</th>
<th>The current page number.</th>
</tr>
</thead>
</table>

Example

```php
$page = docpage();
```

DocPages

**Syntax**

```php
DocPages();
```

**Description**

Returns the total number of pages in the current document. This function cannot be used in the pre-process phase.

**Returns**

A number.

**Example**

```php
$pages_total = docpages();
```
DtisoFormat

Syntax

DtisoFormat(source, format, timezone);

<table>
<thead>
<tr>
<th>source</th>
<th>A string containing the date and time.</th>
</tr>
</thead>
<tbody>
<tr>
<td>format</td>
<td>A string specifying the format of the source string.</td>
</tr>
<tr>
<td></td>
<td>• For date only, a simple string is used, for example</td>
</tr>
<tr>
<td></td>
<td>&quot;MM/DD/YYYY&quot; if source string is &quot;09/15/2007&quot;, or</td>
</tr>
<tr>
<td></td>
<td>&quot;DD-MM-YY&quot; if source string is &quot;15-09-07&quot;</td>
</tr>
<tr>
<td></td>
<td>• For date and time, the following format must be used:</td>
</tr>
<tr>
<td></td>
<td>&quot;{date&lt;date&gt;{&lt;internediate character}time&lt;time&gt;}&quot;</td>
</tr>
<tr>
<td></td>
<td>where</td>
</tr>
<tr>
<td></td>
<td>• &lt;date&gt; is the date format of the source string and</td>
</tr>
<tr>
<td></td>
<td>contains YYYY or YY for the year, MM for the month,</td>
</tr>
<tr>
<td></td>
<td>and DD for the day.</td>
</tr>
<tr>
<td></td>
<td>• &lt;time&gt; is the time format of the source string and</td>
</tr>
<tr>
<td></td>
<td>contains HH for the hour and MM for the minute.</td>
</tr>
<tr>
<td></td>
<td>• &lt;internediate character&gt; is the separator</td>
</tr>
<tr>
<td></td>
<td>between the date and time, for example blank space.</td>
</tr>
<tr>
<td></td>
<td>For example:</td>
</tr>
<tr>
<td></td>
<td>&quot;{date{MM/DD/YYYY}{ }time{HH:MM}}&quot; if source</td>
</tr>
<tr>
<td></td>
<td>string is &quot;09/15/2007 12:30&quot;</td>
</tr>
<tr>
<td>timezone</td>
<td>A string containing timezone (e.g. &quot;+01:00&quot; or &quot;-01:00&quot;). A</td>
</tr>
<tr>
<td></td>
<td>return value including both date and time will be shifted</td>
</tr>
<tr>
<td></td>
<td>according to the specified timezone. If you do not want</td>
</tr>
<tr>
<td></td>
<td>to specify the timezone, you can specify an empty string, &quot;&quot;.</td>
</tr>
<tr>
<td></td>
<td>If you specify a timezone value for date conversion only, it will</td>
</tr>
<tr>
<td></td>
<td>be ignored.</td>
</tr>
</tbody>
</table>

Description

Converts date and optionally time to an ISO 8601 format which is required in the StreamServe repositories.

Returns

A date and time formatted according to ISO 8601 standard.

Example

$formatteddatetime = dtISOFormat("24-06-2007 09:30", "{date{DD-MM-YYYY}{ }time{HH:MM}}", "+01:00"

Returns:
DumpVariables

**Syntax**

DumpVariables(file_name);

| file_name | A string specifying the name of the file to create. A file path can be specified.  
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Note:</strong></td>
<td>If a file path you specify does not exist no file will be created.</td>
</tr>
</tbody>
</table>

**Description**

Dumps all variables and their values to a specified file. All variables that have a value assigned to them at the moment of script function execution are dumped. You can invoke the function at any level of execution. If the file already exists, the new data is appended to the existing file.

**Returns**

<table>
<thead>
<tr>
<th>value</th>
<th>OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**Example**

$dump_var = DumpVariables("C:\mylogs\dumped_variables.txt");

EnableEnvelopeCheck

**Syntax**

EnableEnvelopeCheck(check_arg, enable_arg);

| check_arg | A string specifying the checks to be enabled or disabled:  
| --- | --- |
| • all – all checks are enabled or disabled.  
| • sl – enables or disables checks for sheet layout change.  
| • ip – enables or disables checks for InsertPage() script function on page or process level. |

$formatteddatetime = "2007-06-24T08:30:00"
enable_arg  A number specifying if the check is enabled or disabled.
   •  1 – checks specified by check_arg are enabled.
   •  0 – checks specified by check_arg are disabled.

**Description**
Enables or disables a check for an inconsistent number of sheets when using enveloping. The number of sheets can be inconsistent for example if
- you are using post-processing scripting to for example change the sheet layout by changing a variable on which the sheet layout depends,
- the number of inserts are changed via an OMR change, or
- if extra pages are inserted using the `InsertPage()` function.

**Returns**
A number indicating whether a check is set.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Check is set.</td>
</tr>
<tr>
<td>num&lt;&gt;0</td>
<td>Error</td>
</tr>
</tbody>
</table>

**Example**

EnableEnvelopeCheck("ip", 0);"
Description
Used in a Retrieved script to set Event Order Last for the Event. The Event will be appended as the last Event before the Message is sent for processing. If there is no previous Event, this Event will be the only one in the Message.

Returns
N/A

Example
if ($invoiceno != $oldinvoiceno)
{
    EndMessage();
}
$oldinvoiceno = $invoiceno;

EraseSubst

Syntax
EraseSubst(tbl_file);

| tbl_file | A string specifying the path and filename of the substitution table. |

Description
Removes the specified substitution table from memory. It does not affect the physical file stored on disk, only the representation of the table in the memory.

The EraseSubst function is similar to ClearSubst:
- EraseSubst removes the table from the memory.
- ClearSubst clears the table contents but keeps the actual table in the memory.

A substitution table is kept in memory until EraseSubst is called for that table. This means StreamServer performance will be affected if several tables are created in a job, and if the tables are never removed from memory during the job using the EraseSubst function. For example, if you use variables in the table names, you must make sure there are a limited number of tables in the memory. In this case you must call EraseSubst for each table after it is used.

If you use ClearSubst, and then access the substitution table with for example GetSubst, an empty string will be returned. If you access the table after using EraseSubst, the table will be re-read from disk, and a value will be returned.
Returns

<table>
<thead>
<tr>
<th></th>
<th>Table removed</th>
</tr>
</thead>
</table>

Example

In this example, the ID of the previous document is copied to the current document, and the substitution table of the previous document is removed from memory and from disk. The variable $documentID is defined in the Event. Its value comes from the input data. The variable $prevDocumentID is the ID of the previous document.

\[
\begin{align*}
$prevSubTable &= "document_" + $prevDocumentID + ".tbl"; \\
$subTable &= "document_" + $documentID + ".tbl"; \\
$prevDocumentType &= GetSubst ($prevDocumentName, "type", 0); \\
$documentType &= $prevDocumentType; \\
ReadSubst ($subTable); \\
SetSubst ($subTable, "type", 0, $documentType); \\
WriteSubst ($subTable); \\
EraseSubst ($prevSubTable); \\
FileDelete ($prevSubTable);
\end{align*}
\]

Eval

Syntax

Eval(str);

<table>
<thead>
<tr>
<th>str</th>
<th>A string containing one or several variables.</th>
</tr>
</thead>
</table>

Description

Evaluates all variables in the argument string.

Returns

<table>
<thead>
<tr>
<th>str</th>
<th>A string in which all variables have been replaced by their values.</th>
</tr>
</thead>
</table>

Example

\[
\begin{align*}
$question &= "life, the universe, and everything"; \\
$answer &= "42"; \\
$result &= Eval ("The answer to the question of $question is $answer"); \\
$result \text{ will then contain} \\
"The answer to the question of life, the universe, and everything is 42" \end{align*}
\]
Execute

Syntax

Execute(command, timeout, input, output);

<table>
<thead>
<tr>
<th>command</th>
<th>The command to execute.</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeout</td>
<td>Timeout in seconds.</td>
</tr>
<tr>
<td></td>
<td>StreamServer will wait the specified amount of time before it tries to shutdown the called process and returns to operation.</td>
</tr>
<tr>
<td></td>
<td>Timeout set to -1 means StreamServer will “wait forever”. This is the default value.</td>
</tr>
<tr>
<td></td>
<td>Timeout set to 0 means output is not returned, and StreamServer will not wait for a result. The external process is expected to either terminate when it is finished, or continue “forever”.</td>
</tr>
<tr>
<td>input</td>
<td>Input data to send to the called process.</td>
</tr>
<tr>
<td>output</td>
<td>Array of three output variables:</td>
</tr>
<tr>
<td></td>
<td>• output[0] – STDOUT output from the called process.</td>
</tr>
<tr>
<td></td>
<td>• output[1] – STDERR output from the called process.</td>
</tr>
<tr>
<td></td>
<td>• output[2] – Exit code from the called process.</td>
</tr>
</tbody>
</table>

Description

This script function executes an external program, script, bat file or process.

Note: This script function has the same security context as the one defined for the StreamServer application (Log on setting defined for the StreamServer application). For example, if Log on as: Local System account is defined for the StreamServer application, this script function can only be executed on the local computer.
Returns

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OK.</td>
</tr>
<tr>
<td>1</td>
<td>Command not found.</td>
</tr>
<tr>
<td>2</td>
<td>Timeout reached.</td>
</tr>
<tr>
<td>3</td>
<td>Maximum number of spawned process reached.</td>
</tr>
<tr>
<td>4</td>
<td>Other error.</td>
</tr>
</tbody>
</table>

Examples

**Example 54** Calling an external Java application to download document

```
$command = eval("java -cp <22>..\data<22> FileDownloader $link $1Path");
$return = execute($command, 120, ",", $outvar);
$output = $outvar[0];
$error = $outvar[1];
$exitcode = $outvar[2];
```

Exist

**Syntax**

```
Exist(str);
```

**str** A string specifying a path.

**Description**

Checks if a directory or file exists. The chain of directories is checked directly, i.e. session variables are not employed by the `Exist` function.

**Returns**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>True</td>
</tr>
<tr>
<td>0</td>
<td>False</td>
</tr>
</tbody>
</table>

**Examples**

```
Exist("u/bin");
Exist("c:/data/forms");
Exist("c:/tables/abc.txt");
```

If the path exists, the result is True (1). If the path does not exist, the result is False (0).
**FileClose**

**Syntax**

```plaintext
FileClose(filename);
```

| `filename` | A string specifying the path and name of the file.

**Description**

Closes the specified file. See also `IoErrText`.

**Returns**

A number indicating whether the file was successfully closed.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The file was successfully closed.</td>
</tr>
<tr>
<td>&gt;0</td>
<td>Failure. This value can be sent to <code>IoErrText</code> which will return an error text for the failure.</td>
</tr>
</tbody>
</table>

**Example**

```plaintext
//Close myfile.txt and store the returned value in $err

$err = fileclose("myfile.txt");

//If the action was unsuccessful, pass the error value
//to ioerrtext and store the error text returned by
//ioerrtext in $errtext.
//Send the error text to the log file

if(num($err)!=0)
{
    $errtext=IoErrText($err);
    log(0,"Errortext "+$errtext);
}
```

**FileCopy**

**Syntax**

```plaintext
FileCopy(sourcefile, dest_file);
```

<table>
<thead>
<tr>
<th><code>sourcefile</code></th>
<th>A string specifying the path and name of the source file.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dest_file</code></td>
<td>A string specifying the path and name of the destination file</td>
</tr>
</tbody>
</table>

---

**Description**  
Copies the specified file, and leaves the original file unchanged. See also IoErrText.

**Returns**  
A number indicating whether the file was successfully copied.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The file was successfully copied.</td>
</tr>
<tr>
<td>&gt;0</td>
<td>Failure. This value can be sent to IoErrText which will return an error text for the failure.</td>
</tr>
</tbody>
</table>

**Example**

```javascript
//Copy the contents of myfile.txt to mycopy.txt  
//and store the returned value in $err  

$err = filecopy("myfile.txt", "mycopy.txt");

//If the action was unsuccessful, pass the error value  
//to ioerrtext and store the error text returned by  
//ioerrtext in $errtext.  
//Send the error text to the log file

if(num($err)!=0)
{
   $errtext=ioerrtext($err);
   log(0,"Errortext "+$errtext);
}
```

**FileDelete**

**Syntax**

`FileDelete(filename);`

| filename | A string specifying the path and name of the file. |

**Description**  
Deletes the specified file. See also IoErrText.

**Returns**  
A number indicating whether the file was successfully deleted.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The file was successfully deleted.</td>
</tr>
<tr>
<td>&gt;0</td>
<td>Failure. This value can be sent to IoErrText which will return an error text for the failure.</td>
</tr>
</tbody>
</table>
Example

//Delete myfile.txt and store the returned value in $err

$err = filedelete("myfile.txt");

//If the action was unsuccessful, pass the error value
//to ioerrtext and store the error text returned by
//ioerrtext in $errtext.
//Send the error text to the log file

if(num($err)!={0)
{
    $errtext=ioerrtext($err);
    log(0,"Errortext "+$errtext);
}

FileMove

Syntax

FileMove(sourcefile, dest_file);

| sourcefile  | A string specifying the path and name of the source file. |
| dest_file   | A string specifying the path and name of the destination file. |

Description

Moves the specified file from one location to another (the original file is deleted).
See also IoErrText.

Returns

A number indicating whether the file was successfully moved.

<table>
<thead>
<tr>
<th></th>
<th>The file was successfully moved.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>&gt;0</td>
<td>Failure. This value can be sent to IoErrText which will return an error text for the failure.</td>
</tr>
</tbody>
</table>

Example

//Move the contents of myfile.txt to mynewfile.txt
//and store the returned value in $err

$err = filemove("myfile.txt","mynewfile.txt");

//If the action was unsuccessful, pass the error value
//to ioerrtext and store the error text returned by
//ioerrtext in $errtext.
//Send the error text to the log file
if(num($err)!=0) {
    $errtext=ioerrtext($err);
    log(0,"Errortext "+$errtext);
}

FileOpen

Syntax
FileOpen(filename, mode);

<table>
<thead>
<tr>
<th>filename</th>
<th>A string specifying the path and name of the file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>A string specifying the mode of opening:</td>
</tr>
<tr>
<td></td>
<td>• w – Opens an empty file for writing. If the given file exists, its contents are destroyed.</td>
</tr>
<tr>
<td></td>
<td>• a – Opens for writing at the end of the file (appending) without removing the EOF marker before writing new data to the file; creates the file first if it does not exist.</td>
</tr>
<tr>
<td></td>
<td>• r – Opens for reading. If the file does not exist or cannot be found, the call fails.</td>
</tr>
<tr>
<td></td>
<td>• b – Open in binary mode; translations involving carriage-return and line-feed characters are suppressed.</td>
</tr>
</tbody>
</table>

Description
Opens the specified file for reading or writing according to mode specified. See also IoErrText.

Returns
A number indicating whether the file was successfully opened.

| 0       | The file was successfully opened. |
| >0      | Failure. This value can be sent to IoErrText which will return an error text for the failure. |

Example
//Open myfile.txt for writing and store the returned value in $err

$err = fileopen("D:\documents\myfile.txt", "w");

//If the action was unsuccessful, pass the error value to ioerrtext and store the error text returned by ioerrtext in $errtext.
//Send the error text to the log file
if(num($err)!==0)
{
    $errtext=ioerrtext($err);
    log(0,*Errortext *+$errtext);
}

FileReadLn

**Syntax**

FileReadLn(filename, var);

| filename | A string specifying the path and name of the file. |
| var      | A string specifying the variable in which to store the data. |

**Description**

Reads a line from the specified file until a new line or an end of file is reached, and returns the line to a specified variable. The new line character is not included in the result. You must open the file to be read with FileOpen before calling FileReadLn. Use the FileClose function to close the file after reading.

**Note:** Using FileReadLn with a file previously written by FileWrite in a job might cause errors due to the pre-processing phase (for more information on the pre-processing and processing phases, see *Formatting phase* on page 19).

**Returns**

A number indicating whether the line was successfully read.

| 0 | The line was successfully read. |
| >0 | Failure. This value can be sent to IoErrText which will return an error text for the failure. |

**Example**

```plaintext```
while(fileredLn(".ile.txt", $inbuf) = 0){}
```

Reads from file.txt until EOF is reached.

FileSize

**Syntax**

FileSize(filename);

| filename | A string specifying the path and name of the file. |
Description
Returns the size of the specified file.

Returns
The file size in bytes.

Example
discount = FileSize("c:\my_file.txt");

FileStream

Syntax
FileStream(filename, str_data);

| filename | A string specifying a path and name of the file. |
| str_data | A string specifying the data to be written to the file. |

Description
Writes data to the specified file.

Returns
A number indicating whether data was successfully written to the file.

| 0          | The data was successfully written to the file. |
| >0         | Failure. This value can be sent to IoErrText which will return an error text for the failure. |

Example

//Write the contents of $utbuf to myfile.txt
//and store the returned value in $err

$err = filewrite("myfile.txt",$utbuf);

//If the action was unsuccessful, pass the error value
//to ioerrtext and store the error text returned by
//ioerrtext in $errtext.
//Send the error text to the log file

if(num($err)!=0)
{
    $errtext=ioerrtext($err);
    log(0,"Errortext "+$errtext);
}
Script functions in alphabetical order

Script functions reference

FileWriteLn

Syntax

FileWriteLn(filename, str_data);

<table>
<thead>
<tr>
<th>filename</th>
<th>A string specifying a path and name of the file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>str_data</td>
<td>A string specifying the data to be written to the file.</td>
</tr>
</tbody>
</table>

Description

Writes data to the specified file, and adds a new line character at the end. You must open the file with FileOpen before calling FileWriteLn, and use the FileClose function to close the file.

Returns

A number indicating whether data was successfully written to the file.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The data was successfully written to the file.</td>
</tr>
<tr>
<td>&gt;0</td>
<td>Failure. This value can be sent to IoErrText which will return an error text for the failure.</td>
</tr>
</tbody>
</table>

Example

//Write the contents of $utbuf to myfile.txt, //add a new line, and store the returned value in $err

$err = filewriteln("myfile.txt",$utbuf);

//If the action was unsuccessful, pass the error value //to ioeerrtext and store the error text returned by //iocerrtext in $errtext. //Send the error text to the log file

if(num($err)!=0)
{
   $errtext=iocerrtext($err);
   log(0,"Errortext "+$errtext);
}

FindInArray

Syntax

FindInArray(array_name, str);

<table>
<thead>
<tr>
<th>array_name</th>
<th>A string specifying the name of the array.</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>A variable or a string specifying the value to search for.</td>
</tr>
</tbody>
</table>
Description
Searches for a value in an array. Returns the position of the array element that contains the specified value or superset of the specified value.
For example if you specify str value "10", the position of "510" is returned if it is found first.

Returns

<table>
<thead>
<tr>
<th>num</th>
<th>An integer specifying the position of the array element.</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Failure.</td>
</tr>
</tbody>
</table>

Example
This example has two arrays:
- $arrTele containing phone numbers.
- $arrAmount containing call charges.

The FindInArray function is used to search for the phone number in $arrTele. If a phone number for a specific call is found in $arrTele, the charge for the call should be added to the corresponding position in $arrAmount. If the phone number is not found, the new phone number should be added to $arrTele and the new charge to $arrAmount. The ArraySize function specifies the position where to add the new number and charge at the end of the arrays.

```plaintext
$retIndex=FindInArray($arrTele, $noDialled);
if ($retIndex=-1)
{
    $retSize=ArraySize($arrTele);
    $arrTele[$retSize]=$noDialled;
    $arrAmount[$retSize]=$callCharge;
}
else
    $arrAmount[$retIndex]=$callCharge;
```

FindStringExactInArray

**Syntax**
FindStringExactInArray(array_name, str);

<table>
<thead>
<tr>
<th>array_name</th>
<th>A string specifying the name of the array.</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>A variable or a string specifying the value to search for.</td>
</tr>
</tbody>
</table>
Description
Searches for a value in an array. Returns the position of the array element that contains exactly the specified value.

Returns

<table>
<thead>
<tr>
<th>num</th>
<th>An integer specifying the position of the array element.</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Failure.</td>
</tr>
</tbody>
</table>

Example
This example has two arrays:
- $arrTele containing phone numbers.
- $arrAmount containing call charges.

The FindStringExactInArray function is used to search for the phone number in $arrTele.

If a phone number for a specific call is found in $arrTele, the charge for the call should be added to the corresponding position in $arrAmount.

If the phone number is not found, the new phone number should be added to $arrTele and the new charge to $arrAmount. The ArraySize function specifies the position where to add the new number and charge at the end of the arrays.

```
$retIndex=FindStringExactInArray($arrTele, $noDialled);
if ($retIndex="-1")
{
    $retSize=ArraySize($arrTele);
    $arrTele[$retSize]=$noDialled;
    $arrAmount[$retSize]=$callCharge;
}
else
    $arrAmount[$retIndex]=$callCharge;
```

FirstBlockInst

**Syntax**
FirstBlockInst();

**Description**
Determines if the current block is the first of its kind in a frame.
Returns

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>True.</td>
</tr>
<tr>
<td>0</td>
<td>False.</td>
</tr>
</tbody>
</table>

Example

FirstBlockInst();
Where there are two block types, b1 and b2, in the input:
b10, b11, b12, b20, b13, b14
Result:
True (1) for b10, b20 and b13
b13 returns True (1) because it is separated from b10 by b20.

FirstBlockOnPage

Syntax

FirstBlockOnPage();

Description

Determines if the current block is the first of its kind in a frame on each page. This function is not useful during the pre-process phase, where it always returns 0.

Returns

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>True.</td>
</tr>
<tr>
<td>0</td>
<td>False.</td>
</tr>
</tbody>
</table>

Example

$fp=FirstBlockOnPage();
if(num($fp)=1) {
    $var=1
}

FlushOutput

Syntax

FlushOutput();

Description

Used during the processing of a job. Sends portions of a large output job to the printer, even though processing of the job has not yet been completed. The FlushOutput function can only be used in After Process scripts.
Returns

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OK. Action successful.</td>
</tr>
<tr>
<td>-1</td>
<td>Failed. Action NOT successful.</td>
</tr>
</tbody>
</table>

Example

```$pagecounter = num($pagecounter) + 1;
if(num($pagecounter) = 25
{    flushoutput();
}
```

ForcePoll

**Syntax**

```
ForcePoll(str_connectorName);
```

<table>
<thead>
<tr>
<th><strong>str_connectorName</strong></th>
<th>The name of the input connector to activate.</th>
</tr>
</thead>
</table>

**Description**

This script function signals an input connector to poll, and can be used with any input connector that supports scheduled polling (polling configured in the Scheduler Configuration dialog box).

**Returns**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OK. Action successful.</td>
</tr>
<tr>
<td>0</td>
<td>Failed. Action NOT successful.</td>
</tr>
</tbody>
</table>

**Example**

An external application has finished putting data into a database. StreamServer has a JDBC input connector called JDBCinput, and no polling is configured for this input connector. The external application signals StreamServer by creating a job on a TCP/IP connector. This job executes a script with the script function `ForcePoll("JDBCinput")`, and this script function signals the JDBC input connector to poll the database.

FrameOverflow

**Syntax**

```
FrameOverflow();
```
Description
In the output frame only. Generates a new page. If called from a Before Script, the current block is the first block on the next page; if called from an After Script, the current block is the last block on the current page.

Note: You cannot use the script functions NewPage or FrameOverflow when using the MultiPage function in PageOUT.

Returns
N/A

Example
if(num($count)=10)
{
    frameoverflow();
}

G

GetAttachmentContentEncoding

Syntax
GetAttachmentContentEncoding(num_file);

| num_file | The index of the attachment. Accepts the index of the attachment starting from 1 up to the number of attachments returned by GetConnectorValue("AttCount") or GetAttachmentCount(). |

Description
Returns the encoding of an attachment saved to disk via an EmailIN connector.

Returns

| str | The encoding of the attachment |

Example
$encoding = GetAttachmentContentEncoding(1);
GetAttachmentContentType

**Syntax**

\[
\text{GetAttachmentContentType}(\text{num\_file});
\]

<table>
<thead>
<tr>
<th><strong>num_file</strong></th>
<th>The index of the attachment. Accepts the index of the attachment starting from 1 up to the number of attachments returned by GetConnectorValue(&quot;AttCount&quot;) or GetAttachmentCount().</th>
</tr>
</thead>
</table>

**Description**

Returns the content-type of an attachment saved to disk via an EmailIN connector.

**Returns**

<table>
<thead>
<tr>
<th><strong>str</strong></th>
<th>The content-type of the attachment</th>
</tr>
</thead>
</table>

**Example**

\[
\$encoding = \text{GetAttachmentContentType}(1);
\]

GetAttachmentCount

**Syntax**

\[
\text{GetAttachmentCount}();
\]

**Description**

Returns the number of attachments in the current email that have been saved to disk via an EmailIN connector.

**Returns**

<table>
<thead>
<tr>
<th><strong>num</strong></th>
<th>The number of attachments in the current email that have been saved to disk by the EmailIN connector.</th>
</tr>
</thead>
</table>

**Example**

\[
\$count = \text{GetAttachmentCount}();
\]

GetAttachmentFile

**Syntax**

\[
\text{GetAttachmentFile}(\text{num\_file});
\]

<table>
<thead>
<tr>
<th><strong>num_file</strong></th>
<th>The index of the attachment to retrieve. To get the first attachment, set num_file to 1.</th>
</tr>
</thead>
</table>
Description

Returns the file name of an attachment saved to disk via an EmailIN connector. The file names are not the original attachment file names. Use the scripting function GetAttachmentOriginalFile to map the stored files to the original attachment names.

Returns

| str   | The file name of the attachment. |

Example

$file = GetAttachmentFile(1);

GetAttachmentOriginalFile

Syntax

GetAttachmentOriginalFile(num_file);

| num_file | The index of the attachment to retrieve. To get the first attachment, set num_file to 1. |

Description

Returns the original file name of an attachment saved to disk via an EmailIN connector. Maps the stored file to the original attachment name.

Returns

| str   | The original file name of the attachment. |

Example

$original = GetAttachmentOriginalFile(1);

GetConnectorValue

Syntax

GetConnectorValue(field_name);

| field_name | A string specifying the name of the field to be retrieved from the connector. |

Description

Returns the value of a named field from a connector.
Returns
A string specifying the value of a connector field. If the field is not found, an empty string is returned.

Example
$count=GetConnectorValue("subject");
Result:
$count="This is a test subject"

GetCurrModuleinfo

Syntax
GetCurrModuleinfo();

Description
Returns the current Event or Process status, depending on where the script is executed.

Note: You must enable Use Notifications in the Configure platform dialog. If this is not enabled, GetCurrModuleinfo will fail.

Returns

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Current status OK (running)</td>
</tr>
<tr>
<td>1</td>
<td>Current status Warning</td>
</tr>
<tr>
<td>2</td>
<td>Current status Failed</td>
</tr>
</tbody>
</table>

Example
$test = getcurrmoduleinfo();
if ($test = "2")
{ //Module has failed. Perform the following action. }

GetCurrSegDoc

Syntax
GetCurrSegDoc();

Description
Returns an index that identifies the current document, in the current segment, in the set of documents sent to the output connector. The function can be used at the following post-processing scripting levels:

Before and after document, process and page.
Script functions in alphabetical order

Script functions reference

Returns

<table>
<thead>
<tr>
<th>num</th>
<th>An index identifying the current document in the current output segment document array.</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Error.</td>
</tr>
</tbody>
</table>

Example

```plaintext
$doc_index = GetCurrSegDoc();
$invoice=GetSegMetadata($doc_index, "invoice_no");
```

GetCurrX

**Syntax**

GetCurrX();

**Description**

Returns the current position of a PageOUT object along the X axis in millimeters. You can change the X position of an object in PageOUT either by using script functions or by using variables in the Position dialog box.

See also GetCurrBlockY, GetCurrObjY and SetCurrX.

**Returns**

A number specifying the X coordinate in millimeters.

**Example**

```plaintext
$xpos = getcurrx();
```

GetCurrY

GetCurrY is an old function and is replaced with the functions GetCurrBlockY and GetCurrObjY.

GetCurrBlockY

**Syntax**

GetCurrBlockY();

**Description**

Returns the current position of a PageOUT block along the Y axis (i.e. the position of the upper left corner of the block) in millimeters. You can change the y position of an object in PageOUT either by using script functions or by using variables in the Position dialog box.

See also GetCurrObjY, GetCurrX and SetCurrY.
Returns
A number specifying the Y coordinate in millimeters.

Example
$ypos = getcurrblocky();

GetCurrObjY

Syntax
GetCurrObjY();

Description
Returns the current position of a PageOUT object along the Y axis (i.e. the position of the upper left corner of the object) in millimeters. You can change the y position of an object in PageOUT either by using script functions or by using variables in the Position dialog box.

See also GetCurrBlockY, GetCurrX and SetCurrY.

Returns
A number specifying the Y coordinate in millimeters.

Example
$ypos = getcurrobjy();

GetDate

Syntax
GetDate();

Description
Returns the system date as a string in the format yyyyMMdd, where:

- yyyy – four digit year, e.g. 2005.
- mm – month.
- dd – day.

Returns
A string specifying a date in the format yyyyMMdd.

Example
$a=getdate();
Result:
$a="20070508"
GetDateTime

Syntax
GetDateTime(picture_clause);

| picture_clause | A string specifying the format of the returned date and time |

Description
Uses the Picture Clause engine to return the system date and time as a string as specified by the picture clause.

An empty string returns GMT date and time in ISO format including milliseconds.

For Picture Clause syntax, see e.g. the StoryTeller documentation.

Returns
A string specifying a date and time in the format specified by the picture clause.

Example
$a=getdatetime("*");
Result:
$a="2009-04-03T11:03:30.456+01:00"
//
$b=getdatetime("MMM DD, YYYY");
Result:
$b="April 03, 2009";

GetDesignCenterVersion

Syntax
GetDesignCenterVersion();

Description
Returns the Design Center version that was used to create the Project.

Returns
The Design Center version.

Example
$version=GetDesignCenterVersion();
Result:
$version="StreamServe_Design_Center_5_0_0_build_729"
GetDistributionChannelsForRole

**Syntax**

GetDistributionChannelsForRole(role_name, doc_type, directory, channels_var);

<table>
<thead>
<tr>
<th>role_name</th>
<th>The name of the role for which you want to retrieve the distribution channels.</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc_type</td>
<td>The document type for which the role’s users use the distribution channels. The document type must be defined in the Project and the StreamServer application started.</td>
</tr>
<tr>
<td>directory</td>
<td>0 = internal directory, 1 = external directory</td>
</tr>
<tr>
<td>channels_var</td>
<td>The variable that will contain the returned values. If you specify $&lt;channels_variable&gt;[0]$, all channels are returned (if your counter starts on 0, see example below). If you specify $&lt;channels_variable&gt;$, only the first channel is returned.</td>
</tr>
</tbody>
</table>

**Description**

Returns the distribution channels for a role, for a specific document type.

**Returns**

An array with the distribution channels for a role’s users. If the argument channels_var is a string, only the first distribution channel is returned as a string.

**Example**

```php
$channels[0] = "";
$cnt = getDistributionChannelsForRole("administrator","invoice",0,$channels[0]);
$a = 0;
while ( num($a) < num($cnt) )
{
    Log( 0, "Channel : " + $channels[num($a)] );
    $a++;
}
```

GetDistributionChannelsForUser

**Syntax**

GetDistributionChannelsForUser(user_name, doc_type, directory, channels_var);

| user_name | The name of the user for which you want to retrieve the distribution channels. |
Script functions in alphabetical order

**Script functions reference**

### Description

Returns the distribution channels for a user, for a specific document type.

### Returns

An array with the distribution channels for a user. If the argument `channels_var` is a string, only the first distribution channel is returned.

### Example

```plaintext
$channels[0] = "";
$cnt = getDistributionChannelsForUser("dcool","order",0,$channels[0]);
$a = 0;
while ( num($a) < num($cnt) )
{
    Log( 0, "Channel : " + $channels[num($a)] );
    $a++;
}
```

### GetDocActualInserts

#### Syntax

```
GetDocActualInserts();
```

#### Description

Returns the inserts assigned to the document. The function can be used at the following post-processing scripting levels:

Before and after document, process and page.

**Note:** Enveloping must be used and the **Count inserts** option must be selected.
Returns

| strs  | A string listing the inserts used. |

Example

$inserts = GetDocActualInserts();
Result:
$inserts = "3;6;7"
$num_of_tokens = StrTok($inserts,";",$arr1);
Result
arr1[0] = 3
arr1[1] = 6
arr1[2] = 7

GetDocEnvelopeCount

Syntax

GetDocEnvelopeCount();

Description

Returns the number of envelopes for the current document. The function can be used at the following post-processing scripting levels:
Before and after document, process and page.

Note: Enveloping must be used.

Returns

| num  | A number specifying the number of envelopes. |

Example

$numEnvels = GetDocEnvelopeCount();
Result:
$numEnvels = 2

GetDocInsertCount

Syntax

GetDocInsertCount();

Description

Returns the number of inserts for the current document. The function can be used at the following post-processing scripting levels:
Before and after document, process and page.
Returns

| num | A number specifying the number of inserts |

Example

```plaintext
$inserts = GetDocInsertCount();
Result:
$inserts = 3
```

GetDocInsertEquivalents

Syntax

```plaintext
GetDocInsertEquivalents();
```

Description

Returns the insert equivalent count for the current document. The function can be used at the following post-processing scripting levels:

Before and after document, process and page.

Note: Enveloping must be used and the Count inserts option must be selected.

Returns

<table>
<thead>
<tr>
<th>num</th>
<th>A number specifying the inserts equivalent count.</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Failure. Enveloping is not used or the Count inserts options is not selected.</td>
</tr>
</tbody>
</table>

Example

```plaintext
$equivals = GetDocInsertEquivalents();
Result:
$equivals = 3.5
```

GetDocSheetCount

Syntax

```plaintext
GetDocSheetCount();
```

Description

Returns the number of sheets within the current document. You can only call this function after enveloping. Inserts are not included in the returned value. The function can be used at the following post-processing scripting levels:

Before and after document, process and page.
Returns
The number of sheets within the current document.

Example
$sheet_count = GetDocSheetCount();
Result:
$sheet_count = 2

GetDocumentDefinitionId

Syntax
GetDocumentDefinitionId();

Description
This script function can be used Before or After Process. It returns the identifier (GUID) of the current document definition.

Returns
Returns the identifier (GUID) of the current document definition.

Example
$currDocDefId = GetDocumentDefinitionId();

GetDocumentDefinitionName

Syntax
GetDocumentDefinitionName();

Description
This script function can be used Before or After Process. It returns the name of the current document definition.

Returns
Returns the name of the current document definition.

Example
$currDocDefName = GetDocumentDefinitionName();

GetEnvelNr

Syntax
GetEnvelNr();
Description
Returns the envelope number of the current document. The function can be used at the following post-processing scripting levels:
Before and after document, process and page.

Returns
The envelope number. If the envelope has no number assigned, 0 is returned.

Example
$env_nr = GetEnvelNr();
Result:
$env_nr = 12

GetExtJobId

Syntax
GetExtJobID();

Description
Returns the external ID of the current job. This function is usually called from an After Process script.
See also GetIntJobID, GetJobStatus, SetExtJobId.

Returns
A string containing the external ID of the current job.

Examples

Example 55 Using GetExtJobId

//Get the external ID of the current job
//and send the ID to the log

$jobid = getextjobid();
log (1, $jobid);

//Get the status of the current job. If an error has occurred,
//send an error message to the log.

$jobstatus = getjobstatus();
if ($jobstatus = "0")
    log (1, "An error has occurred");

//Get the internal ID of the current job
//and send it to the log.
$intjobid = getintjobid();
log(1, $intjobid);

Result:
The external ID of the current job is written to the log with log level 1. If the job was not completed successfully, an error message is written to the log. Finally, the internal ID of the current job is written to the log.

---

**Example 56 Using GetExtJobId**

$extjobid = " " + GetExtJobID();
Log (9, ">> ExtJobID =" + $extjobid + " : Time = " + gettime());

Result:
>> ExtJobID = SS_SAP-1001 : Time = 090807

---

### GetFirstPPDoc

**Syntax**

GetFirstPPDoc();

**Description**

Returns a counter index that identifies the first document, in a set of documents sent to the output connector in a Post-processor job.

**Returns**

<table>
<thead>
<tr>
<th>num</th>
<th>Counter index identifying the first document in the set of documents.</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Error.</td>
</tr>
</tbody>
</table>

**Example**

$doc_index = GetFirstPPDoc();
$total = 0;
while(num($doc_index) > 0)
{
    $invoice = GetPPMetadata($doc_index, "invoice_no");
    $doc_index = GetNextPPDoc($doc_index);
}
GetFirstSegDoc

Syntax
GetFirstSegDoc();

Description
Returns the index that identifies the first document in the current segment, in the set of documents sent to the output connector.

Returns

<table>
<thead>
<tr>
<th>num</th>
<th>An index identifying the first document in the current output segment document array.</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Error.</td>
</tr>
</tbody>
</table>

Example
$doc_index = GetFirstSegDoc();
$total = 0;
while(num($doc_index) > 0)
{
    $invoice = GetSegMetadata($doc_index, "invoice_no");
    $doc_index = GetNextSegDoc($doc_index);
}

GetGlobalSerNo

Syntax
GetGlobalSerNo(key);

<table>
<thead>
<tr>
<th>key</th>
<th>A string uniquely identifying a serial number key, representing for example a document type.</th>
</tr>
</thead>
</table>

Description
Returns the next serial number available for the specified key.

Returns

<table>
<thead>
<tr>
<th>num</th>
<th>A number specifying the next available serial number.</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Error.</td>
</tr>
</tbody>
</table>

Example
Assume the repository table has the following key entry:

| invoices   | 1234   | 1000   | 2000   |
Where:
First column (invoices) – the key
Second column (1234) – current value
Third column (1000) – minimum value
Fourth column (2000) – maximum value
By issuing the following function:

```php
$ser = getglobalserno("invoices");
$ser is set to 1234 and the table entry contains:
invoices   1235   1000   2000
```

**GetGlobalSerNoRange**

**Syntax**

GetGlobalSerNoRange(key, range);

<table>
<thead>
<tr>
<th>key</th>
<th>A string uniquely identifying a serial number key, representing for example a document type.</th>
</tr>
</thead>
<tbody>
<tr>
<td>range</td>
<td>The range of serial numbers you want to reserve, starting from the current value.</td>
</tr>
</tbody>
</table>

**Description**

Returns the first value in the specified range of serial numbers for the specified key. The specified range is reserved in the repository used by the StreamServer application. This can be useful if you want to avoid a database operation for each retrieval of serial number. Instead you reserve a set of serial numbers at once.

**Returns**

<table>
<thead>
<tr>
<th>num</th>
<th>The first value in the specified range.</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Error.</td>
</tr>
</tbody>
</table>

**Example**

Assume the repository table has the following key entry:

```plaintext
invoices   1234   1000   2000
```

Where:
First column (invoices) – the key
Second column (1234) – current value
Third column (1000) – minimum value
Fourth column (2000) – maximum value
By issuing the following function:
$ser = getglobalsernorange("invoices", 100);
$ser is set to 1234 and the table entry contains:

invoices   1334   1000   2000

GetHTTPHeaderValue

**Syntax**
GetHTTPHeaderValue(header_field);

| header_field | A string specifying the name of the field in the HTTP header. |

**Description**
Returns the value of a named field in the HTTP header of a job created by a request on an HTTP input connector.

**Returns**
A string specifying the value of the HTTP header field. If the field is not found, an empty string is returned.

**Example**
$type=GetHTTPHeaderValue("content");
Result:
$type="text"

GetIntJobId

**Syntax**
GetIntJobID();

**Description**
Returns the internal ID of the current job. This function is usually called from an **After Process** script. See also GetExtJobID, GetJobStatus, SetExtJobId.

**Returns**
A numeric value containing the internal ID of the current job.

Examples

**Example 57** Using GetIntJobId

```plaintext
//Get the external ID of the current job
//and send the ID to the log

$jobid = getextjobid();
log (1, $jobid);
```
//Get the status of the current job. If an error has occurred, 
//send an error message to the log.

$jobstatus = getjobstatus();
if ($jobstatus = "0")
    log (1, "An error has occurred");

//Get the internal ID of the current job 
//and send it to the log.

$intjobid = getintjobid();
log(1, $intjobid);

Result:
The external ID of the current job is written to the log with the log level of 1. If
the job was not completed successfully, an error message is written to the log.
Finally, the internal ID of the current job is written to the log.

Example 58 Using GetIntJobId

$intjobid = " " + Str (GetIntJobID());
Log (9, ">> IntJobID =" + $intjobid + " : Time = " + gettime());

Result:

>> IntJobID = 10 : Time = 090807

GetJobIDAttribute

GetJobIDAttribute is an old function and is removed.

GetJobIDDateAttribute

GetJobIDDateAttribute is an old function and is removed.

GetJobIDJob

GetJobIDJob is an old function and is removed.

GetJobIDNumAttribute

GetJobIDNumAttribute is an old function and is removed.
GetJobQueueItemId

GetJobQueueItemId is an old function and is removed.

GetJobQueueURI

GetJobQueueURI is an old function and is removed.

GetJobResourceIndex

Syntax

GetJobResourceIndex();

Description

Retrieves the index of a job sent to a Job Resource output connector. This function must be called from an After Process script, and cannot be used in Post-processor scripting.

You can only call the GetJobResourceIndex function once for each job resource. Calling the function a second time will cause a new index to be allocated to the job resource, which will result in confusion as to which index belongs to which job resource.

See also the Job Resource output connector settings, and the script functions OutputLXFJobResource and DeleteJobResource.

Returns

| num | The job resource index. A negative value indicates failure. |

Example

Enter the following in a PageOUT Process:

```
$pageoutresourceindex = GetJobResourceIndex();
```

The variable $pageoutresourceindex will contain the index of the job resource that is currently being created.

GetJobStatus

Syntax

GetJobStatus();

Description

Returns the status of the current job. This function is usually called from an After Process script. See also GetExtJobID, GetIntJobID, SetExtJobId.

Note: You must enable Use Notifications in the Configure platform dialog. Otherwise this function will always return 1.
Returns
A number that indicates whether the job has been successfully completed.

<table>
<thead>
<tr>
<th>num</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num&lt;0</td>
<td>The job has been completed successfully.</td>
</tr>
<tr>
<td>0</td>
<td>Failure.</td>
</tr>
</tbody>
</table>

Example
//Get the external ID of the current job
//and send the ID to the log

$jobid = getextjobid();
log (1, $jobid);

//Get the status of the current job. If an error has occurred,
//send an error message to the log.

$jobstatus = getjobstatus();
if ($jobstatus = "0")
   log (1, "An error has occurred");

//Get the internal ID of the current job
//and send it to the log.

$intjobid = getintjobid();
log(1, $intjobid);

Result:
The external ID of the current job is written to the log with the log level of 1. If
the job was not completed successfully, an error message is written to the log.
Finally, the internal ID of the current job is written to the log.

GetMetaDataDocument

Syntax
GetMetaDataDocument (metadata);

| metadata | The name of the metadata item for which to return the value. |
Description
This script function is used for retrieving the value of the specified metadata item of the current incoming document. The metadata item must be defined in the Document Type set on the current Message. This function is typically used in the Process to get the value of a metadata item. By using this script function instead of referring directly to a variable, it is possible to create scripts that are less dependent on a specific job and Message. Instead they can be reused in any Process that uses the same metadata group. By using a global metadata group it is possible to make very generic scripts.

Returns
Returns the value of the specified metadata item. If the metadata item has not been defined, an empty string is returned.

Example
$invoiceNumber = GetMetaDataDocument("invoice number");

GetMetaDataJob

Syntax
GetMetaDataJob(key);

<table>
<thead>
<tr>
<th>key</th>
<th>The name of the metadata attribute.</th>
</tr>
</thead>
</table>

Description
Returns the value of a metadata attribute associated with the input job. You use the SetMetaDataJob function to create the metadata attribute.

Returns
The value of the metadata attribute.

Example
$value=GetMetaDataJob("ref");

GetMetaDataMessage

Syntax
GetMetaDataMessage(key);

<table>
<thead>
<tr>
<th>key</th>
<th>A string, uniquely identifying the GUID or the name of the metadata (case sensitive).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Note</td>
<td>If name is used, make sure the name is not updated in the Design Center configuration.</td>
</tr>
</tbody>
</table>
Description
Returns the current value of a metadata. The metadata must have the Message context.

⚠️ Using the \texttt{GetMetaDataMessage} scripting function for a metadata without the Message context is not supported.

To save space in the repositories and to improve search performance, you should not use the Message context unless the metadata really has to be available to external applications.

This function is typically used in an exception rule to get the value of a metadata. The metadata is assigned its value before the exception rule is executed. By using the scripting function instead of referring directly to a variable, any changes made to the metadata by external applications are considered.

The scripting function can be used in:

• Scripts related to a Process
• Scripts inside the Process.
• Before and after scripts.

For example, if you want to use the same metadata in a Process as in an exception rule, the scripting function can be used in the Process as well.

Returns
A string with the value of the metadata. If the metadata is not found or the metadata value is yet not set, an empty string is returned.

Example
\texttt{$\$Category = GetMetaDataMessage("a86199fd-5f5a-4e53-a3af-47c72ae6321f");}

GetMailMachine

Syntax
GetMailMachine();

Description
Returns the name of the enveloping machine to which the current document was assigned. The function can be used at the following post-processing scripting levels:

Before and after document, process and page.

Returns
A string specifying the enveloping machine. If no string is assigned to the enveloping machine, an empty string is returned.
Example
$mail_mach=GetMailMachine();

GetNextPPDoc

Syntax
GetNextPPDoc(doc_index);

<table>
<thead>
<tr>
<th>doc_index</th>
<th>index that identifies the previous document in the set of Documents sent to the output connector.</th>
</tr>
</thead>
</table>

Description
Returns a counter index that identifies the next document in a set of documents in the post-processor job sent to the output connector.

Returns

<table>
<thead>
<tr>
<th>num</th>
<th>A counter index identifying the next document.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>There are no more Documents.</td>
</tr>
<tr>
<td>-1</td>
<td>Error</td>
</tr>
</tbody>
</table>

Example
$doc_index = GetFirstPPDoc();
while(Num($doc_index) > 0 )
{
    $invoice=GetPPMetadata($doc_index, "invoice_no");
    $doc_index=GetNextPPDoc($doc_index);
}

GetNextSegDoc

Syntax
GetNextSegDoc(doc_index);

<table>
<thead>
<tr>
<th>doc_index</th>
<th>Index of the previous document in the current segment in the set of documents sent to the output connector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>num</td>
<td>An counter index identifying the next document.</td>
</tr>
</tbody>
</table>

Description
Returns an index that identifies the next document, in the current segment, in the set of documents sent to the output connector.
Returns

<table>
<thead>
<tr>
<th>num</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>There are no more documents.</td>
</tr>
<tr>
<td>-1</td>
<td>Error.</td>
</tr>
</tbody>
</table>

Example

```php
$doc_index = GetFirstSegDoc();
while(num($doc_index) > 0 )
{
    $invoice=GetSegMetadata($doc_index, "invoice_no");
    $doc_index=GetNextSegDoc($doc_index);
}
```

GetPhysicalPlatform

**Syntax**

GetPhysicalPlatform();

**Description**

Returns the name of the physical platform layer deployed in Control Center.

**Returns**

The name of the physical platform layer.

**Example**

```php
$project=GetPhysicalPlatform();
```

GetPPDocId

**Syntax**

GetPPDocId();

**Description**

Returns the document ID of a document stored in a Post-processor repository. You can use this function when storing documents in a Post-processor repository as well as when retrieving documents from a Post-processor repository.

If the function is used when storing documents, the value (document ID) is set each time a new document is created. Note that if a document spans over several Messages, the value is set when the first Message in the document is called, and that this value is kept through all Messages in the document.
If the function is used when retrieving documents, the value (document ID) is set each time a new document is retrieved. In this case, the function can be used at the following post-processing scripting levels:

- Before document, process and page.
- After document, process and page.

**Returns**

<table>
<thead>
<tr>
<th>num</th>
<th>The document ID.</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Error.</td>
</tr>
</tbody>
</table>

**Example**

```javascript
$doc_id = GetPPDocId();
```

**GetPPDocProperty**

**Syntax**

```javascript
GetPPDocProperty(docHandle, propertyName);
```

<table>
<thead>
<tr>
<th>docHandle</th>
<th>An identifier specifying a document. You can retrieve the identifier by using either the GetFirstPPDoc() or the GetNextPPDoc() function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>propertyName</td>
<td>The name of the document property. The <code>propertyName</code> can be:</td>
</tr>
<tr>
<td></td>
<td>• ppdocid – the Post-processor document ID.</td>
</tr>
<tr>
<td></td>
<td>• ppjobid – the Post-processor job ID.</td>
</tr>
<tr>
<td></td>
<td>• status – the document status.</td>
</tr>
<tr>
<td></td>
<td>• priority – the document priority.</td>
</tr>
<tr>
<td></td>
<td>• creationtime – the document creation time.</td>
</tr>
<tr>
<td></td>
<td>• processtime – the document process time.</td>
</tr>
<tr>
<td></td>
<td>• error – the document error.</td>
</tr>
<tr>
<td></td>
<td>• pages – the number of logical pages in the document.</td>
</tr>
<tr>
<td></td>
<td>• resrepository – the resource repository.</td>
</tr>
<tr>
<td></td>
<td>• resserver – the resource repository server.</td>
</tr>
</tbody>
</table>

**Description**

Returns the value of the specified property for the document specified by the `docHandle` parameter.
Returns
The value of propertyName.

Example
$doc_handle=GetFirstPPDoc();
$subject=GetPPDocProperty($doc_handle, "ppdocid");
Result:
$subject=123

GetPPError

Syntax
GetPPError();

Description
Returns the last error that occurred during the Post-processor job.

Returns

<table>
<thead>
<tr>
<th>num</th>
<th>Error code of the last error that occurred during post-processor job.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error exists.</td>
</tr>
<tr>
<td>-1</td>
<td>The function did not succeed.</td>
</tr>
</tbody>
</table>

Example
$error = GetPPError();
if(IsLastSegment())
{
    if(num($error) > 0)
    {
        $doc_id = GetFirstPPDoc();
        while(num($doc_id) > 0)
        {
            $ret = UpdatePPDocStatus(num($doc_id), "error", num($error));
            $doc_id = GetNextPPDoc($doc_id);
        }
    }
}

GetPPJobId

Syntax
GetPPJobId();
Script functions in alphabetical order

Script functions reference

Description
Returns the Post-processor repository job ID for the current job. You can call this function as soon as the SDR driver has started to store the job in the repository. The function can be used at the following post-processing scripting levels:
Before and after job, document, process and page.
You can also use this function on Job End level (not post-processing job level) to retrieve the Post-processor repository job ID.

Returns

<table>
<thead>
<tr>
<th>num</th>
<th>The job ID.</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Error</td>
</tr>
</tbody>
</table>

Example

$job_id = GetPPJobId();

GetPPJobProperty

Syntax

GetPPJobProperty(docHandle, propertyName);

docHandle | An identifier specifying a document. You can retrieve the identifier by using either the GetFirstPPDoc() or the GetNextPPDoc() function.

propertyName | The name of the job property. The propertyName can be:
• ppjobid – the Post-processor job ID.
• status – the job status.
• priority – the job priority.
• creationtime – the job creation time.
• processtime – the job process time.
• error – the job error.
• pages – the number of logical pages in the job.
• name – the name of the job.
• strsid – the StreamServe job ID.
• user – the user who submitted the job.
• description – the description of the job.

Description
Returns the value of the specified property for the job containing the document specified by the docHandle parameter.
Returns

The value of propertyName.

Example

$doc_handle=GetFirstPPDoc();
$jobid=GetPPJobProperty($doc_handle, "ppjobid")
Result:
$jobid = 1

GetPPMetadata

Syntax

GetPPMetadata(docHandle, metadataName);

<table>
<thead>
<tr>
<th>docHandle</th>
<th>An identifier specifying a document. You can retrieve the identifier by using either the GetFirstPPDoc() or the GetNextPPDoc() function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>metadataName</td>
<td>The metadata name connected to the current document</td>
</tr>
</tbody>
</table>

Description

Returns the value of the specified metadata for the document specified by the docHandle parameter.

Returns

| str               | The value of the specified metadata. If the specified metadata is not found, an empty string is returned. |

Example

$doc_handle=GetFirstPPDoc();
$subject=GetPPMetadata($doc_handle, "invoice_no")
Result:
$subject = "1234"

GetPPReposProperty

Syntax

GetPPReposProperty(docHandle, propertyName);

| docHandle         | An identifier specifying a document. You can retrieve the identifier by using either the GetFirstPPDoc() or the GetNextPPDoc() function. |


<table>
<thead>
<tr>
<th>propertyName</th>
<th>The name of the repository property. The propertyName can be:</th>
</tr>
</thead>
<tbody>
<tr>
<td>server</td>
<td>the FO server name.</td>
</tr>
<tr>
<td>dbname</td>
<td>the FO database name.</td>
</tr>
<tr>
<td>physicalserver</td>
<td>the physical server for the FO server.</td>
</tr>
<tr>
<td>dblogicalname</td>
<td>the database logical name.</td>
</tr>
</tbody>
</table>

**Description**

Returns the value of the specified repository property storing the document specified by the docHandle parameter.

**Returns**

The value of propertyName.

**Example**

```javascript
$doc_handle=GetFirstPPDoc();
$subject=GetPPReposProperty($doc_handle, "server");
```

**GetPreviewType**

**Syntax**

```javascript
GetPreviewType();
```

**Description**

Returns a number, indicating whether the current job is a preview job or not. If the job is a preview job, the number indicates from where the preview job was invoked.

**Returns**

<table>
<thead>
<tr>
<th></th>
<th>The job is not a preview job.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The job is a preview job, invoked from Ad Hoc Correspondence, Correspondence Reviewer, or Composition Center.</td>
</tr>
<tr>
<td>1</td>
<td>The job is a preview job for which one of the following applies:</td>
</tr>
<tr>
<td></td>
<td>• The job is invoked from a Preview input connector.</td>
</tr>
<tr>
<td></td>
<td>• The job is invoked from a connector, containing an strs-tm-preview metadata (as HTTP header value).</td>
</tr>
<tr>
<td>2</td>
<td>The job is a preview job, invoked from a Design Center Project (for example, from a PageOUT Process).</td>
</tr>
</tbody>
</table>

**Example**

See Example 45 on page 82.
GetProjectName

**Syntax**

GetProjectName();

**Description**

Returns the name of the Design Center Project.

**Returns**

The name of the Project that runs on the StreamServer used for processing the job.

**Example**

$project=GetProjectName();

GetProjectRevision

**Syntax**

GetProjectRevision();

**Description**

Returns the revision of the Design Center Project. The revision is the version control system label defined when issuing the Create Release command in Design Center. The syntax of the returned string depends on the version control system used.

This function is only applicable if you use a version control system.

**Returns**

The Project revision. Note that the syntax depends on the version control system used.

**Example**

$revision=GetProjectRevision();

GetRequestedPreviewContentType

**Syntax**

GetRequestedPreviewContentType();

**Description**

*This function is only available for preview calls from web service clients.*

Returns a string containing an optionally requested content type for a response to a preview request. The returned value can be used to select an appropriate connector for the preview Process.
Returns
A string containing the content type requested by a web service client. If no content type is requested, an empty string is returned.

Example
See Example 45 on page 82.

GetRolesForUser

Syntax
GetRolesForUser(user_name, directory, roles_var);

| user_name | The name of the user for which you want to retrieve the roles |
| directory | 0 = internal directory, 1 = external directory |
| roles_var | The variable that will contain the returned values. If you specify $<roles_variable>[0], all roles are returned (if your counter starts on 0, see example below). If you specify $<roles_variable>, only the first role is returned. |

Description
Returns the roles associated with a user.

Returns
An array with the roles associated with a user. If the argument roles_var is a string, only the first role is returned.

Example

```perl
$roles[0] = "";
$cnt = getRolesForUser("dcool",0,$roles[0]);
$a = 0;
while ( num($a) < num($cnt) )
{
    Log( 0, "Roll : " + $roles[num($a)] );
    $a++;
}
```
GetSegDocProperty

Syntax

GetSegDocProperty(docHandle, propertyName);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>docHandle</td>
<td>An identifier specifying a document. You can retrieve this identifier by using either the GetFirstSegDoc(), GetCurrSegDoc(), or GetNextSegDoc() function.</td>
</tr>
<tr>
<td>propertyName</td>
<td>The name of the document property. The propertyName can be:</td>
</tr>
<tr>
<td></td>
<td>• ppdocid – the Post-processor document ID.</td>
</tr>
<tr>
<td></td>
<td>• ppjobid – the Post-processor job ID.</td>
</tr>
<tr>
<td></td>
<td>• status – the document status.</td>
</tr>
<tr>
<td></td>
<td>• priority – the job priority.</td>
</tr>
<tr>
<td></td>
<td>• creationtime – returns the time when the job was stored.</td>
</tr>
<tr>
<td></td>
<td>• processetime – the time when the job was processed.</td>
</tr>
<tr>
<td></td>
<td>• error – the job error code.</td>
</tr>
<tr>
<td></td>
<td>• pages – the number of pages in the job.</td>
</tr>
<tr>
<td></td>
<td>• resrepository – the resource repository name.</td>
</tr>
<tr>
<td></td>
<td>• strsid – the resource server name.</td>
</tr>
</tbody>
</table>

Description

Returns the document property specified by propertyName for the document specified by the docHandle parameter. The function can only be used if you have defined an enveloping machine. See the Document sorting and bundling documentation.

Returns

The value of propertyName.

Example

$doc_handle = GetFirstSegDoc();
$doc_id = GetSegDocProperty($doc_handle, "ppdocid");
Result:
$doc_id= 123
GetSegJobProperty

Syntax

GetSegJobProperty(docHandle, propertyName);

<table>
<thead>
<tr>
<th>docHandle</th>
<th>An identifier specifying a document. You can retrieve this identifier by using either the GetFirstSegDoc(), GetCurrSegDoc(), or GetNextSegDoc() function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>propertyName</td>
<td>The name of the job property. The propertyName can be:</td>
</tr>
<tr>
<td></td>
<td>• ppjobid – the Post-processor job ID.</td>
</tr>
<tr>
<td></td>
<td>• status – the job status.</td>
</tr>
<tr>
<td></td>
<td>• priority – the job priority.</td>
</tr>
<tr>
<td></td>
<td>• creationtime – the time when the job was stored.</td>
</tr>
<tr>
<td></td>
<td>• processtime – the time when the job was processed.</td>
</tr>
<tr>
<td></td>
<td>• error – the job error code.</td>
</tr>
<tr>
<td></td>
<td>• pages – the number of pages in job.</td>
</tr>
<tr>
<td></td>
<td>• name – the job name.</td>
</tr>
<tr>
<td></td>
<td>• strsid – the StreamServe job ID.</td>
</tr>
<tr>
<td></td>
<td>• user – the user name.</td>
</tr>
<tr>
<td></td>
<td>• description – the description of the job.</td>
</tr>
</tbody>
</table>

Description

Returns the job property specified in the propertyName parameter for the document specified in the docHandle parameter. The function can only be used if you have defined an enveloping machine. See the Document sorting and bundling documentation.

Returns

The value of propertyName.

Example

$doc_handle = GetSegFirstDoc();
$job_id = GetSegJobProperty($doc_handle, "ppjobid");
Result:
$job_id = 1

GetSegment

Syntax

GetSegment();
Description
Returns the current segment number. This is the same number that replaces the ${SEGMENT} variable in the output file name. The function can only be used if you have defined an enveloping machine. See the Document sorting and bundling documentation.

Returns
A number specifying the current segment.

Example
$segment = getSegment();

GetSegMetadata

Syntax
GetSegMetadata(docHandle, metadataName);

docHandle An identifier specifying a document.
metadataName The metadata name connected to the document specified in the docHandle parameter.

Description
Returns the value of the metadata specified in the metadataName parameter, for the document in the current segment specified in the docHandle parameter.

Returns
A string specifying the value of the specified metadata. If the metadata is not found, an empty string is returned.

Example
$doc_handle = GetSegFirstDoc();
$subject = GetSegMetadata($doc_handle, "invoice_no");
Result:
$subject = "1234"

GetSegReposProperty

Syntax
GetSegReposProperty(docHandle, propertyName);

docHandle An identifier specifying a document. You can retrieve this identifier by using either the GetFirstSegDoc(), GetCurrSegDoc(), or GetNextSegDoc() function.
GetSegReposProperty

Syntax
GetSegReposProperty($docHandle, $propertyName);

Description
Returns the repository property specified in the $propertyName parameter for the document specified in the $docHandle parameter. The function can only be used if you have defined an enveloping machine. See the Document sorting and bundling documentation.

Returns
The value of $propertyName.

Example
$doc_handle = GetFirstDoc();
$ret = GetSegReposProperty($doc_handle, "server");

GetSender

Syntax
GetSender();

Description
Returns the name of the sender of the job. You specify the sender in the Runtime configuration in Design Center.

Note: The sender is set when the collect phase is complete and before the pre-process phase has started. Therefore, you should not use the GetSender function in Before Job or Retrieved scripts. If used in these types of scripts, the function will return user anonymous.

Returns
The name of the sender.

Example
$sender = GetSender();
GetSerNo

Syntax
GetSerNo(filename, str_key);

<table>
<thead>
<tr>
<th>filename</th>
<th>A string specifying the path and filename of the file containing the number sequence.</th>
</tr>
</thead>
<tbody>
<tr>
<td>str_key</td>
<td>A string specifying the key.</td>
</tr>
</tbody>
</table>

Description
Calculates the next counter value in a number sequence according to a specified key. Returns the next counter value and amends the value to the number sequence.

Returns
A number specifying the next value in a number sequence.

Example
$a="C:\stream\serve\sernofile";  
$b="test";  
$c=getserno($a,$b);

Contents of the file sernofile:

<table>
<thead>
<tr>
<th>key</th>
<th>value1</th>
<th>value2</th>
<th>value3</th>
<th>value4</th>
</tr>
</thead>
<tbody>
<tr>
<td>test0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>test</td>
<td>10</td>
<td>50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>test1</td>
<td>57</td>
<td>50</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

Result: $c=11

Each line in the file can contain 2, 3, or 4 columns.

<table>
<thead>
<tr>
<th>columns</th>
<th>description</th>
</tr>
</thead>
</table>
| 2 columns: | column1 = key  
|           | column2 = current counter value (incremented after call) |
| 3 columns: | column1 = key  
|           | column2 = current counter value (incremented after call)  
|           | column3 = maximum counter value (When this number is reached, the counter will be set to zero) |
| 4 columns: | column1 = key  
|           | column2 = current counter value (incremented. after call)  
|           | column3 = minimum counter value  
|           | column4 = maximum counter value (when this number is reached, the counter will be set to the minimum counter value) |
GetSubst

Syntax
GetSubst(tbl_file, str_key, num_col);

<table>
<thead>
<tr>
<th>tbl_file</th>
<th>A string specifying the path and filename of the substitution table.</th>
</tr>
</thead>
<tbody>
<tr>
<td>str_key</td>
<td>A string specifying the key of the substitution table entry to search for.</td>
</tr>
<tr>
<td>num_col</td>
<td>A number specifying the value column. The first value column number is 0.</td>
</tr>
</tbody>
</table>

Description
Returns the value from the specified value column in the specified substitution table entry.

Returns
The value located in num_col of the entry specified by str_key.

Example
In this example, the following substitution table (author.tbl) is used.

```plaintext
//!CodePage UTF8!
//!multicolumn!
agatha christie  1890
alfred hitchcock 1899
```

The following script returns the second value from the entry with key agatha:

```plaintext
$born = GetSubst("../data/tables/author.tbl","agatha",1);
```

When the script is run, the variable $born gets the value 1890.

Notes
• A substitution table is kept in memory until the script function EraseSubst is called for that table.
• For backwards compatibility, Subst and SubstArr do not remember comments made in the substitution file. If you want WriteSubst to keep your comments, call GetSubst before any other substitution function accesses the substitution file.
• When retrieving a specific value, the GetSubst function does not take the column separators (tab or space characters) into account. To make sure that the correct value is returned, you should use double quote (""") to indicate empty columns in the table. Otherwise, the next non-empty column will be returned.
GetSubstColCount

Syntax

GetSubstColCount(tbl_file, str_key);

tbl_file | A string specifying the path and filename of the substitution table.
str_key  | The key of the entry to search for.

Description

Returns the number of columns in the entry specified by str_key in the substitution table tbl_file. The column count includes any empty columns.

Returns

<table>
<thead>
<tr>
<th>num</th>
<th>The number of value columns in the entry specified by str_key.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The key does not exist.</td>
</tr>
</tbody>
</table>

Examples

Example 59 GetSubstColCount – example 1.

In this example, the following substitution table (author.tbl) is used.

```plaintext
//!CodePage UTF8!
//!multicolumn!
agatha christie  1890
alfred hitchcock 1899
```

The following script returns the number of value columns for the entry with the key agatha:

```
$numCol = GetSubstColCount("../data/tables/author.tbl", "agatha");
```

When the script is run, the variable $numCol gets the value 2.

Example 60 GetSubstColCount – example 2.

The StreamServer does not keep empty columns. If you need a fixed number of columns, you must use a value indicating an empty value. You can, for example, use (void) as indicator.
In this example, the following substitution table (author.tbl) is used.

```plaintext
//!CodePage UTF8!
//!multicolumn!
agatha christie (void) 1890
alfred hitchcock (void) 1899
```

The following script returns the number of value columns for the entry with the key agatha:

```plaintext
$numCol = GetSubstColCount("../data/tables/author.tbl", "agatha");
```

When the script is run, the variable `numCol` gets the value 3.

**Notes**

- A substitution table is kept in memory until the script function `EraseSubst` is called for that table.
- For backwards compatibility, `Subst` and `SubstArr` do not remember comments made in the substitution file. If you want `WriteSubst` to keep your comments, call `GetSubst` before any other substitution function accesses the substitution file.

### GetSubstKey

**Syntax**

```plaintext
GetSubstKey(tbl_file, num_key_index);
```

<table>
<thead>
<tr>
<th><strong>tbl_file</strong></th>
<th>A string specifying the path and filename of the substitution table.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>num_key_index</strong></td>
<td>A number specifying the index of the entry key to return (i.e. a row in the substitution table). The first key index is 0.</td>
</tr>
</tbody>
</table>

**Description**

Returns a key from a substitution table.

**Returns**

The entry key with index (i.e. row) `num_key_index`. If the entry was not found, an empty string is returned.
Example

In this example, the following substitution table (author.tbl) is used.

```
/*!CodePage UTF8!
/*!multicolumn!
agatha christie  1890
alfred hitchcock 1899
```

The following script returns the entry keys for the two first entries in the substitution table:

```
$name1 = GetSubstKey("../data/tables/author.tbl",0);
$name2 = GetSubstKey("../data/tables/author.tbl",1);
```

When the script is run, $name1 and $name2 gets the following values:

```
$name1 = "agatha"
$name2 = "alfred"
```

Notes

- A substitution table is kept in memory until the script function EraseSubst is called for that table.
- For backwards compatibility, Subst and SubstArr do not remember comments made in the substitution file. If you want WriteSubst to keep your comments, call GetSubst before any other substitution function accesses the substitution file.

GetSubstKeyCount

**Syntax**

```
GetSubstKeyCount(tbl_file);
```

**Description**

Returns the number of entries in a substitution table.

**Returns**

The number of entries in the substitution table specified by tbl_file.

**Example**

In this example, the following substitution table (author.tbl) is used.

```
/*!CodePage UTF8!
/*!multicolumn!
agatha christie  1890
alfred hitchcock 1899
```
The following script returns the number of entries in `author.tbl`:

```plaintext
$numRow = GetSubstKeyCount("../data/tables/author.tbl");
```

When the script is run, the variable `$numRow` gets the value 2.

**Notes**

- A substitution table is kept in memory until the script function `EraseSubst` is called for that table.
- For backwards compatibility, `Subst` and `SubstArr` do not remember comments made in the substitution file. If you want `WriteSubst` to keep your comments, call `GetSubst` before any other substitution function accesses the substitution file.

### GetTextFormattingInfo

**Syntax**

```plaintext
GetTextFormattingInfo(base, lxf_data, width, height);
```

<table>
<thead>
<tr>
<th><strong>base</strong></th>
<th>A string specifying the prefix of the returned variable names.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>lxf_data</strong></td>
<td>The LXF formatted data to be split into separate areas.</td>
</tr>
<tr>
<td><strong>width</strong></td>
<td>A number specifying the width of the first text area.</td>
</tr>
<tr>
<td><strong>height</strong></td>
<td>A number specifying the height of the first text area.</td>
</tr>
</tbody>
</table>

**Description**

Enables splitting of LXF text into separate areas, for example to create multi-column output. The LXF text must be comply with the Advanced Text functionality in PageOUT.

The function returns variables with values specifying:

- The text that fits into the specified area.
- The actual dimensions and the number of text rows in this text area.
- The text that does not fit into the specified area.
- The dimensions and the number of text rows in this remaining text area.
Returns

<table>
<thead>
<tr>
<th>$base_text</th>
<th>A string in LXF format that represents text that fits into the specified area (height/width arguments)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$base_width</td>
<td>The width of the $base_text string.</td>
</tr>
<tr>
<td>$base_height</td>
<td>The height of the $base_text string.</td>
</tr>
<tr>
<td>$base_rows</td>
<td>The number of text rows that fit into the specified area.</td>
</tr>
<tr>
<td>$base_next</td>
<td>A string in LXF format that does not fit into the specified area and can be used in the next area.</td>
</tr>
<tr>
<td>$base_next_width</td>
<td>The width of the text that does not fit into the specified area.</td>
</tr>
<tr>
<td>$base_next_height</td>
<td>The height of the text that does not fit into the specified area.</td>
</tr>
<tr>
<td>$base_next_rows</td>
<td>The number of text rows that does not fit into the specified area.</td>
</tr>
<tr>
<td>0</td>
<td>Error</td>
</tr>
</tbody>
</table>

Example

```plaintext
$file = "data/overlays/fragment.lxf";
// load the LXF fragment into variable
$err = FileOpen( $file, "r" );
$part = 0;
if(num($err) = 0)
{
    $lxf = "";
    $line = "";
    while( 0 = FileReadLn( $file, $line ) )
    {
        $lxf = $lxf + $line;
        $lxf = $lxf + "<0D><0A>";
    }
    FileClose( $file );

    $test_width = 50;
    $test_height = 60;

    // write test output, original
    $filename = "data/overlays/fragment.part." + $part + ".lxf";
    fileopen($filename, "wb" );
```

"base_text" A string in LXF format that represents text that fits into the specified area (height/width arguments)

"base_width" The width of the $base_text string.

"base_height" The height of the $base_text string.

"base_rows" The number of text rows that fit into the specified area.

"base_next" A string in LXF format that does not fit into the specified area and can be used in the next area.

"base_next_width" The width of the text that does not fit into the specified area.

"base_next_height" The height of the text that does not fit into the specified area.

"base_next_rows" The number of text rows that does not fit into the specified area.

0 Error
filewrite($filename,$lxf);
fileclose($filename);
$part++;

GetTextFormattingInfo( "test", $lxf, $test_width, $test_height );

// write test output, part 1
$testfile = "data/overlays/fragment.part." + $part + ".lxf";
fileopen($testfile, "wb" );
filewrite($testfile,$test_text);
fileclose($testfile);
$part++;
//write test output, part 2
$testfile = "data/overlays/fragment.part." + $part + ".lxf";
fileopen($testfile, "wb" );
filewrite($testfile,$test_next);
fileclose($testfile);
$part++;
log(0, "variable: test_width: " + $test_width );
log(0, "variable: test_height: " + $test_height );
log(0, "variable: test_rows: " + $test_rows );
log(0, " ");
log(0, "variable: test_next_width: " + $test_next_width );
log(0, "variable: test_next_height: " + $test_next_height );
log(0, "variable: test_next_rows: " + $test_next_rows );

fileclose( $file );
}
else
{ $errtext=ioerrtext($err);
  log(0,"Could not open " + $file + ": " + $errtext);
}

GetTime

**Syntax**
GetTime();

**Description**
Returns the system time in the format **hhmmss** (24-hour time format).

**Returns**
A string in the format **hhmmss**.
Example

$a = \text{gettime}();
Returns:
$a = "120559"

GetTopJobId

**Syntax**

GetTopJobID();

**Description**

Returns the internal ID of the input job (top job). See also `GetIntJobID`.

**Returns**

A numeric value containing the internal ID of the input job.

GetXoffs

**Syntax**

GetXoffs();

**Description**

Returns the current X offset value in millimeters. See also `SetXoffs`.

**Returns**

A number specifying a measurement in millimeters.

Example

$xoffs = \text{getxoffs}();

GetYoffs

**Syntax**

GetYoffs();

**Description**

Returns the current Y offset value in millimeters. See also `SetYoffs`.

**Returns**

A number specifying a measurement in millimeters.

Example

$yoffs = \text{getyoffs}();
HexStr

Syntax

HexStr(str);

<table>
<thead>
<tr>
<th>str</th>
<th>A string.</th>
</tr>
</thead>
</table>

Description

Copies a string and interprets numbers in angle brackets as two-digit hexadecimal numbers (useful for Escape sequences). Other characters are returned unchanged.

Returns

A string.

Example

```
$a=hexstr("pre<1B,7C,8A>post");
$b=hexstr("xyz <30,39>");  
```

Result:

```
$a="pre...post"
$b="xyz 09"
```

In2Mm

Syntax

In2Mm(num)

<table>
<thead>
<tr>
<th>num</th>
<th>A number in inches to convert to millimeters.</th>
</tr>
</thead>
</table>

Description

Converts a number given in inches to millimeters.

Returns

A number in millimeters as a floating point value.

Example

```
$in = 5;
$mm = in2mm($in);
```
In2Pt

Syntax

In2Pt(num)

| num | A number in inches to convert to points unit. |

Description

Converts a number given in inches to points unit.

Returns

A number in points unit as a floating point value.

Example

$in = 5;
$pt = in2pt($in);

Int

Syntax

Int(num)

| num | A number. |

Description

Returns truncation-to-integer (as a floating point value) of the argument.

Returns

A number.

Example

int(324.66)
Result: 324

InConnectorName

Syntax

InConnectorName();
Description
Returns the name of the current input connector. An Event can ask where the input arrived from. This function can be used in Rules for both Events and Processes. The input from the business application arriving at a specific connector can trigger the processing of certain documents.

Returns
A string.

Example
$inconnector=inconnectorname();

IncProcStatCounter

Syntax
IncProcStatCounter(counter_name);

<table>
<thead>
<tr>
<th>counter_name</th>
<th>A string or a variable.</th>
</tr>
</thead>
</table>

Description
Increases the value of counter_name with one. If counter_name does not exist, it is created and set to one. The value of counter_name is included in the reports created by the Communication Reporter. The function can be used at any level in a Project.

Note: The value of counter_name increases by default also in the pre-processing phase. To avoid this you should check that the function is not executed in the pre-processing phase.

Returns

<table>
<thead>
<tr>
<th></th>
<th>OK</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example
if (PreProc()==0 {
   $result = incProcStatCounter("department1");
}  

InQueueName

InQueueName is replaced by InConnectorName.
InsertOverlay

**Syntax**

```
InsertOverlay(overlay_name, x_offset, y_offset);
```

<table>
<thead>
<tr>
<th>overlay_name</th>
<th>The name of the overlay to use on the page.</th>
</tr>
</thead>
<tbody>
<tr>
<td>x_offset</td>
<td>The horizontal distance of the overlay, calculated in points from the upper left corner of the page.</td>
</tr>
<tr>
<td>y_offset</td>
<td>The vertical position of the overlay, calculated in points from the upper left corner of the page.</td>
</tr>
</tbody>
</table>

**Description**

Adds an overlay to the current page.

If `${<variable>}` or `$<variable>` is used in the overlay specified in the **overlay_name** parameter, the variable is replaced by its value when the `InsertOverlay()` function is called.

The function can be used at the following post-processing scripting levels:

Before and after page.

**Returns**

<table>
<thead>
<tr>
<th>num</th>
<th>The page overlay was successfully added.</th>
</tr>
</thead>
<tbody>
<tr>
<td>num&gt;0</td>
<td>Error.</td>
</tr>
</tbody>
</table>

**Example**

```
InsertOverlay("SummaryOverlay", 0, 5);
```

InsertPage

**Syntax**

```
InsertPage(overlay_name, strProcesslink);
```

<table>
<thead>
<tr>
<th>overlay_name</th>
<th>The name of the overlay to use on the inserted page.</th>
</tr>
</thead>
<tbody>
<tr>
<td>strProcesslink</td>
<td>The name of the process link. If not specified, the current process link is used if it is available, otherwise the default process link is used.</td>
</tr>
</tbody>
</table>

**Description**

Inserts a new page in the output stream.

If `${<variable>}` or `$<variable>` is used on the **overlay_name** overlay it is replaced by the `$<variable>` value when the `InsertPage` function is called.
The function can be used at the following post-processing scripting levels:
Before and after document, process and page.

**Returns**

<table>
<thead>
<tr>
<th>num</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The page was successfully inserted.</td>
</tr>
<tr>
<td>num&gt;0</td>
<td>Error</td>
</tr>
</tbody>
</table>

**Example**

```
InsertPage("SummaryOverlay","SummaryProcessLink");
```

**IoErrText**

**Syntax**

```
IoErrText(num_errormsg);
```

| num_errormsg | A number specifying the error message. |

**Description**

Returns the error text for IO functions. See also `FileOpen`, `FileClose`, `FileWrite`, `FileWriteLn`, `FileCopy`, `FileMove`, `FileDelete`.

**Returns**

A string.

**Example**

```
$err = fwrite($file,$data);
if(num($err) != 0)
{
    $errtext = ioerrtext($err);
    log(0,"Errortext "+$errtext);
}
```

**IsamAdd**

**Syntax**

```
IsamAdd(isam_file, record);
```

| isam_file | A string specifying the Isam file ‘path’ |
| record | A string specifying the record. |

**Description**

Adds a record to the Isam file ‘path’. The key part of the record must be placed according to the key specified in `IsamCreate`. 
Returns

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OK. Record added successfully.</td>
</tr>
<tr>
<td>-1</td>
<td>Failed. Record NOT added successfully.</td>
</tr>
</tbody>
</table>

Example

```bash
$return = isamadd("order", "1234 $45.50");
```

IsamClose

**Syntax**

```bash
IsamClose(isam_file);
```

**Description**

Closes the specified Isam file.

**Note:** All open Isam files are closed at Job End.

**Returns**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OK.</td>
</tr>
<tr>
<td>-1</td>
<td>Failed.</td>
</tr>
</tbody>
</table>

**Example**

```bash
isamclose("order");
```

IsamCreate

**Syntax**

```bash
IsamCreate(isam_file, num_reclen, key num_pos num_len);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>isam_file</code></td>
<td>A string specifying the name of the Isam file, without the file extension.</td>
</tr>
<tr>
<td><code>num_reclen</code></td>
<td>A number specifying the record length.</td>
</tr>
<tr>
<td><code>num_pos</code></td>
<td>A number specifying the position of the key in the record (starting at 0).</td>
</tr>
<tr>
<td><code>num_len</code></td>
<td>A number specifying the length of the key.</td>
</tr>
</tbody>
</table>
**IsamCreate**

**Description**
Creates an Isam file with the specified record length and the specified key position and length.

**Returns**

<table>
<thead>
<tr>
<th>return code</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OK</td>
</tr>
<tr>
<td>-1</td>
<td>Failed</td>
</tr>
</tbody>
</table>

**Example**

```bash
$rc=isamcreate("order", 30, key 0 10);
```

Creates an Isam file with a record length of 30 characters and a key of 10 characters starting at position 0.

---

**IsamErase**

**Syntax**

```bash
IsamErase(isam_file);
```

**Description**
Removes the specified Isam file.

**Returns**

<table>
<thead>
<tr>
<th>return code</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OK</td>
</tr>
<tr>
<td>-1</td>
<td>Failed</td>
</tr>
</tbody>
</table>

**Example**

```bash
isamerase("order");
```

---

**IsamGet**

**Syntax**

```bash
IsamGet(isam_file, rec_key);
```

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isam_file</td>
<td>A string specifying the name of the Isam file, without the file extension.</td>
</tr>
<tr>
<td>rec_key</td>
<td>A string specifying the record key.</td>
</tr>
</tbody>
</table>
Description
Retrieves a record from the specified Isam file.

Returns
A string containing the value of the record. If the record is not found, the string is empty.

Example
$return = isamget("order", "1234");
Result:
$return = "1234       $45.50"

IsamOpen

Syntax
IsamOpen(isam_file);

| isam_file | A string specifying the name of the Isam file, without the file extension. |

Description
Opens the specified Isam file.

Returns

<table>
<thead>
<tr>
<th>0</th>
<th>OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Failed</td>
</tr>
</tbody>
</table>

Example
isamopen("order");

IsAmount

The IsAmount function is replaced by IsAmountValue.

IsAmountValue

Syntax
IsAmountValue(str_amount, thousand_sep, decimal_sep);

| str_amount | A string specifying the amount. |
| thousand_sep | A string specifying the thousands separator. |
## amountvalue

**Description**
Checks if a string expression is in ‘amount’ format. ‘Amount’ format is a string in the format commonly used to represent e.g. monetary amounts, for example 12,345,678.90

The thousand separator is only allowed every three digits, except in the beginning or at the end of the number. There cannot be any thousand separator after the decimal separator.

For example, `amountvalue("1,200.65", ",", ".")`;

For comparison, `amountvalue("1,2000.65", ",", ".")` is not a valid conversion.

If you leave the separator arguments empty, the separators are not checked. For example, if you leave the thousand separator argument empty, as in `amountvalue("1200.65", ",", ".")`; the number does not need to contain a thousand separator.

The numerical value can contain a minus sign (−) both at the beginning and at the end of the value. A plus sign (+) is only allowed in the beginning.

**Returns**

<table>
<thead>
<tr>
<th>1</th>
<th>True</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>False</td>
</tr>
</tbody>
</table>

**Example**

```script
$var="123,123.45";
if(isamountvalue($var,"","")){
   $ok=1;
}
```

## IsDate

**Syntax**

`IsDate(yyyymmdd)`;

| yyyymmdd | A date string. |

**Description**
Checks if a string expression is a date string.
Returns

<table>
<thead>
<tr>
<th>1</th>
<th>True</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>False</td>
</tr>
</tbody>
</table>

Example

```latex
\$d="19990101";
if(isdate($d))
{  
    \$ok=1;
}
```

IsFirstDocInEnvelope

**Syntax**

```latex
IsFirstDocInEnvelope();
```

**Description**

Checks if the current document is the first document in the envelope. The function can be used at the following post-processing scripting levels:

- Before and after document, process and page.

Returns

<table>
<thead>
<tr>
<th>1</th>
<th>True</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>False</td>
</tr>
</tbody>
</table>

Example

```latex
if(IsFirstDocInEnvelope())
{
    \$ok=1;
}
```

IsFirstPageInEnvelope

**Syntax**

```latex
IsFirstPageInEnvelope();
```

**Description**

Checks if the current page is the first page in the envelope. The function can be used at the following post-processing scripting levels:

- Before and after page.
Returns

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>True</td>
</tr>
<tr>
<td>0</td>
<td>False</td>
</tr>
</tbody>
</table>

Example

```javascript
if(IsFirstPageInEnvelope())
{
    $ok=1;
}
```

IsFirstSegment

**Syntax**

```javascript
IsFirstSegment();
```

**Description**

Checks if the current segment is the first segment of post-processor job.

**Returns**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>True</td>
</tr>
<tr>
<td>0</td>
<td>False</td>
</tr>
</tbody>
</table>

Example

```javascript
if(IsFirstSegment())
{
    $ok = 1;
}
```

IsInsertingPage

**Syntax**

```javascript
IsInsertingPage();
```

**Description**

Checks if the current page is being inserted to the output queue.
Returns

<table>
<thead>
<tr>
<th></th>
<th>True</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>False</td>
</tr>
</tbody>
</table>

Example

```java
if(IsInsertingPage())
{
    InsertPage("SummaryOverlay","SummaryProcessLink");
}
```

IsLastDocInEnvelope

**Syntax**

IsLastDocInEnvelope();

**Description**

Checks if the current document is the last document in the envelope. The function can be used at the following post-processing scripting levels:

Before and after document, process and page.

**Returns**

<table>
<thead>
<tr>
<th></th>
<th>True</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>False</td>
</tr>
</tbody>
</table>

Example

```java
if(IsLastDocInEnvelope())
{
    $ok=1;
}
```

IsLastPageInEnvelope

**Syntax**

IsLastPageInEnvelope();

**Description**

Checks if the current page is the last page in the envelope. The function can be used at the following post-processing scripting levels:

Before and after page.
Returns

<table>
<thead>
<tr>
<th></th>
<th>True</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>False</td>
</tr>
</tbody>
</table>

Example

```javascript
if(IsLastPageInEnvelope())
{
    $ok=1;
}
```

IsLastSegment

Syntax

```javascript
IsLastSegment();
```

Description

Checks if the current segment is the last segment of post-processor job.

Returns

<table>
<thead>
<tr>
<th></th>
<th>True</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>False</td>
</tr>
</tbody>
</table>

Example

```javascript
if(IsLastSegment())
{
    $ok = 1;
}
```

IsNum

Syntax

```javascript
IsNum(str);
```

| str | A string specifying a number. |

Description

Checks if a string expression consists of digits.
Returns

<table>
<thead>
<tr>
<th></th>
<th>True</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>True</td>
</tr>
<tr>
<td>0</td>
<td>False</td>
</tr>
</tbody>
</table>

Example

$n="19990101";$
if(isnum($n))
{
    $ok=1;
}

IsPreview

Syntax

IsPreview();

Description

Checks if the current job is a preview job or not. Preview jobs are all jobs returning a non-zero value from the `GetPreviewType` scripting function.

Returns

A number indicating if a job is a preview job.

<table>
<thead>
<tr>
<th></th>
<th>False. The current job is not a preview job.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>False. The current job is not a preview job.</td>
</tr>
<tr>
<td>&gt;0</td>
<td>True. The current job is a preview job.</td>
</tr>
</tbody>
</table>

Example

See Example 45 on page 82.

J

JobBegin

JobBegin is replaced by `NewJob`.

JobInsertedPages

Syntax

JobInsertedPages();
Description
Returns the number of pages inserted to the output queue within the current job, either by using the `InsertPage()` function or if an address sheet is added when splitting the document into more than one envelope.

Returns

<table>
<thead>
<tr>
<th>num</th>
<th>The number of inserted pages.</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Error</td>
</tr>
</tbody>
</table>

Example

```javascript
$inserted_pages = JobInsertedPages();
```

**JobEnd**

*JobEnd* is replaced by *NewJob*.

**L**

LastBlockInst

**Syntax**

`LastBlockInst();`

**Description**

Determines if the current block is the last of its type.

**Note:** Does not apply in the StreamServe pre-process phase.

Returns

<table>
<thead>
<tr>
<th>1</th>
<th>True</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>False</td>
</tr>
</tbody>
</table>

Example

```javascript
lastblockinst();
```

Where there are two block types in input, b1 and b2:

b10, b11, b12, b20, b13, b14

Result:

True (1) for b12, b20 and b14

In this example, b14 returns True (1) as it is separated from b12 by b20.
LastBlockOnPage

**Syntax**

```plaintext
LastBlockOnPage();
```

**Description**

Determines if the current block is the last block in a frame on each page.

**Note:** This function is not useful during the pre-process phase, where it always returns 0.

**Returns**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>True</td>
</tr>
<tr>
<td>0</td>
<td>False</td>
</tr>
</tbody>
</table>

**Example**

```plaintext

$lb=lastblockonpage();
if(num($lb)==1)
{
    $end=1;
}
```

LdapAddAttrBinaryValue

**Syntax**

```plaintext
LdapAddAttrBinaryValue(entryID, attrName, fileName);
```

**entryID**

The identifier of an entry.

**attrName**

The name of the attribute of interest.

**fileName**

A filename, with optional path.

**Description**

Takes the entryID (returned by the LdapGetEntry function), the attribute name, and a filename, and adds the contents of the file to the given attribute.

LdapAddAttrBinaryValue should be used when there is not yet any data in the attribute.
Returns

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The value was successfully added to the attribute.</td>
</tr>
<tr>
<td>-1</td>
<td>Error. The value could not be added.</td>
</tr>
</tbody>
</table>

Example

`ldapAddAttrBinaryValue($theuser,"photo",my_file);`

In this example, the contents of `my_file` are added to the `photo` attribute of `theuser` entry.

LdapAddAttrValue

**Syntax**

`LdapAddAttrValue(entryID, attrName, value);`

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>entryID</code></td>
<td>The identifier of an entry.</td>
</tr>
<tr>
<td><code>attrName</code></td>
<td>The name of the attribute of interest.</td>
</tr>
<tr>
<td><code>value</code></td>
<td>The new value to be added to the attribute.</td>
</tr>
</tbody>
</table>

**Description**

Adds the specified value to the specified attribute. The changes are not written to the directory until the `LdapUpdateEntry` function is called.

**Returns**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The value was successfully added to the attribute. If the attribute does not already exist it is created.</td>
</tr>
<tr>
<td>-1</td>
<td>Error. The value could not be added.</td>
</tr>
</tbody>
</table>

Example

`ldapAddAttrValue($theuser,"description","$insert");`

In this example, the value of the variable `insert` is added to the `description` attribute of `theuser` entry. For example, if the `description` attribute contains a list of the formats in which the user wants their invoices, and the `insert` variable contains the value `messenger pigeon`, then `messenger pigeon` is added to the list of formats in the `description` attribute.
LdapConnect

Syntax
LdapConnect(connName, hostName, user, password);

<table>
<thead>
<tr>
<th>connName</th>
<th>A name you specify which will be used in all the following LDAP functions to identify which server they will access.</th>
</tr>
</thead>
<tbody>
<tr>
<td>hostName</td>
<td>The name of the server to which you want to establish a connection. The default port is 389, but can be overridden using the host:port syntax in the host name.</td>
</tr>
<tr>
<td>user</td>
<td>The user ID to log on to the server.</td>
</tr>
<tr>
<td>password</td>
<td>The logon password to the server.</td>
</tr>
</tbody>
</table>

Description
Connects to the directory server.

You do not need to use LdapConnect for every scripting function. Once the server connection is established, it will only be closed when you call the LdapDisconnect function, or when the job completes.

To establish an LDAP connection that is not closed when the job completes, see LdapConnectPersistent on page 212.

Leaving the user and password fields blank implies an anonymous logon. The StreamServer will authenticate with anonymous rights (usually a limited read-only authentication).

Timeout limit
You can use the LDAPConnectTimeout keyword to specify a time-out for connecting to the Directory Server. If the connection is not established within the time-out period an error is logged and the LDAPConnect function returns its standard failure code. If no time-out value is specified, the default value 15 seconds is used.

To specify a connection time-out, enter the following keyword to the Platform:
LDAPConnectTimeout <n>
where n is the number of seconds.

Note: The LDAP connection function attempts to open a socket to the server. If the Directory Server cannot be contacted, the time-out will revert to the underlying TCP time-out.

The LDAPConnectTimeout keyword does not work with the LDAPConnectSSL or LDAPConnectSSLCCA functions.
Returns

<table>
<thead>
<tr>
<th>1</th>
<th>A connection to the server has been established successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Error. No connection to the server could be established.</td>
</tr>
</tbody>
</table>

Example
Loading a local file into a binary attribute:

```php
$err = -1;
$myphotofile = "c:\myphotos\myphoto.jpg"; // myPhoto
$myuid = "sab01";
LdapConnect("my_conn", "localhost", "cn=directory manager","streamserve"); // Connect to the ldap server

$results=ldapFind("my_conn","ou=StreamServe Users, dc=streamserve,dc=com", "uid=+$myuid"); // Find my entry matches
$myentry=ldapGetEntry($results, 0); // Get the first entry match
$err = LdapAddAttrBinaryValue($myentry, "photo", $myphotofile); // Add the jpg file to the photo attribute
log(1,"ldapaddattr errorcode= +$err);
$err = LdapUpdateEntry($myentry); // update the attribute
log(1,"ldapupdateentry errorcode= +$err);
LdapDisConnect("my_conn"); //Disconnect
```

In this example, a connection is established to the LDAP server on the local machine (localhost). The connection is given the name con and attaches to the default port (389).

**Note:** The common name (cn) consists of the first and last names of the user.

LdapConnectPersistent

**Syntax**

```php
LdapConnectPersistent(connName, hostName, user, password);
```

<table>
<thead>
<tr>
<th>connName</th>
<th>A name you specify which will be used in all the following LDAP functions to identify which server they will access.</th>
</tr>
</thead>
<tbody>
<tr>
<td>hostName</td>
<td>The name of the server to which you want to establish a connection. The default port is 389, but can be overridden using the host:port syntax in the host name.</td>
</tr>
<tr>
<td>user</td>
<td>The user ID to log on to the server.</td>
</tr>
<tr>
<td>password</td>
<td>The logon password to the server.</td>
</tr>
</tbody>
</table>
Description
Connects to the directory server.

You do not need to use `LdapConnectPersistent` for every scripting function. Once the server connection is established, it will only be closed when you call the `LdapDisconnect` function.

Leaving the user and password fields blank implies an anonymous logon. The StreamServer will authenticate with anonymous rights (usually a limited read-only authentication).

The `LDAPConnectPersistent` function is affected by the a custom setting on the Platform. The `LDAPConnectTimeout` keyword allows you to specify a timeout limit, in seconds, to be used when connecting to a Directory Server. If the connection is not completed within the timeout period an error is logged and the script function returns its standard failure code.

The syntax for the `LDAPConnectTimeout` keyword is

```
LDAPConnectTimeout <n>
```

where `n` is the number of seconds.

**Note:** The LDAP library function for connecting attempts to open a socket to the server before the timeout value comes into play. This means that if the server cannot be contacted, the timeout will revert to the underlying TCP timeout.

Returns

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A connection to the server has been established successfully.</td>
</tr>
<tr>
<td>-1</td>
<td>Error. No connection to the server could be established.</td>
</tr>
</tbody>
</table>

Example
Loading a local file into a binary attribute:

```
$err = -1;
$myphotofile = "c:\myphotos\myphoto.jpg"; // myPhoto
$myuid = "sab01";
LdapConnectPersistent("my_conn", "localhost", "cn=directory manager","streamserve"); // Connect to the ldap server

$results=ldapFind("my_conn","ou=StreamServe Users,dc=streamserve,dc=com", "uid=" +$myuid); // Find my entry matches
$myentry=ldapGetEntry($results, 0); // Get the first entry match
$err = LdapAddAttrBinaryValue($myentry, "photo", $myphotofile); // Add the jpg file to the photo attribute
log(1,"ldapaddattr errorcode=" +$err);
$err = LdapUpdateEntry($myentry); // update the attribute
log(1,"ldapupdateentry errorcode=" +$err);
LdapDisConnect("my_conn"); //Disconnect
```
In this example, a connection is established to the LDAP server on the local machine (localhost). The connection is given the name `con` and attaches to the default port (389).

**Note:** The common name (cn) consists of the first and last names of the user.

### LdapConnectSSL

**Syntax**

```plaintext
LdapConnectSSL(connName, hostName, user, password);
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>connName</td>
<td>A name you specify which will be used in all the following LDAP functions to identify which server they will access.</td>
</tr>
<tr>
<td>hostName</td>
<td>The name of the server to which you want to establish a connection. The default port is 636, but can be overridden using the <code>host:port</code> syntax in the host name.</td>
</tr>
<tr>
<td>user</td>
<td>The user ID to log on to the server.</td>
</tr>
<tr>
<td>password</td>
<td>The logon password to the server.</td>
</tr>
</tbody>
</table>

**Description**

Connects to the directory server using an encrypted connection. To use the `LdapConnectSSL` function, the following argument must have been entered in the argument file:

```
-ldapSslCertDb <path> specifying the path to the cert7.db certificate database
```

You do not need to use `LdapConnectSSL` for every script function. Once the server connection is established, it will only be closed when you apply the `LdapDisconnect` function, or when the job completes.

To establish a secure LDAP connection that is not closed when the job completes, see `LdapConnectSSLPersistent` on page 217.

**Returns**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A connection to the server has been established successfully.</td>
</tr>
<tr>
<td>-1</td>
<td>Error. No connection to the server could be established.</td>
</tr>
</tbody>
</table>

**Example**

```plaintext
LdapConnectSSL("con", "U-Stor-It.company.com:4205", "cn=Directory Manager","streamserve");
```

In this example, an encrypted connection is established to the LDAP server on the machine named `U-Stor-It.company.com`. The connection is given the name `con` and attaches to the port 4205.

**Note:** The common name (cn) consists of the first and last names of the user.
LdapConnectSSLCCA

Syntax
LdapConnectSSLCCA(connName, hostName, certNick, password);

<table>
<thead>
<tr>
<th>connName</th>
<th>A name you specify which will be used in all the following LDAP functions to identify which server they will access.</th>
</tr>
</thead>
<tbody>
<tr>
<td>hostName</td>
<td>The name of the server to which you want to establish a connection. The default port is 636, but can be overridden using the host:port syntax in the host name.</td>
</tr>
<tr>
<td>certNick</td>
<td>Nickname of the Client Certificate used to authenticate the user.</td>
</tr>
<tr>
<td>password</td>
<td>The logon password to the server.</td>
</tr>
</tbody>
</table>

Description
Connects to the directory server using an encrypted connection. A Client Certificate Authentication is provided instead of user/password authentication.

To use the LdapConnectSSLCCA function, the following two arguments must have been entered in the argument file:

-ldapSslCertDb <path> specifying the path to the cert7.db certificate database
-ldapSslKeyDb <path> specifying the path to the key3.db key database

You do not need to use LdapConnectSSLCCA for every script function. Once the server connection is established, it will only be closed when you apply the LdapDisconnect function, or when the job completes.

To establish a secure LDAP connection that is not closed when the job completes, see LdapConnectSSL on page 214.

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A connection to the server has been established successfully.</td>
</tr>
<tr>
<td>-1</td>
<td>Error. No connection to the server could be established.</td>
</tr>
</tbody>
</table>

Example
LdapConnectSSLCCA("con", "U-Stor-It.company.com","cn=Directory Manager","streamserve");

In this example, an encrypted connection is established to the LDAP server on the machine named U-Stor-It.company.com. The connection is given the name con and attaches to the default port (636).

Note: The common name (cn) consists of the first and last names of the user.
LdapConnectSSLCCAPersistent

**Syntax**

LdapConnectSSLCCAPersistent(connName, hostName, certNick, password);

<table>
<thead>
<tr>
<th>connName</th>
<th>A name you specify which will be used in all the following LDAP functions to identify which server they will access.</th>
</tr>
</thead>
<tbody>
<tr>
<td>hostName</td>
<td>The name of the server to which you want to establish a connection. The default port is 636, but can be overridden using the host:port syntax in the host name.</td>
</tr>
<tr>
<td>certNick</td>
<td>Nickname of the Client Certificate used to authenticate the user.</td>
</tr>
<tr>
<td>password</td>
<td>The logon password to the server.</td>
</tr>
</tbody>
</table>

**Description**

Connects to the directory server using an encrypted connection. A Client Certificate Authentication is provided instead of user/password authentication.

To use the LdapConnectSSLCCAPersistent function, the following two arguments must have been entered in the argument file:

- `-ldapSslCertDb <path>` specifying the path to the cert7.db certificate database
- `-ldapSslKeyDb <path>` specifying the path to the key3.db key database

You do not need to use LdapConnectSSLCCAPersistent for every script function. Once the server connection is established, it will only be closed when you apply the LdapDisconnect function.

**Returns**

| -1 | Error. No connection to the server could be established. |
| 1  | A connection to the server has been established successfully. |

**Example**

LdapConnectSSLCCAPersistent("con", "U-Stor-It.company.com", "cn=Directory Manager", "streamserve");

In this example, an encrypted connection is established to the LDAP server on the machine named U-Stor-It.company.com. The connection is given the name con and attaches to the default port (636).

**Note:** The common name (cn) consists of the first and last names of the user.
LdapConnectSSLPersistent

Syntax
LdapConnectSSLPersistent(connName, hostName, user, password);

<table>
<thead>
<tr>
<th>connName</th>
<th>A name you specify which will be used in all the following LDAP functions to identify which server they will access.</th>
</tr>
</thead>
<tbody>
<tr>
<td>hostName</td>
<td>The name of the server to which you want to establish a connection. The default port is 636, but can be overridden using the host:port syntax in the host name.</td>
</tr>
<tr>
<td>user</td>
<td>The user ID to log on to the server.</td>
</tr>
<tr>
<td>password</td>
<td>The logon password to the server.</td>
</tr>
</tbody>
</table>

Description
Connects to the directory server using an encrypted connection. To use the LdapConnectSSLPersistent function, the following argument must have been entered in the argument file:

-ldapsCertDb <path> specifying the path to the cert7.db certificate database

You do not need to use LdapConnectSSLPersistent for every script function. Once the server connection is established, it will only be closed when you apply the LdapDisconnect function.

Returns

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A connection to the server has been established successfully.</td>
</tr>
<tr>
<td>-1</td>
<td>Error. No connection to the server could be established.</td>
</tr>
</tbody>
</table>

Example
LdapConnectSSLPersistent("con", "U-Stor-It.company.com:4205","cn=Directory Manager","streamserve");

In this example, an encrypted connection is established to the LDAP server on the machine named U-Stor-It.company.com. The connection is given the name con and attaches to the port 4205.

Note: The common name (cn) consists of the first and last names of the user.

LdapDisconnect

Syntax
LdapDisconnect(connName);

<table>
<thead>
<tr>
<th>connName</th>
<th>The identifier specified in LdapConnect.</th>
</tr>
</thead>
</table>

Script functions in alphabetical order

Script functions reference

Description
Disconnects from the directory server. Call this function after all other LDAP script functions are run in order to disconnect from the directory server.

Returns

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OK</td>
</tr>
<tr>
<td>-1</td>
<td>Error. Could not disconnect.</td>
</tr>
</tbody>
</table>

Example
LdapDisconnect("con");
In this example, the connection con is disconnected from the directory server.

LdapFind

Syntax
LdapFind(connName, base, search);

<table>
<thead>
<tr>
<th>connName</th>
<th>The identifier specified in the LdapConnect call.</th>
</tr>
</thead>
<tbody>
<tr>
<td>base</td>
<td>The base entry from which to start searching for the specified user ID.</td>
</tr>
<tr>
<td>search</td>
<td>A search string in LDAP search syntax.</td>
</tr>
</tbody>
</table>

Description
Finds entries in the LDAP directory using an LDAP query.

Returns

<table>
<thead>
<tr>
<th>resultID</th>
<th>The function call returns the resultID of the result set, i.e. the set of entries that match the search string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;&quot;</td>
<td>An empty string is returned if the search fails.</td>
</tr>
</tbody>
</table>

Example
$customer=ldapFind("con","dc=streamserve,dc=com", "cn="+$customer_no);  
In this example, the ldapFind function locates all entries that match the specified search string and stores the resultID in the customer variable. The customer variable can then be used in calls to LdapGetEntry to extract the actual entry, or entries, that matched the search.
LdapGetAttrValue

**Syntax**

```plaintext
LdapGetAttrValue(entryID, attrName, valueIdx);
```

<table>
<thead>
<tr>
<th>entryID</th>
<th>The identifier of an entry in a result set. The entry may contain several attributes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrName</td>
<td>The name of the attribute to be retrieved.</td>
</tr>
</tbody>
</table>
| valueIdx | The index of the attribute value. I

**Note:** Indexing is 0 based, i.e. valueIdx 0 is the first value of the attribute, valueIdx 1 is the second value, etc.

**Description**

Retrieves a value from a specific attribute on an entry.

**Returns**

LdapGetAttrValue returns a value of the attribute specified in attrName. An attribute can have several values, and which one is returned depends on the index specified in valueIdx. Note that the indexing is 0 based, i.e. if the specified index is 0, the first value of the attribute is returned.

An empty string is returned if the search fails.

**Example**

```plaintext
$name = ldapGetAttrValue($theuser, "cn", 0);
```

In this example, the first value of the cn attribute of the theuser entry is assigned to the name variable.

LdapGetAttrValueCount

**Syntax**

```plaintext
LdapGetAttrValueCount(entryID, attrName);
```

<table>
<thead>
<tr>
<th>entryID</th>
<th>The ID of a specific entry.</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrName</td>
<td>The name of the attribute of interest.</td>
</tr>
</tbody>
</table>

**Description**

Returns the number of values of a specific attribute on an entry.
Returns

<table>
<thead>
<tr>
<th>num</th>
<th>The number of values for the specific attribute.</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;-1&quot;</td>
<td>The attribute does not exist.</td>
</tr>
</tbody>
</table>

Example

$\text{num}=\text{LdapGetAttrValueCount}(\$\text{theuser}, \text{"mail"});$

In this example, the number of values associated with the \text{mail} attribute of the entryID (\$\text{theuser}) is stored in the \text{num} variable.

LdapGetEntry

Syntax

\text{LdapGetEntry}(\text{resultID}, \text{entryNumber});

<table>
<thead>
<tr>
<th>resultID</th>
<th>The ID representing the result of a search.</th>
</tr>
</thead>
<tbody>
<tr>
<td>entryNumber</td>
<td>The number of the entry in the result set to be retrieved. Note that indexing is 0 based, i.e. \text{entryNumber} 0 is the first entry in the result set, \text{entryNumber} 1 is the second entry, etc.</td>
</tr>
</tbody>
</table>

Description

Retrieves a specific entry from a result set (the result of a query).

Returns

<table>
<thead>
<tr>
<th>entryID</th>
<th>The function call returns the entryID of the string found.</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;&quot;</td>
<td>An empty string is returned if the search fails.</td>
</tr>
</tbody>
</table>

Example

$\text{theuser}=\text{LdapGetEntry}(\$\text{theuserlist}, \text{0});$

In this example, the value of the first entry (index 0) in \text{theuserlist} is assigned to \text{theuser} variable.

LdapGetEntryCount

Syntax

\text{LdapGetEntryCount}(\text{resultID});

<table>
<thead>
<tr>
<th>resultID</th>
<th>The ID representing the result of a search.</th>
</tr>
</thead>
</table>
Description
Returns the number of entries associated with a resultID.

Returns
Number of entries in the result set.

Example
$cust_entry_count=LdapGetEntryCount($customer);
In this example, customer represents the resultID from a search. The number of entries found by the search is stored in the cust_entry_count variable.

LdapGetObjectByDN

Syntax
LdapGetObjectByDN(connName, dn);

<table>
<thead>
<tr>
<th>connName</th>
<th>The identifier specified in the LdapConnect call.</th>
</tr>
</thead>
<tbody>
<tr>
<td>dn</td>
<td>The distinguished name of the object (entry) to be retrieved. For example, the distinguished name might consist of the objects uid, ou, and dc. Note that the parameters must be given in exact order from the leaf entry to the root entry. dc is the first level or root level to be specified, i.e. the organization to which the object belongs, ou is the second level, i.e. the parent node of the object, and the uid is the ID of the specific user or object.</td>
</tr>
</tbody>
</table>

Description
Searches the LDAP directory for a specific object.

Returns

<table>
<thead>
<tr>
<th>entryID</th>
<th>If the search is successful the entryID of the object found. The entryID can be used with subsequent calls to access entry attributes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;&quot;</td>
<td>An empty string is returned if the search fails.</td>
</tr>
</tbody>
</table>

Example
$groupuser=LdapGetObjectByDN("con","uid=123456, ou=People,dc=streamserve,dc=com");
In this example, the ID of the object with username 123456 in the People group, in the streamserve.com organization is stored in the groupuser variable. The variable can then be used in subsequent calls to access the attributes of the object.
LdapGetUser

**Syntax**

LdapGetUser(connName, base, user, attribute, objectClass);

<table>
<thead>
<tr>
<th>connName</th>
<th>The identifier specified in the LdapConnect call.</th>
</tr>
</thead>
<tbody>
<tr>
<td>base</td>
<td>The base entry from which to start searching for the specified user.</td>
</tr>
<tr>
<td>user</td>
<td>The user to find.</td>
</tr>
<tr>
<td>attribute</td>
<td>The ldap attribute used to identify user. Empty string defaults to uid.</td>
</tr>
<tr>
<td>objectClass</td>
<td>The ldap object class used to identify user. Empty string defaults to person.</td>
</tr>
</tbody>
</table>

**Description**

Searches the LDAP directory for a specific user.

**Returns**

<table>
<thead>
<tr>
<th>entryID</th>
<th>If the search is successful an entryID is returned. The entryID can be used with subsequent calls to access the attributes of the entry.</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;&quot;</td>
<td>An empty string is returned if the search fails.</td>
</tr>
</tbody>
</table>

**Example**

```bash
$user=ldapGetUser("con","cn=Users,dc=dev,dc=streamserve,dc=com","Administrator", "cn", "user");
```

LdapNewEntry

**Syntax**

LdapNewEntry(connName, dn);

<table>
<thead>
<tr>
<th>connName</th>
<th>The identifier specified in the LdapConnect call.</th>
</tr>
</thead>
</table>
Script functions in alphabetical order
Script functions reference

LdapNewEntry

Syntax

LdapNewEntry(dn);  

Description

Creates a new entry in the LDAP directory. Changes are not written to the directory until the LdapUpdateEntry function is called.

Returns

<table>
<thead>
<tr>
<th>entryID</th>
<th>An entryID is returned if the new entry was created successfully. The entryID can be used with subsequent calls to access entry attributes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;&quot;</td>
<td>An empty string is returned if the search fails.</td>
</tr>
</tbody>
</table>

Example

```javascript
$theuser=ldapNewEntry("con","uid=123456,ou=People,dc=streamserve,dc=com");
```

In this example a new entry is created and the value is assigned to the theuser variable. The theuser entry contains the value of the distinguished name of the object (in this case we may assume that the object is a new user). The distinguished name consists of the following components: user ID, organization unit (group), and the organization that the user belongs to.

LdapReleaseNewEntries

Syntax

LdapReleaseNewEntries();

Description

To decrease the memory usage, you can call this function which releases memory allocated for all new entries created by LdapNewEntry function calls. This means that you can have a very large job that creates entries and writes them to the LDAP server, without running out of memory.
Returns

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The result set was released successfully.</td>
</tr>
<tr>
<td>-1</td>
<td>Error: The result set could not be released.</td>
</tr>
</tbody>
</table>

Example

```php
$myresults = ldapNewEntry($myConnection, "uid=newperson,dc=streamserve,dc=com");
ldapReleaseNewEntries();
```

LdapReleaseResultSet

**Syntax**

LdapReleaseResultSet(resultID);

| resultID | The ID of the result set to be released. |

**Description**

To decrease the memory usage, you can call this function which frees result sets that are returned from `ldapFind`, `ldapGetUser`, and `ldapGetObjectByDN` function calls. This means that you can have a very large job that queries the LDAP server without running out of memory.

**Returns**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The result set was released successfully</td>
</tr>
<tr>
<td>-1</td>
<td>Error: The result set could not be released</td>
</tr>
</tbody>
</table>

**Example**

```php
$myresults = ldapFind($myConnection, "dc=streamserve,dc=com", "objectClass=*" );
ldapReleaseResultSet($myResults);
```

LdapReplaceAttrBinaryValue

**Syntax**

LdapReplaceAttrBinaryValue(entryID, attrName, fileName);

| entryID | The identifier of an entry. |
| attrName | The name of the attribute whose set of values is to be replaced. |
| fileName | A filename, with optional path. |
Description
Takes the entryID (as returned by LdapGetEntry), the attribute name, and a filename, and adds the contents of the file to the given attribute.

This function works in the same way as LdapAddAttrBinaryValue, except that it replaces any existing data in the attribute.

Returns

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The values were replaced successfully.</td>
</tr>
<tr>
<td>-1</td>
<td>Error. The value could not be replaced.</td>
</tr>
</tbody>
</table>

Example

ldapReplaceAttrBinaryValue($theuser, "photo", _my_file);

In this example, the value of the photo attribute in theuser entry is replaced by the contents of my_file.

LdapReplaceAttrValue

Syntax

LdapReplaceAttrValue(entryID, attrName, value);

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entryID</td>
<td>The identifier of an entry.</td>
</tr>
<tr>
<td>attrName</td>
<td>The name of the attribute whose set of values is to be replaced.</td>
</tr>
<tr>
<td>value</td>
<td>The new value to replace the existing value(s).</td>
</tr>
</tbody>
</table>

Description

Replaces the value(s) of an attribute with another value.

Note: All values of the attribute named in attrName are removed and replaced by the value given in value. To add further values to an attribute you must use the LdapAddAttrValue function. Changes are not written to the LDAP directory until the LdapUpdateEntry function is called.

Returns

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The values were replaced successfully. If the attribute does not already exist it is created.</td>
</tr>
<tr>
<td>-1</td>
<td>Error. The value could not be replaced.</td>
</tr>
</tbody>
</table>

Example

ldapReplaceAttrValue($theuser, "description", $insert);
In this example, the value of the `description` attribute in the `user` entry is replaced by the value of the `insert` variable. For example, if the `description` attribute contains a list of the formats in which the user wants their invoices, and the `insert` variable contains the value `fax`, then all values in the list are removed and `fax` is added.

### LdapSort

**Syntax**

```plaintext
LdapSort(resultId, sortType, attrName_1[, attrName_2... attrNameN]);
```

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>resultId</strong></td>
<td>The result set id returned from LdapFind</td>
</tr>
<tr>
<td><strong>sortType</strong></td>
<td>A string of maximum three characters that specifies the sort type to be performed. You must specify one of the following:</td>
</tr>
<tr>
<td></td>
<td><code>N</code> – Numeric sort</td>
</tr>
<tr>
<td></td>
<td><code>S</code> – String sort</td>
</tr>
<tr>
<td></td>
<td><code>C</code> – String collation</td>
</tr>
<tr>
<td></td>
<td>You can specify the sort order by adding one of the following characters to the string:</td>
</tr>
<tr>
<td></td>
<td><code>A</code> – Ascending (Default)</td>
</tr>
<tr>
<td></td>
<td><code>D</code> – Descending</td>
</tr>
<tr>
<td></td>
<td>If you specify sort type <code>S</code> you can add the following to the string:</td>
</tr>
<tr>
<td></td>
<td><code>I</code> – Case insensitive</td>
</tr>
<tr>
<td><strong>attrNameN</strong></td>
<td>The attribute to compare.</td>
</tr>
</tbody>
</table>

**Description**

Sorts entries in a result set based on one or more entry attributes. The sorting is based on `attrName_1` first, then `attrName_2`, and so on.

**Returns**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The result set was sorted successfully.</td>
</tr>
<tr>
<td>-1</td>
<td>Error. The result set was not sorted successfully.</td>
</tr>
</tbody>
</table>

**Example**

```plaintext
// Sort the search result by string (case insensitive) in ascending order using the "common name" ("cn") attribute.
LdapSort($myResultId, "SAI", "cn");
```
### LdapUpdateEntry

**Syntax**

```javascript
LdapUpdateEntry(entryID);
```

<table>
<thead>
<tr>
<th>entryID</th>
<th>The identifier of an entry.</th>
</tr>
</thead>
</table>

**Description**

Writes any changes that have been made to an entry back to the LDAP server.

**Note:** Changes are not written to the LDAP directory until the `LdapUpdateEntry` function is called.

**Returns**

<table>
<thead>
<tr>
<th>1</th>
<th>The changes made to the entry were successfully written back to the LDAP server.</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Error. The entry could not be updated in the LDAP server.</td>
</tr>
</tbody>
</table>

**Example**

```javascript
ldapUpdateEntry($theuser);
```

In this example, all changes made to `$theuser` variable and its attributes are written back to the LDAP server.

### LdapWriteAttrBinaryValue

**Syntax**

```javascript
LdapWriteAttrBinaryValue(entryID, attrName, fileName);
```

<table>
<thead>
<tr>
<th>entryID</th>
<th>The identifier of an entry.</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrName</td>
<td>The name of the attribute of interest.</td>
</tr>
<tr>
<td>fileName</td>
<td>A filename, with optional path.</td>
</tr>
</tbody>
</table>

**Description**

Writes the contents of the attribute to a file.

**Returns**

<table>
<thead>
<tr>
<th>1</th>
<th>The value was successfully added to the attribute.</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Error. The value could not be written to a file.</td>
</tr>
</tbody>
</table>

**Example**

```javascript
ldapWriteAttrBinaryValue($theuser, "photo", my_file);
```
In this example, the contents of the `photo` attribute of the `user` entry are written to `my_file`.

**Log**

**Syntax**

```
Log(num_loglevel, str);
```

<table>
<thead>
<tr>
<th>num_loglevel</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Log level that specifies when to append a string to the log file.</td>
</tr>
<tr>
<td>0</td>
<td>• 0 corresponds to log level <strong>Severe error messages in the Platform.</strong></td>
</tr>
<tr>
<td>1</td>
<td>• 1 corresponds to log level <strong>All error messages in the Platform.</strong></td>
</tr>
<tr>
<td>2-3</td>
<td>• 2-3 corresponds to log level <strong>All error and warning messages in the Platform.</strong></td>
</tr>
<tr>
<td>4-8</td>
<td>• 4-8 corresponds to log level <strong>All error, warning and information messages in the Platform.</strong></td>
</tr>
<tr>
<td>9</td>
<td>• 9 corresponds to log level <strong>All error, warning and extended information messages in the Platform.</strong></td>
</tr>
</tbody>
</table>

The string is appended only if the log level specified in the Platform is greater than, or equal to, `num_loglevel`.

| str | A string to append to the log file. |

**Description**

Appends a string to the log file.

**Returns**

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OK</td>
</tr>
<tr>
<td>-1</td>
<td>Failed</td>
</tr>
</tbody>
</table>

**Example**

```
log(9,"The number is:"+$inv_nr);
```

Result:

The string "The number is:123456" is appended to the log file only if the log level specified in the Platform is **All error, warning and extended information messages.**
LotusNotesAddNote

Syntax
LotusNotesAddNote();

Description
Adds a Lotus Notes Note to a Lotus Notes database.

Returns

<table>
<thead>
<tr>
<th>0</th>
<th>The Lotus Notes Note was successfully created in the Lotus Notes database.</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Failed to add the Lotus Notes Note to the Lotus Notes database. Possible causes include no connection to the database.</td>
</tr>
</tbody>
</table>

Example
$ret = LotusNotesAddNote();

LotusNotesAttachFile

Syntax
LotusNotesAttachFile(str_path, str_compress);

<table>
<thead>
<tr>
<th>str_path</th>
<th>The file to be attached. You must specify the full path.</th>
</tr>
</thead>
<tbody>
<tr>
<td>str_compress</td>
<td>A boolean option for file compression. If the value is true (not case sensitive), the file will be compressed in the Lotus Notes Note.</td>
</tr>
</tbody>
</table>

Description
Attaches a file to the current Lotus Notes Note. If a file already is attached to the Lotus Notes Note it is replaced by the new file.

A Lotus Notes Note must be available, i.e. opened with the LotusNotesAddNote or the LotusNotesFind functions.

Returns

<table>
<thead>
<tr>
<th>0</th>
<th>The file was successfully attached to the Lotus Notes Note.</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Failed to attach the file to the document. Possible causes include no connection to the database, or that a Lotus Notes Note or the file was not available.</td>
</tr>
</tbody>
</table>

Example
$ret = LotusNotesAttachFile("C:\Files\My_file.pdf", "True");
LotusNotesConnect

Syntax
LotusNotesConnect(str_server, str_database, str_password);

<table>
<thead>
<tr>
<th>str_server</th>
<th>The Lotus Domino server.</th>
</tr>
</thead>
<tbody>
<tr>
<td>str_database</td>
<td>The filename of the Lotus Notes database on the Lotus Domino server.</td>
</tr>
<tr>
<td></td>
<td>If the script requires read access to the Lotus Notes database, the database specified in str_database must allow read access for the user specified in the Lotus ID file. See the str_password argument.</td>
</tr>
<tr>
<td></td>
<td>If the script requires write access to the Lotus Notes database, the database specified in str_database must allow write access for the user specified in the Lotus ID file. See the str_password argument.</td>
</tr>
<tr>
<td></td>
<td>Only a Lotus Domino administrator can enable read or write access to the Lotus Notes database.</td>
</tr>
<tr>
<td>str_password</td>
<td>The password for unlocking the Lotus ID file which authenticates access to a Lotus Notes database.</td>
</tr>
<tr>
<td></td>
<td>The StreamServer searches for the Lotus ID file named ssdominouser.id in the export directory of the StreamServe Project. If the Lotus ID file is not found, or if str_password fails to unlock it, the StreamServer tries to use the previously used Lotus ID file that ssnotes.ini has a reference to.</td>
</tr>
<tr>
<td></td>
<td>If you connect to the database via a Lotus Domino server, without access to a Lotus Notes client installation, you must:</td>
</tr>
<tr>
<td></td>
<td>• Obtain a suitable Lotus ID file.</td>
</tr>
<tr>
<td></td>
<td>• Ensure that the Lotus ID file is recertified and contains a valid password.</td>
</tr>
<tr>
<td></td>
<td>• Rename the Lotus ID file to ssdominouser.id</td>
</tr>
<tr>
<td></td>
<td>• Store the Lotus ID file in the export directory of the Project.</td>
</tr>
<tr>
<td></td>
<td>Ask your Lotus Domino administrator for assistance.</td>
</tr>
</tbody>
</table>

Description
Establishes a connection to a Lotus Notes database on a Lotus Domino server.

If you call the LotusNotesConnect function again, when a connection has been established, the previous connection will be closed. Using the LotusNotesDisconnect function to close a connection ensures that all data written to the database is saved.
Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A connection was successfully established.</td>
</tr>
<tr>
<td>-1</td>
<td>Failed to establish a connection to the Lotus Notes database.</td>
</tr>
</tbody>
</table>

Example

```$ret = LotusNotesConnect("windows_server05/my_domino_server", "invoice.nsf", "my_password");```

LotusNotesDisconnect

**Syntax**

`LotusNotesDisconnect();`

**Description**

Disconnects an existing connection to a Lotus Notes database.

- An established connection will be automatically disconnected at Job End.
- If you call the `LotusNotesConnect` function, all previous connections will be disconnected automatically. Using the `LotusNotesDisconnect` function to close a connection ensures that all data written to the database is saved.

**Returns**

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The connection was successfully disconnected.</td>
</tr>
<tr>
<td>-1</td>
<td>Failed to disconnect.</td>
</tr>
</tbody>
</table>

Example

```$ret = LotusNotesDisconnect();```

LotusNotesFind

**Syntax**

`LotusNotesFind(str_search);`

**Description**

Searches the database for Lotus Notes Notes that match the specified search criteria.

---

*str_search* The search criteria. The criteria must be given in Lotus Notes syntax.
Script functions in alphabetical order

Returns

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>The number of Lotus Notes Notes found that match the search criteria.</td>
</tr>
<tr>
<td>-1</td>
<td>Invalid search syntax.</td>
</tr>
<tr>
<td>-2</td>
<td>The search could not be performed. Possible causes include no connection to the database.</td>
</tr>
</tbody>
</table>

Example

Please refer to the Lotus Notes documentation for search syntax examples.

LotusNotesFirst

Syntax

LotusNotesFirst();

Description

If the LotusNotesFind function finds more than one Lotus Notes Note that matches the search criteria it creates a list of Lotus Notes Notes. The LotusNotesFirst function can be used to return the first Lotus Notes Note in the list.

Returns

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The Lotus Notes Note was successfully returned.</td>
</tr>
<tr>
<td>-1</td>
<td>Failed to perform the search. Possible causes include no connection to the database.</td>
</tr>
<tr>
<td>-2</td>
<td>Failed to find the specified Note.</td>
</tr>
</tbody>
</table>

Example

$ret = LotusNotesFirst();

LotusNotesGet

Syntax

LotusNotesGet(str_field);

| str_field | The name of the field to retrieve data from. |

Description

Retrieves data from a field in a Lotus Notes Note.

The Lotus Notes Note must be available.
Returns

| str       | A string that contains the retrieved data. If the data cannot be retrieved, an empty string is returned. Maximum length of returned string, in UTF-16 code units, is 999. If longer, the returned string is truncated without warning. |

Example

```bash
$invoiceNr = LotusNotesGet("invoiceNumber");
```

LotusNotesLast

**Syntax**

LotusNotesLast();

**Description**

If the LotusNotesFind function finds more than one Lotus Notes Note that matches the search criteria it creates a list of Lotus Notes Notes. The LotusNotesLast function can be used to return the last Lotus Notes Note in the list.

**Returns**

| 0       | The Lotus Notes Note was successfully returned. |
| -1      | Failed to perform the search. Possible causes include no connection to the database. |
| -2      | Failed to find the specified Lotus Notes Note. |

Example

```bash
$ret = LotusNotesLast();
```

LotusNotesNext

**Syntax**

LotusNotesNext();

**Description**

If the LotusNotesFind function finds more than one Note that matches the search criteria it creates a list of Lotus Notes Notes. The LotusNotesNext function can be used to return the next Lotus Notes Note in the list.
Returns

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The Lotus Notes Note was successfully returned.</td>
</tr>
<tr>
<td>-1</td>
<td>Failed to perform the search. Possible causes include no connection to the database.</td>
</tr>
<tr>
<td>-2</td>
<td>Failed to find the specified Lotus Notes Note.</td>
</tr>
</tbody>
</table>

Example

\$ret = LotusNotesNext();

LotusNotesPrev

Syntax

LotusNotesPrev();

Description
If the LotusNotesFind function finds more than one Lotus Notes Note that matches the search criteria it creates a list of Lotus Notes Notes. The LotusNotesPrev function can be used to return the previous Lotus Notes Note in the list.

Returns

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The Lotus Notes Note was successfully returned.</td>
</tr>
<tr>
<td>-1</td>
<td>Failed to perform the search. Possible causes include no connection to the database.</td>
</tr>
<tr>
<td>-2</td>
<td>Failed to find the specified Lotus Notes Note.</td>
</tr>
</tbody>
</table>

Example

\$ret = LotusNotesPrev();

LotusNotesSetDateTime

Syntax

LotusNotesSetDateTime(str_fieldname, str_date);

<table>
<thead>
<tr>
<th>str_fieldname</th>
<th>The item to be updated or added.</th>
</tr>
</thead>
<tbody>
<tr>
<td>str_date</td>
<td>The date and time with which to update the field.</td>
</tr>
</tbody>
</table>
Description
Sets the value of an item in a Note to the date and, optionally, the time specified. A Note must be available, i.e. opened with the LotusNotesAddNote or the LotusNotesFind function.

The date/time format depends on the format set on the Lotus Domino or Lotus Notes installation on the local system on which the StreamServer is running. Three formats are supported (HHMMSS is optional):

- YYYYMMDD[HHMMSS] (recommended)
- DMMYYYY[HHMMSS]
- MMDDYYYY[HHMMSS]

Returns

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The Note was successfully updated.</td>
</tr>
<tr>
<td>-1</td>
<td>Failed to update the Note. Possible causes include no connection to the database.</td>
</tr>
</tbody>
</table>

Example

$ret = LotusNotesSetDateTime("created", "20001231101555");

LotusNotesSetNumber

Syntax

LotusNotesSetNumber(str_fieldname, str_value);

Parameters

- **str_fieldname**
  - The item to be updated or added.
- **str_value**
  - The value with which to update the item. You can use integers and decimal numbers.

Description

Sets the value of an item in a Lotus Notes Note to the specified value.

A Lotus Notes Note must be available, i.e. opened with the LotusNotesAddNote or the LotusNotesFind function.

Returns

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The Lotus Notes Note was successfully updated.</td>
</tr>
<tr>
<td>-1</td>
<td>Failed to update the Lotus Notes Note. Possible causes include no connection to the database.</td>
</tr>
</tbody>
</table>

Example

$ret = LotusNotesSetNumber("invoiceNumber", "123456");
LotusNotesSetText

Syntax
LotusNotesSetText(str_fieldname, str_value);

<table>
<thead>
<tr>
<th>str_fieldname</th>
<th>The item to be updated or added.</th>
</tr>
</thead>
<tbody>
<tr>
<td>str_value</td>
<td>The text to use a data for the item.</td>
</tr>
</tbody>
</table>

Description
Updates an item in a Lotus Notes Note with the specified text.
The Lotus Notes Note must be available, i.e. opened with the LotusNotesAddNote or the LotusNotesFind function.

Returns

| 0   | The Lotus Notes Note was successfully updated. |
| -1  | Failed to update the Lotus Notes Note. Possible causes include no connection to the database. |

Example
$ret = LotusNotesSetText("invoiceReference", "Kalle");

Mat

Syntax
Mat(num_col, num_row, num_len);

<table>
<thead>
<tr>
<th>num_col</th>
<th>Number specifying the column.</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_row</td>
<td>Number specifying the row.</td>
</tr>
<tr>
<td>num_len</td>
<td>Number specifying the number of characters to be copied.</td>
</tr>
</tbody>
</table>

Description
Copies characters from the input matrix. Removes leading and trailing blanks.
Only applicable to PageIN.

Returns
String copied from the matrix.
MatBlk

Syntax
MatBlk(num_col, num_row, num_len);

| num_col | Number specifying the column. |
| num_row | Number specifying the row.    |
| num_len | Number specifying the number of characters to be copied. |

Description
Copies characters from the input matrix. Keeps leading and trailing blanks. Only applicable to PageIN.

Returns
String copied from the matrix.

Example
matblk(5, 7, 25);

MatchCol

Syntax
MatchCol(num_col);

| num_col | A numeric expression specifying the column. |

Description
Returns the content of a specified column. Only available within scripts in StreamIN description files.

Returns
A string containing the contents of the column.

Example
From a RecordIN Description file (*.dsc):
Record "INVOICE" 1 ChrSep ";"
newevent "SalesInvoice";
match script
{
    if(matchcol(1)="0001")
        return 1;
    else if (matchcol(1)="0002")
        return 1;
    else
        return 0;
}
fields
    "Type;                          // 1
    "Invoice number";
    "Page number";
    .....  
end
end

Result:
If the incoming data were:
0001;93127;23.00;990321.....
or
0002;92345;12;40;000101.....
The SalesInvoice Event would be triggered.

MatchPos

Syntax
MatchPos(num_pos, num_length);

<table>
<thead>
<tr>
<th>num_pos</th>
<th>Number specifying the position from which the length of data will be returned.</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_length</td>
<td>Number specifying the length of the data to be returned.</td>
</tr>
</tbody>
</table>

Description
Returns the content of the data from position num_1 with the length num_2. Only available within scripts in StreamIN description files.

Returns
A string containing the contents of the specified position.

Example
From a FieldIN Description file (*.dsc):
SCRIPTFIELD
{
   if (strrblk(matchpos(12,41-12)) == "") {
       if (strrblk(matchpos(42,172-42)) == "")
           return strrblk(matchpos(2,8));
       else
           return strrblk(matchpos(2,8)) + "+" + strrblk(matchpos(42,172-42));
   }
   if (strrblk(matchpos(42,172-42)) == ")
       return strrblk(matchpos(2,8)) + "+" + strrblk(matchpos(12,41-12));
   else
       return strrblk(matchpos(2,8)) + "+" + strrblk(matchpos(12,41-12)) + "+" + strrblk(matchpos(42,172-42));
}

MaxCol

Syntax
MaxCol();

Description
Returns the number of columns in the input matrix. Only applicable to PageIN.

Returns
A number.

Example
$column = maxcol();

MaxRow

Syntax
MaxRow();

Description
Returns the number of rows in the input matrix. Only applicable to PageIN.

Returns
A number.

Example
$row = maxrow();
MkDir

Syntax
MkDir(path);

<table>
<thead>
<tr>
<th>path</th>
<th>A path.</th>
</tr>
</thead>
</table>

Description
Creates a new directory using the specified path.

Returns

<table>
<thead>
<tr>
<th>0</th>
<th>OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Failed</td>
</tr>
</tbody>
</table>

Example
mkdir(C:\StreamServe\Company_ABC);

Mm2In

Syntax
Mm2In(num)

<table>
<thead>
<tr>
<th>num</th>
<th>A number in millimeters to convert to inches.</th>
</tr>
</thead>
</table>

Description
Converts a number given in millimeters to inches unit.

Returns
A number in inches as a floating point value.

Example
$mm = 100;
$inches = mm2in($mm);

Mm2Pt

Syntax
Mm2Pt(num)

<table>
<thead>
<tr>
<th>num</th>
<th>A number in millimeters to convert to points.</th>
</tr>
</thead>
</table>
**Description**
Converts a number given in millimeters to points unit.

**Returns**
A number in points unit as a floating point value.

**Example**
$\text{mm} = 100;
\text{points} = \text{mm2pt}(\text{mm});$

---

**Mod**

**Syntax**

```plaintext
Mod(num_1, num_2)
```

<table>
<thead>
<tr>
<th>num_1</th>
<th>A number being divided.</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_2</td>
<td>A number being the divisor.</td>
</tr>
</tbody>
</table>

**Description**
Returns a number between 0 (inclusive) and num_2 (exclusive) that is the modulus of the arguments.

**Returns**
A number specifying the calculation remainder.

**Example**
```plaintext
mod(106, 10)  Returns: 6
mod(3.5, 1.5) Returns: 0.5
mod(-1, 2)    Returns: 1
mod(-106, 10) Returns: 4
```

---

**Mod10**

**Syntax**

```plaintext
Mod10(str)
```

| str    | A string specifying a number. |

---
Description
Performs a Pad 10-based check digit calculation with the alternating weights 2 and 1, starting at the end of the string. The digits of the values are summed. This check digit calculation method is known as Luhn's algorithm.

Returns
A string specifying a single digit number (0-9).

Example
{
    $billnr="1234567890";
    $checkdigit=mod10($billnr);
    $newbillnr=$billnr+$checkdigit;
}

Result:
$checkdigit is "3"
$newbillnr is "12345678903"

Mod10L

Syntax
Mod10L(str)

<table>
<thead>
<tr>
<th>str</th>
<th>A string specifying a number.</th>
</tr>
</thead>
</table>

Description
Calculates a length check digit and a Luhn check digit. First the length check digit is calculated (length plus 2, mod 10). Then, to calculate the check digit, the length check digit is added at the end of the string and a pad 10-based check digit calculation (same as for Mod10) is performed. Both the length check digit (first) and the pad 10-based check digit are included in the returned result.

Returns
A string specifying a two-digit number consisting of a length check digit and a Luhn check digit.

Example
{
    $noterefnr = "1234567890";
    $modlendig = mod10L($noterefnr);
    $newnoterefnr = $noterefnr + $modlendig;
}

Result:
$modlendig is "23"
Mod11

Syntax

\[ \text{Mod11}(\text{str}) \]

| str          | A string specifying a number. |

Description

Performs a Pad 11-based check digit calculation with the cycled weights 7, 9, 5, 3, 2, 4, 6, 8, starting at the end of the string. Weighted values are summed. A resulting 0 is replaced by a 5, and a resulting 10 is replaced by a 0.

Returns

A string specifying a single digit number (0-9).

Example

```plaintext
$total = "11234567";
$chk = mod11($total);
```

MoveFile

MoveFile is an old name for the `FileMove` function.

MsgBlockId

Syntax

\[ \text{MsgBlockId}(\text{travhandl}) \]

| travhandl | The handle used to identify this specific traversing pass. The handle is specified using the `MsgOpen`, `MsgFrameOpen`, or `MsgFrameOpen` function. |

Description

Returns the name of the current block.
Returns

string The name of the current block. An empty string indicates that the end of the traversed Message was reached.

Example

In this example, the Message contains the blocks blkA, blkB, and blkC structured as shown in the figure below.

```
    root
         |___blkA
             |___blkB
                 |___blkC
             |___blkB
         |___blkA
             |___blkB
```

The following Before Frame script in a PageOUT Process returns the name of the first child block in the first instance of block blkA:

```
$handle=MsgFrameOpen("blkA:1/*", 0);
$blockID=MsgBlockId($handle);
```

MsgClose

Syntax

```
MsgClose(travhandl);
```

travhandl The handle used to identify this specific traversing pass. The handle is specified using the MsgOpen, MsgFrameOpen, or MsgFrameOpen function.

Description

Closes the Message after it has been traversed.

When the Message, or the specific context, has been traversed, the Message is closed. For example, if the context is a frame, the Message is closed when the frame has been traversed. The MsgClose function can be used, for example, to avoid unnecessarily allocating of memory if you need to restart a traversing process.
Returns

| 0 | The Message was successfully closed. |

Example

MsgClose($handle);

MsgCountId

Syntax

MsgCountId(block);

**block**  A string specifying a block.

Description

Counts the number of instances of the specified block in the Message.

Returns

The number of blocks found.

Examples

In the following examples, the Message contains the blocks `blkA`, `blkB`, and `blkC` structured as shown in the figure below.

Example 61  **Script returning the number of blkB blocks.**

The following script returns the number of `blkB` blocks in the Message:

MsgCountId("blkB");  //Returns 4
Example 62  
**Script returning the number of blkB blocks in the first instance of block blkA.**

The following script returns the number of blkB blocks in the first instance of block blkA:

```plaintext```
MsgCountId("blkA:1/blkB"); //Returns 2
```

Example 63  
**Script returning the number of blkB blocks in the \(N\):th instance of block blkA.**

The following script sample includes a counter to specify the \(N\):th instance of block blkA:

```plaintext```
```text```
$counter = 1;
$blkBinA = MsgCountId("blkA: "+ $counter + "/blkB");
$counter++;
...
```

MsgFieldId

**Syntax**

```plaintext```
MsgFieldId(travhandl, num_index);
```

| travhandl | The handle used to identify this specific traversing pass. The handle is specified using the `MsgOpen`, `MsgFrameOpen`, or `MsgFrameOpen` function. |
| num_index | The field index in the block. |

**Description**

Retrieves the name of a field (not field value) in the current block.
Returns

| str | The field name. If the field does not exist, or index is out of range, an empty string is returned. |

Example

In this example, the Message contains the blocks `blkA`, `blkB`, and `blkC` structured as shown in the figure below.

```
root
  blkA
    blkB
      blkC
    blkB
  blkA
    blkB
    blkB
```

The following Before Frame script in a PageOUT Process returns the name of the second field in the first instance of block `blkB` in the first instance of block `blkA`:

```
$handle=MsgFrameOpen("blkA:1/blkB:1", 0);
$numFields=MsgFieldId($handle, 2);
```

MsgFieldValue

**Syntax**

```
MsgFieldValue(travhandl, num_index);
```

| travhandl | The handle used to identify this specific traversing pass. The handle is specified using the `MsgOpen`, `MsgFrameOpen`, or `MsgFrameOpen` function. |
| num_index | The field index in the block. |

**Description**

Returns the value of a field in the current block.
Returns

| str_fieldval | The value of the field. If the field does not exist, or index is out of range, an empty string is returned. |

Examples

In the following examples, the Message contains the blocks `blkA`, `blkB`, and `blkC` structured as shown in the figure below.

```
Example 64  Returning the value of field #2 in the first instance of blkA.

The following Before Frame script in a PageOUT Process returns the value of the second field in the first instance of block `blkA`:

```c
$handle=MsgFrameOpen("**", 0);
$ fieldValue=MsgFieldValue($handle, 2);
```

Example 65  Returning the value of field #2 in the 5:th instance of blkA.

The following Before Frame script returns the value of the second field in the 5:th instance of block `blkA`:

```c
$handle=MsgFrameOpen("blkA:5", 0);
$ fieldValue=MsgFieldValue($handle, 2);
```

Example 66  Returning the value of field #2 in the second instance of blkB in the first instance of blkA

The following Before Frame script returns the value of the second field in the second instance of block `blkB` in the first instance of block `blkA`:

```c
$handle=MsgFrameOpen("blkA:1/blkB:2", 0);
$ fieldValue=MsgFieldValue($handle, 2);
```
MsgFrameCountId

Syntax

MsgFrameCountId(block);

| block          | A string specifying a block in the current frame. |

Description

Counts the number of instances of the specified block in the current frame.

Returns

The number of blocks found.

Examples

In the following examples, the frame contains the blocks blkA, blkB, and blkC structured as shown in the figure below.

Example 67  

Script returning the number of blkB blocks.

The following script returns the number of blkB blocks in the frame:

MsgFrameCountId("blkB"); //Returns 4

Example 68  

Script returning the number of blkB blocks in the first instance of block blkA.

The following script returns the number of blkB blocks in the first instance of block blkA:

MsgFrameCountId("blkA:1/blkB"); //Returns 2

Example 69  

Script returning the number of blkB blocks in the N:th instance of block blkA.

The following script sample includes a counter to specify the N:th instance of block blkA:
$counter = 1;
$blkBlnA = MsgFrameCountId("blkA:" + $counter + "/blkB");
$counter++;

---

**MsgFrameGetValue**

**Syntax**

\[
\text{MsgFrameGetValue}(\text{field});
\]

<table>
<thead>
<tr>
<th>field</th>
<th>A string specifying the name of a field in the current frame.</th>
</tr>
</thead>
</table>

**Description**

Retrieves the value of a specified field in the current frame.

**Returns**

<table>
<thead>
<tr>
<th>str_fieldval</th>
<th>The value of the field. If the field does not exist, an empty string is returned.</th>
</tr>
</thead>
</table>

**Examples**

In the following examples, the frame contains the blocks \texttt{blkA}, \texttt{blkB}, and \texttt{blkC} structured as shown in the figure below.

---

**Example 70  Returning the value of field idType in the second instance of blkA.**

The following Before Frame script returns the value of the field \texttt{idType} in the second instance of block \texttt{blkA}:

\[
\$\text{fieldValue}=\text{MsgFrameGetValue}(\text{"blkA:2/idType"});
\]
Example 71  

**Returning the value of field idType in the second instance of blkB in the first instance of blkA.**

The following Before Frame script returns the value of the field `idType` in the second instance of block `blkB` in the first instance of block `blkA`:

```plaintext
$fieldValue=MsgFrameGetValue("blkA:1/blkB:2/idType");
```

Example 72  

**Returning the value of field idType in the second instance of blkB in the N\:th instance of blkA.**

The following script sample includes a counter to specify the N\:th instance of block `blkA`:

```plaintext
$counter = 1;
$fieldValue=MsgFrameGetValue("blkA:" + $counter +"/blkB:2/idType");
$counter++;
```

**MsgFrameOpen**

**Syntax**

```plaintext
MsgFrameOpen(str_block_address, num_traverse_level);
```

<table>
<thead>
<tr>
<th><code>str_block_address</code></th>
<th>A string specifying the block where to start traversing the Message.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Empty string</strong></td>
<td>If you specify an empty string (&quot;&quot;), the Message traversing will start at the current location.</td>
</tr>
<tr>
<td><strong>Wildcards</strong></td>
<td>The following three examples illustrate how you can use wildcards in the block address string.</td>
</tr>
<tr>
<td>Match the first block in the frame:</td>
<td><code>*</code></td>
</tr>
<tr>
<td>Match the first child block of the block <code>blkA</code>:</td>
<td><code>blkA/*</code></td>
</tr>
<tr>
<td>Match the 5:th instance of the block <code>blkA</code>:</td>
<td><code>blkA:5</code></td>
</tr>
</tbody>
</table>
**Script functions in alphabetical order**

### Script functions reference

<table>
<thead>
<tr>
<th>num_traverse_level</th>
<th>A number specifying the number of levels to traverse, starting from the specified block:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Traverse the entire subtree.</td>
</tr>
<tr>
<td>1</td>
<td>Include blocks at the same level as specified block.</td>
</tr>
<tr>
<td>2</td>
<td>Same as 1, plus child blocks.</td>
</tr>
<tr>
<td>3</td>
<td>Same as 2, plus child blocks of child blocks.</td>
</tr>
<tr>
<td>etc.</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

Defines where in the current frame to start traversing the Message. This function can only be used in PageOUT.

**Returns**

<table>
<thead>
<tr>
<th>travhandl</th>
<th>A handle is returned if the frame was successfully traversed. The handle identifies this specific traversing pass.</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Failed to perform the traversing pass. Possible reasons include no block found, or that the function was called in the wrong context.</td>
</tr>
</tbody>
</table>

**Example**

```bash
$handle=MsgFrameOpen("blkA/blkB", 0);
```

**MsgFrameValueExist**

**Syntax**

`MsgFrameValueExist (field);`

<table>
<thead>
<tr>
<th>field</th>
<th>A string specifying the name of a field in the current frame.</th>
</tr>
</thead>
</table>

**Description**

Checks if the specified field contains a value.
Returns

<table>
<thead>
<tr>
<th></th>
<th>The field value exists.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The field value does not exist.</td>
</tr>
</tbody>
</table>

Example

In the following examples, the frame contains the blocks `blkA`, `blkB`, and `blkC` structured as shown in the figure below.

![Diagram of frame structure with `blkA`, `blkB`, and `blkC` blocks]

**Example 73 Checking if the field `idType` in the second instance of `blkA` contains a value.**

The following script returns 1 if the field `idType` in the second instance of block `blkA` contains a value:

```
$fieldExists=MsgFrameValueExists("blkA:2/idType");
```

**Example 74 Checking if the field `idType` in the second instance of `blkB` in the first instance of `blkA` contains a value.**

The following script returns 1 if the field `idType` in the second instance of block `blkB` in the first instance of block `blkA` contains a value:

```
$fieldExists=MsgFrameValueExists("blkA:1/blkB:2/idType");
```

**Example 75 Checking if the field `idType` in the second instance of `blkB` in the N:th instance of `blkA` contains a value.**

The following script sample includes a counter to specify the N:th instance of block `blkA`:

```
...  
$counter = 1;
$fieldExists=MsgFrameValueExists("blkA:" + $counter + "/blkB:2/idType");
```
$counter++;  
...

**MsgGetValue**

**Syntax**

```
MsgGetValue(field);
```

<table>
<thead>
<tr>
<th>field</th>
<th>A string specifying the name of a field.</th>
</tr>
</thead>
</table>

**Description**

Retrieves the value of a specified field.

**Returns**

<table>
<thead>
<tr>
<th>str_fieldval</th>
<th>The value of the field. If the field does not exist, an empty string is returned.</th>
</tr>
</thead>
</table>

**Examples**

In the following examples, the Message contains the blocks `blkA`, `blkB`, and `blkC` structured as shown in the figure below.

---

**Example 76**  **Returning the value of field idType in the second instance of blkA.**

The following Before Message script returns the value of the field `idType` in the second instance of block `blkA`:

```
$fieldValue=MsgGetValue("blkA:2/idType");
```
Example 77  Returning the value of field idType in the second instance of blkB in the first instance of blkA.

The following Before Message script returns the value of the field idType in the second instance of block blkB in the first instance of block blkA:

```
$fieldValue=MsgGetValue("blkA:1/blkB:2/idType");
```

Example 78  Returning the value of field idType in the second instance of blkB in the N:th instance of blkA.

The following script sample includes a counter to specify the N:th instance of block blkA:

```
...  
$counter = 1;
$fieldValue=MsgGetValue("blkA:" + $counter +"/blkB:2/idType");
$counter++;
...  
```

MsgNextBlock

**Syntax**

MsgNextBlock(travhandl);

| **travhandl** | The handle used to identify this specific traversing pass. The handle is specified using the `MsgOpen`, `MsgFrameOpen`, or `MsgFrameOpen` function. |

**Description**

Moves to the next block.
Script functions reference

**Returns**

<table>
<thead>
<tr>
<th></th>
<th>Successfully moved to the next block.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Failed to move to the next block. Possible reasons include end of Message or invalid handle.</td>
</tr>
</tbody>
</table>

**Example**

In this example, the Message contains the blocks `blkA`, `blkB`, and `blkC` structured as shown in the figure below.

```
$handle=MsgFrameOpen("blkA:1/blkB:1", 0);
$blockID1=MsgBlockId($handle);
MsgNextBlock($handle);
$blockID2=MsgBlockId($handle);
```

**MsgNumFields**

**Syntax**

```
MsgNumFields(travhandl);
```

**Description**

The handle used to identify this specific traversing pass. The handle is specified using the `MsgOpen`, `MsgFrameOpen`, or `MsgFrameOpen` function.

Returns the number of fields in the current block.
Returns

| num_fields | The number of fields in the current block. |

Example

In this example, the Message contains the blocks \texttt{blkA}, \texttt{blkB}, and \texttt{blkC} structured as shown in the figure below.

![Block Diagram]

The following Before Frame script in a PageOUT Process returns the number of fields in the first instance of block \texttt{blkB} in the first instance of block \texttt{blkA}:

```plaintext
$handle=MsgFrameOpen("blkA:1/blkB:1", 0);
$numFields=MsgNumFields($handle);
```
**MsgOpen**

**Syntax**

```
MsgOpen(str_block_address, num_traverse_level);
```

<table>
<thead>
<tr>
<th><strong>str_block_address</strong></th>
<th>A string specifying the block where to start traversing the Message.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Empty string</strong></td>
<td>If you specify an empty string (&quot;&quot;), the Message traversing will start at the current location.</td>
</tr>
<tr>
<td><strong>Wildcards</strong></td>
<td>The following three examples illustrate how you can use wildcards in the block address string.</td>
</tr>
<tr>
<td>Match the first block in the Message:</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td></td>
</tr>
<tr>
<td>Match the first child block of the block blkA:</td>
<td></td>
</tr>
<tr>
<td>blkA/*</td>
<td></td>
</tr>
<tr>
<td>Match the 5:th instance of the block blkA:</td>
<td></td>
</tr>
<tr>
<td>blkA:5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>num_traverse_level</strong></th>
<th>A number specifying the number of levels to traverse, starting from the specified block:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Traverse the entire subtree.</td>
</tr>
<tr>
<td>1</td>
<td>Include blocks at the same level as specified block.</td>
</tr>
<tr>
<td>2</td>
<td>Same as 1, plus child blocks.</td>
</tr>
<tr>
<td>3</td>
<td>Same as 2, plus child blocks of child blocks.</td>
</tr>
<tr>
<td>etc.</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

Defines where to start traversing the Message.
Returns

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$travhandle</td>
<td>A handle is returned if the frame was successfully traversed. The handle identifies this specific traversing pass.</td>
</tr>
<tr>
<td>-1</td>
<td>Failed to perform the traversing pass. Possible reasons include no block found, or that the function was called in the wrong context.</td>
</tr>
</tbody>
</table>

Example

```bash
$handle=MsgOpen("blkA/blkB", 0);
```

**MsgProcCountId**

**Syntax**

```bash
MsgProcCountId(block);
```

<table>
<thead>
<tr>
<th>block</th>
<th>A string specifying a block in the current Process.</th>
</tr>
</thead>
</table>

**Description**

Counts the number of instances of the specified block in the current Process.

**Returns**

The number of blocks found.

**Examples**

In the following examples, the Process contains the blocks `blkA`, `blkB`, and `blkC` structured as shown in the figure below.

```text
Example 79  Script returning the number of blkB blocks.
```

The following script returns the number of `blkB` blocks in the Process:
Example 80  
*Script returning the number of blkB blocks in the first instance of block blkA.*

The following script returns the number of blkB blocks in the first instance of block blkA:

```cpp
MsgProcCountId("blkA:1/blkB"); //Returns 2
```

Example 81  
*Script returning the number of blkB blocks in the N:th instance of block blkA.*

The following script sample includes a counter to specify the N:th instance of block blkA:

```cpp
$counter = 1;
$blkBinA = MsgProcCountId("blkA:" + $counter + "/blkB");
$counter++;
```

**MsgProcGetValue**

**Syntax**

```cpp
MsgProcGetValue(field);
```

<table>
<thead>
<tr>
<th>field</th>
<th>A string specifying the name of a field in the current Process.</th>
</tr>
</thead>
</table>

**Description**

Retrieves the value of a specified field in the current Process.
Returns

| str_fieldval | The value of the field. If the field does not exist, an empty string is returned. |

Examples

In the following examples, the Process contains the blocks `blkA`, `blkB`, and `blkC` structured as shown in the figure below.

```
Example 82 Returning the value of field idType in the second instance of blkA.

The following script returns the value of the field `idType` in the second instance of block `blkA`:

```
$ fieldValue=MsgProcGetValue("blkA:2/idType");
```

Example 83 Returning the value of field idType in the second instance of blkB in the first instance of blkA.

The following script returns the value of the field `idType` in the second instance of block `blkB` in the first instance of block `blkA`:

```
$ fieldValue=MsgProcGetValue("blkA:1/blkB:2/idType");
```

Example 84 Returning the value of field idType in the second instance of blkB in the N:th instance of blkA.

The following script sample includes a counter to specify the N:th instance of block `blkA`:

```
...$counter = 1;
$ fieldValue=MsgProcGetValue("blkA:" + $counter +"/blkB:2/idType");
$counter++;
```
**MsgProcOpen**

**Syntax**

```
MsgProcOpen(str_block_address, num_traverse_level);
```

<table>
<thead>
<tr>
<th><strong>str_block_address</strong></th>
<th>A string specifying the block where to start traversing the Message.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Empty string</strong></td>
<td>If you specify an empty string (&quot;&quot;), the Message traversing will start at the current location.</td>
</tr>
<tr>
<td><strong>Wildcards</strong></td>
<td>The following three examples illustrate how you can use wildcards in the block address string.</td>
</tr>
<tr>
<td>Match the first block in the Process:</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>Match the first child block of the block blkA:</td>
</tr>
<tr>
<td>blkA/*</td>
<td>Match the 5:th instance of the block blkA:</td>
</tr>
<tr>
<td>blkA:5</td>
<td></td>
</tr>
<tr>
<td><strong>num_traverse_level</strong></td>
<td>A number specifying the number of levels to traverse, starting from the specified block:</td>
</tr>
<tr>
<td>0</td>
<td>Traverse the entire subtree.</td>
</tr>
<tr>
<td>1</td>
<td>Include blocks at the same level as specified block.</td>
</tr>
<tr>
<td>2</td>
<td>Same as 1, plus child blocks.</td>
</tr>
<tr>
<td>3</td>
<td>Same as 2, plus child blocks of child blocks.</td>
</tr>
<tr>
<td>etc.</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

Defines where in the current Process to start traversing the Message. This function cannot be used in PageOUT.
Returns

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>travhandl</td>
<td>A handle is returned if the frame was successfully traversed. The handle identifies this specific traversing pass.</td>
</tr>
<tr>
<td>-1</td>
<td>Failed to perform the traversing pass. Possible reasons include no block found, or that the function was called in the wrong context.</td>
</tr>
</tbody>
</table>

Example

$handle=MsgProcOpen("blkA/blkB", 0);

MsgProcValueExist

Syntax

MsgProcValueExist(field);

Description

Checks if the specified field contains a value.

Returns

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The field value exists.</td>
</tr>
<tr>
<td>0</td>
<td>The field value does not exist.</td>
</tr>
</tbody>
</table>

Example

In the following examples, the Process contains the blocks blkA, blkB, and blkC structured as shown in the figure below.

![Diagram of Process structure with blocks blkA, blkB, and blkC](image-url)
Example 85  Checking if the field idType in the second instance of blkA contains a value.

The following script returns 1 if the field idType in the second instance of block blkA contains a value:

\$fieldExists=MsgProcValueExists("blkA:2/idType");

Example 86  Checking if the field idType in the second instance of blkB in the first instance of blkA contains a value.

The following script returns 1 if the field idType in the second instance of block blkB in the first instance of block blkA contains a value:

\$fieldExists=MsgProcValueExists("blkA:1/blkB:2/idType");

Example 87  Checking if the field idType in the second instance of blkB in the N:th instance of blkA contains a value.

The following script sample includes a counter to specify the N:th instance of block blkA:

...  
\$counter = 1;
\$fieldExists=MsgProcValueExists("blkA:" + $counter + "/blkB:2/idType");
\$counter++;
...

MsgValueExist

Syntax

MsgValueExist(field);

| field       | A string specifying the name of a field. |

Description

Checks if the specified field contains a value.
Returns

| 1 | The field value exists. |
| 0 | The field value does not exist. |

Example

In the following examples, the Message contains the blocks blkA, blkB, and blkC structured as shown in the figure below.

```
Example 88 Checking if the field idType in the second instance of blkA contains a value.
```

The following script returns 1 if the field idType in the second instance of block blkA contains a value:

```javascript
$fieldExists=MsgValueExists("blkA:2/idType");
```

```
Example 89 Checking if the field idType in the second instance of blkB in the first instance of blkA contains a value.
```

The following script returns 1 if the field idType in the second instance of block blkB in the first instance of block blkA contains a value:

```javascript
$fieldExists=MsgValueExists("blkA:1/blkB:2/idType");
```

```
Example 90 Checking if the field idType in the second instance of blkB in the N:th instance of blkA contains a value.
```

The following script sample includes a counter to specify the N:th instance of block blkA:

```
... $counter = 1;
$fieldExists=MsgValueExists("blkA:1/blkB:2/idType");
$counter++;
```
NextDoc

**Syntax**

NextDoc();

**Description**

Skips the current document and processes the next document if it exists. The function can be used at the following post-processing scripting levels:

Before and after document, process and page.

**Returns**

<table>
<thead>
<tr>
<th>num</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The document was successfully skipped.</td>
</tr>
<tr>
<td>num&lt;0</td>
<td>Error</td>
</tr>
</tbody>
</table>

**Example**

NextDoc();

NextPage

**Syntax**

NextPage();

**Description**

Skips the current page and processes the next page if it exists. The function can be used at the following post-processing scripting levels:

Before page.

**Returns**

<table>
<thead>
<tr>
<th>num</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The page was successfully skipped.</td>
</tr>
<tr>
<td>num&lt;0</td>
<td>Error</td>
</tr>
</tbody>
</table>

**Example**

NextPage();
NextProc

Syntax

`NextProc();`

Description

Skips the rest of the current Process, and processes the next Process if it exists. The function can be used at the following post-processing scripting levels: Before and after process and page.

Returns

<table>
<thead>
<tr>
<th>num</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The Process was successfully skipped.</td>
</tr>
<tr>
<td>num&lt;&gt;0</td>
<td>Error</td>
</tr>
</tbody>
</table>

Example

`NextProc();`

NextSegment

Syntax

`NextSegment();`

Description

Skips the rest of the current segment, and processes the next segment if it exists.

Returns

<table>
<thead>
<tr>
<th>num</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The segment was successfully skipped.</td>
</tr>
<tr>
<td>num&lt;&gt;0</td>
<td>Error</td>
</tr>
</tbody>
</table>

Example

`NextSegment();`

NewDate

Syntax

`NewDate([yy] yymmdd, num_days);`

<table>
<thead>
<tr>
<th>[yy] yymmdd</th>
<th>A number specifying a date.</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_days</td>
<td>A number specifying a negative or positive number of days.</td>
</tr>
</tbody>
</table>
Script functions reference

Description
Calculates a date from a given date and a negative or positive number of days.

Returns
A string specifying a date in `yyyyMMdd` format.

Example
$\text{a} = "980427";$
$\text{b} = \text{newdate(}$\text{a},15);$
$\text{c} = \text{newdate(}$\text{a},-10);$
Result:
$\text{b} = "19980512"$
$\text{c} = "19980417"$

NewJob

Syntax
\text{NewJob();}

Description
The \text{NewJob} function replaces the \text{JobBegin} and \text{JobEnd} functions, and is used to split large input jobs into smaller jobs.

The \text{NewJob} function must be called from a Retrieved script. The \text{NewJob} function starts a new job at the beginning of the Event. Any previous Events are then treated as a separate job and sent to the pre-process phase.

Note: This scripting function cannot be used when applying dynamic overlays.

Returns
N/A

Example
if (num($\text{count}) > 500)  
{  
\text{NewJob();}  
}  
$\text{count}++;$

NewPage

Syntax
\text{NewPage();}

Description
Generates a new page according to the context in which it is called.
Note: NewPage is not the same as FrameOverflow which affects only the current frame. You can not use the script functions NewPage or FrameOverflow when using the MultiPage function in PageOUT.

You can call NewPage in the following contexts:

- **Script before sheet, Script after sheet**
  A new page of header information, no Frames are started or affected. Useful for activating static reverse page information.
  Note: An infinite loop occurs if there is a bad or no condition for NewPage.

- **Header (field-script in header)**
  No frames are started on the current page.
  Note: An infinite loop occurs if there is a bad or no condition for NewPage.

- **Script before frame, After frame overflow**
  No more frames, including the current frame, are started on the current page.
  Note: An infinite loop occurs if there is a bad or no condition for NewPage.

- **Inside a frame**
  Behaves as FrameOverflow for the current frame. No more frames are started on the current page.

- **Before frame overflow, Script after frame**
  No more frames are started on the current page.
  Note: An infinite loop occurs if there is a bad or no condition for NewPage.

Returns
N/A

Example

```java
if($var="end")
{
    newpage();
}
```
NFormat

Syntax

NFormat(syntax, num);

<table>
<thead>
<tr>
<th>syntax</th>
<th>A string specifying the format syntax.</th>
</tr>
</thead>
<tbody>
<tr>
<td>z –</td>
<td>A digit displayed if required. If the value of a leading or trailing digit is zero, the digit is replaced by a space, &quot; &quot;.</td>
</tr>
<tr>
<td>0 –</td>
<td>A digit displayed if required. If the value of a leading or trailing digit is zero, the digit is removed.</td>
</tr>
<tr>
<td>9 –</td>
<td>A digit that is always displayed.</td>
</tr>
<tr>
<td># –</td>
<td>A digit on either side of a decimal separator, or the last sign if no decimal separator exists. The digit is always displayed.</td>
</tr>
<tr>
<td>Other characters, including spaces, remain unchanged.</td>
<td></td>
</tr>
</tbody>
</table>

Note: 

- `z` and `0` cannot coexist on the same side of the decimal separator. For example, `ZB#.##BZ` is not a valid format.

Note: 

- `#.##` formats the value 0 as an empty string.

| num     | The number to be formatted. |

Description

Converts a number to a string, formatting the string according to the specified format syntax.

Returns

A string according to the specified format syntax.

Example

```plaintext
$a=1234.5;
$b=nformat("Z,ZZ#.##",$a);
$c=nformat("9999#.##",$a);
$d=nformat("ZZ.ZZ",0);
$e=nformat("ZZ.ZZ",0.07);
```

Result:

```plaintext
$b="1,234.50"
$c="01234.50"
$d=""
$e=".07"
```

NoFormFeed

Syntax

NoFormFeed();
**Description**  
Suppresses the form feed sequence output at end of page.

**Returns**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OK</td>
</tr>
<tr>
<td>-1</td>
<td>Failed</td>
</tr>
</tbody>
</table>

**Example**

```plaintext
if(num($var)<20)
{
    noformfeed();
}
```

---

### Num

**Syntax**

```
Num(str);
```

<table>
<thead>
<tr>
<th><strong>str</strong></th>
<th>A string specifying a number.</th>
</tr>
</thead>
</table>

**Description**

Converts a string expression to a numerical value.

**Returns**

A number.

**Example**

```plaintext
num("10");
```

Result: 10 (as a numeric expression)

---

### NumberOfEvents

**Syntax**

```
NumberOfEvents();
```

**Description**

Counts the number of Events in a job. The function is not supported in Postprocessor scripting.

**Returns**

A number indicating the number of Events in the current job.
Example
The following example is a Before-Process script, where there is one Process per Event and the desired output is:

```plaintext
proc1, proc n/2, proc2, procn/2+1, etc
```

```plaintext
if($nproc="")
{
    $nev2=NumberOfEvents() / 2;
    $lowerpart="0";
    $upperpart="1";
}
$nproc++;
if(num($nproc) <= num($nev2))
{
    sort("xx.", $lowerpart);
    $lowerpart += 2;
}
else
{
    sort("xx", $upperpart);
    $upperpart += 2;
}
```

NumberOfEventsEx

Syntax

```plaintext
NumberOfEventsEx(str);
```

| str       | A string specifying an Event. |

Description

Returns the number of the specified Event. The function is not supported in Post-processor scripting.

Returns

A number indicating the number of the Event in the current job.

Example

```plaintext
$eventnumber = numberOfEventsEx("invoice");
```
OdbcBeginTrans

Syntax

```
ODBCBeginTrans(str_dbidentifier);
```

| `str_dbidentifier` | The identifier specified in OdbcConnect. |

Description

Begins a new transaction, which opens two possibilities:

- To commit the transaction, call the `ODBCCommitTrans` function.
- To end the transaction, call the `ODBCRollbackTrans` function.

Returns

N/A

Example

```
ODBCBeginTrans("my_identifier");
```

OdbcCommitTrans

Syntax

```
ODBCCommitTrans(str_dbidentifier);
```

| `str_dbidentifier` | The identifier specified in OdbcConnect. |

Description

Commits a transaction that you have opened with `ODBCBeginTrans`. The changes within the transaction are saved to the database and the transaction is ended.

Returns

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The transaction was successfully saved to the database.</td>
</tr>
<tr>
<td>-1</td>
<td>Error. Problems occurred while trying to save the transaction to the database.</td>
</tr>
</tbody>
</table>

Example

```
ODBCCommitTrans("my_identifier");
```
OdbcConnect

Syntax
OdbcConnect(str_dbidentifier, str_data_source, str_userID, str_password);

<table>
<thead>
<tr>
<th>str_dbidentifier</th>
<th>A name you specify which will be used in all the following ODBC functions to identify which data source they will access.</th>
</tr>
</thead>
<tbody>
<tr>
<td>str_data_source</td>
<td>The name of the ODBC data source to which you want to establish a connection.</td>
</tr>
<tr>
<td>str_userID</td>
<td>The user ID to login to the data source.</td>
</tr>
<tr>
<td>str_password</td>
<td>The login password to the data source.</td>
</tr>
</tbody>
</table>

Description
Connects the user to the data source. You do not need to use OdbcConnect for every scripting function. Once the server connection is established, it will only be closed when you call the OdbcDisconnect function, or when the job completes.

If the data retrieved with ODBCGetFirst, ODBCGetNext, or ODBCGetOne is of Unicode type, you should use the OdbcConnectW (instead of OdbcConnect) when connecting to the data source.

If str_dbidentifier is already associated with a data source, no new connection is established. The existing connection remains open.

Note: If you call ODBCConnect twice without calling ODBCDisconnect in between, ODBCDisconnect will automatically be called before the new connection is made.

Returns

<table>
<thead>
<tr>
<th></th>
<th>A connection to the data source has been established successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>Error. No connection to the data source could be established.</td>
</tr>
</tbody>
</table>

Example
OdbcConnect("my_identifier","address.dsn","sa","abc_1");
**OdbcConnectW**

**Syntax**

```c
OdbcConnectW(str_dbidentifier, str_data_source, str_userID,
str_password, str_codepage);
```

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>str_dbidentifier</td>
<td>A name you specify which will be used in all the following ODBC functions to identify which data source they will access.</td>
</tr>
<tr>
<td>str_data_source</td>
<td>The name of the ODBC data source to which you want to establish a connection.</td>
</tr>
<tr>
<td>str_userID</td>
<td>The user ID to login to the data source.</td>
</tr>
<tr>
<td>str_password</td>
<td>The login password to the data source.</td>
</tr>
<tr>
<td>str_codepage</td>
<td>The code page corresponding to the retrieved data.</td>
</tr>
</tbody>
</table>

**Description**

Connects the user to the data source. If the data retrieved with `ODBCGetFirst`, `ODBCGetNext`, or `ODBCGetOne` is of Unicode type, you must use the `OdbcConnectW` (instead of `OdbcConnect`) when connecting to the data source. You can also set the code page after the connection is established with the `OdbcSetCodepage` function.

You do not need to use `OdbcConnectW` for every scripting function. Once the server connection is established, it will only be closed when you call the `OdbcDisconnect` function, or when the job completes.

If `str_dbidentifier` is already associated with a data source, no new connection is established. The existing connection remains open.

**Note:** If you call `ODBCConnectW` twice without calling `ODBCDisconnect` in between, `ODBCDisconnect` will automatically be called before the new connection is made.

**Returns**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A connection to the data source has been established successfully.</td>
</tr>
<tr>
<td>-1</td>
<td>Error. No connection to the data source could be established.</td>
</tr>
</tbody>
</table>

**Example**

```c
OdbcConnect("my_identifier","address.dsn","sa", "abc_1", "ISO 8859-2");
```
OdbcClearReorderText

OdbcClearReorderText is obsolete.

OdbcCloseQuery

Syntax
OdbcCloseQuery(str_dbidentifier);

| str_dbidentifier | The identifier specified in OdbcConnect. |

Description
Closes any open queries (statements) in the data source. An open query exists when OdbcGetFirst has been executed and all rows in the result set have not been retrieved with OdbcGetNext. Closing the query means that all resources allocated by the query will be released. OdbcGetNext cannot be called after OdbcCloseQuery has been called. This function should only be called when you are certain that you will not want to retrieve any more data from the current query.

If no data is found when OdbcGetFirst or OdbcGetNext tries to retrieve data, these functions will automatically call OdbcCloseQuery to close the query.

Returns

<table>
<thead>
<tr>
<th></th>
<th>All open queries have been closed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>No open query exists.</td>
</tr>
</tbody>
</table>

Example
OdbcCloseQuery("my_identifier");

OdbcDate2Date

Syntax
OdbcDate2Date(str_date_time);

| str_date_time | A string containing date and time. |

Description
Extracts the date from a date and time string in the following format:

date
time
Returns

| date         | The date extracted from a string containing date and time. |

Example

A column of the smalldatetime SQL data type could be returned from the data source as:

2002-01-01 15:48:00

In this case, `OdbcDate2Date` would return:

2002-01-01

OdbcDate2Time

Syntax

OdbcDate2Time(str_date_time);

| str_date_time | A string containing date and time. |

Description

Extracts the time from a date and time string in the following format:

date time

Returns

| time         | The time extracted from a string containing date and time. |

Example

A column of the smalldatetime SQL data type could be returned from the data source as:

2002-01-01 15:48:00

In this case, `OdbcDate2Date` would return:

15:48:00

OdbcDisconnect

Syntax

OdbcDisconnect(str_dbidentifier);

| str_dbidentifier | The identifier specified in `OdbcConnect`. |
Description
Disconnects the server from the data source. You use this function after all the other script functions that you want to carry out on a StreamServe Process.

If you do not call the OdbcDisconnect function, StreamServe automatically disconnects from the data source when StreamServe is terminated.

Returns

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The server has been disconnected from the data source.</td>
</tr>
<tr>
<td>-1</td>
<td>Error. The server could not be disconnected from the data source.</td>
</tr>
</tbody>
</table>

Example

OdbcDisconnect("my_identifier");

OdbcExecute

Syntax

OdbcExecute(str_dbidentifier, str_sql_statement);

<table>
<thead>
<tr>
<th>str_dbidentifier</th>
<th>The identifier specified in OdbcConnect.</th>
</tr>
</thead>
<tbody>
<tr>
<td>str_sql_statement</td>
<td>The SQL statement to execute.</td>
</tr>
</tbody>
</table>

Description

Executes SQL statements. Any result sets that the statement generates are ignored. This function can be used, for example, to update record fields in the data source. To use this function, you must first write the SQL statement and give it an identifier. You then use the OdbcExecute to execute the statement.

See also OdbcExecuteEx.

Returns

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The statement was executed successfully.</td>
</tr>
<tr>
<td>-1</td>
<td>Error. An error occurred while trying to execute the statement.</td>
</tr>
</tbody>
</table>

Example

$st="INSERT INTO CUSTOMER(CNR)VALUES("+$custid+"));"
OdbcExecute("my_identifier",$st);

In this script, the customer ID is stored in a variable. An SQL statement is written using the value of this variable. $st would contain something like the following when OdbcExecute is called:

INSERT INTO CUSTOMER (CNR) VALUES (124);
OdbcExecuteEx

Syntax

OdbcExecuteEx(str_dbidentifier, str_sql_statement, num);

<table>
<thead>
<tr>
<th>str_dbidentifier</th>
<th>The identifier specified in OdbcConnect.</th>
</tr>
</thead>
<tbody>
<tr>
<td>str_sql_statement</td>
<td>The SQL statement to execute.</td>
</tr>
</tbody>
</table>
| num              | • If num = 0, StreamServer will not stop if a CREATE TABLE statement fails.  
                 | • If num is not = 0, this function works as OdbcExecute.                  |

Description

Same as OdbcExecute. The difference is that this function can be configured to accept that a CREATE TABLE statement fails.

For example, if you use the OdbcExecute function to create a table that already exists, an error will be received, and the StreamServer will stop. If you use the OdbcExecuteEx statement instead, and set num = 0, the received error will be ignored and the StreamServer will not stop.

OdbcGetOne

Syntax

OdbcGetOne(str_dbidentifier, str_sql_statement, var_1[, var_2, var_n]);

<table>
<thead>
<tr>
<th>str_dbidentifier</th>
<th>The identifier specified in OdbcConnect.</th>
</tr>
</thead>
<tbody>
<tr>
<td>str_sql_statement</td>
<td>The SQL statement to execute.</td>
</tr>
<tr>
<td>var_n</td>
<td>The variables that will store the data.</td>
</tr>
</tbody>
</table>

Description

Executes an SQL statement and copies the values from the first row in the result set to script variables. The query is closed when the first row has been retrieved, and all resources allocated by the query will be released. Using this function is equivalent to calling OdbcGetFirst and then calling OdbcCloseQuery.
Returns

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The statement was executed successfully.</td>
</tr>
<tr>
<td>0</td>
<td>No data found. The statement did not return any result set. The variables are cleared.</td>
</tr>
<tr>
<td>-1</td>
<td>Error. An error occurred while trying to execute the statement.</td>
</tr>
</tbody>
</table>

Example

```
$st="SELECT CUSTNAME FROM CUSTOMER WHERE CUSTID="+$cnr + ";";
OdbcGetOne("my_identifier", $st, $custname);
```

OdbcGetFirst

**Syntax**

```
OdbcGetFirst(str_dbidentifier, str_sql_statement, var_1[, var_2, var_n]);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>str_dbidentifier</td>
<td>The identifier specified in OdbcConnect.</td>
</tr>
<tr>
<td>str_sql_statement</td>
<td>The SQL statement to execute.</td>
</tr>
<tr>
<td>var_n</td>
<td>The variables that will store the data.</td>
</tr>
</tbody>
</table>

**Description**

Executes an SQL statement and copies the values from the first row in the result set to the specified script variables. The OdbcGetNext function is then used to retrieve the remaining parts of the result set.

The value of the first data source column of the result set will be copied to the first variable in the list of variables. The second column will be copied to the second variable, and so on.

If the number of variables is less than the number of columns in the result set, the last columns that do not have matching variables will be ignored.

If the number of columns in the result set is less than the number of variables, the last variables that do not have matching columns will be cleared.

If the result set from the SQL statement is empty, all variables will be cleared.
Returns

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The statement was executed successfully.</td>
</tr>
<tr>
<td>0</td>
<td>No data found. The statement did not return a result set. The variables are cleared.</td>
</tr>
<tr>
<td>-1</td>
<td>Error. An error occurred while trying to execute the statement.</td>
</tr>
</tbody>
</table>

Example

$st="SELECT CUSTNAME FROM CUSTOMER WHERE CUSTID="$cnr + ";";
OdbcGetFirst("my_identifier", $st, $custname);

In this example, the data source, my_identifier contains the CUSTOMER table. The $cnr variable should contain the customer ID, which is the key in the CUSTOMER table.

The SQL statement is constructed using the $cnr variable, and OdbcGetFirst is called to retrieve the first row of the CUSTNAME column from the data source.

OdbcGetNext

Syntax

OdbcGetNext(str_dbidentifier, var_1[, var_2, var_n]);

<table>
<thead>
<tr>
<th>str_dbidentifier</th>
<th>The identifier specified in OdbcConnect.</th>
</tr>
</thead>
<tbody>
<tr>
<td>var_n</td>
<td>The variables that will store the data.</td>
</tr>
</tbody>
</table>

Description

Retrieves the values of the columns in the second row of a result set. This function can only be called after OdbcGetFirst has been called to retrieve the first row in the result set. OdbcGetNext can then be called repeatedly to process all the remaining rows. When all the rows have been processed, no data will be found and OdbcGetNext will return 0 (zero).

The value of the first column of the result set will be copied to the first variable in the list of variables. The second column will be copied to the second variable and so on.

If the number of variables is less than the number of columns in the result set, the last columns that do not have matching variables will be ignored.

If the number of columns in the result set is less than the number of variables, the last variables that do not have matching columns will be cleared.

If the result set from the SQL statement is empty, all variables will be cleared.
Script functions in alphabetical order

Returns

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The statement was executed successfully.</td>
</tr>
<tr>
<td>0</td>
<td>No data found. The statement did not return any result set. The variables are cleared.</td>
</tr>
<tr>
<td>-1</td>
<td>Error. An error occurred while trying to execute the statement.</td>
</tr>
</tbody>
</table>

Example

```perl
$st="SELECT CUSTNAME FROM CUSTOMER WHERE CITY='Gothenburg';";
$ret=OdbcGetFirst("my_identifier",$st,$custname);
while(Num($ret)>0){
    callblock("customer");
    $ret=OdbcGetNext("my_identifier",$custname);
}
```

The CUSTNAME column is retrieved for all customers that have the CITY column set to Gothenburg in the CUSTOMER table.

The callblock function is a StreamServe scripting function. It is used to call a free block. In this example, a free block with the name "customer" will be called for each retrieved customer.

OdbcRollbackTrans

Syntax

```perl
OdbcRollbackTrans(str_dbidentifier);
```

| str_dbidentifier | The identifier specified in OdbcConnect. |

Description

Rolls back and ends a transaction that you have opened with ODBCBeginTrans.

Returns

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The transaction was successfully rolled back.</td>
</tr>
<tr>
<td>-1</td>
<td>Error. Problems occurred while trying to roll back the transaction.</td>
</tr>
</tbody>
</table>

Example

```perl
OdbcRollbackTrans("my_identifier");
```
OdbcSetCodepage

**Syntax**

```c
OdbcSetCodepage(str_dbidentifier, str_codepage);
```

<table>
<thead>
<tr>
<th><strong>str_dbidentifier</strong></th>
<th>A name you specify which will be used in all the following ODBC functions to identify which data source they will access.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>str_codepage</strong></td>
<td>The code page corresponding to the retrieved data.</td>
</tr>
</tbody>
</table>

**Description**

Sets the code page corresponding to the data retrieved from a data source.

**Note:** If the specified code page does not exist the server will terminate.

**Returns**

N/A

**Example**

```c
OdbcSetCodepage("my_identifier", "ISO 8859-2");
```

OdbcSetReorderText

**OdbcSetReorderText** is obsolete.

OutputFile

**Syntax**

```c
OutputFile(str_filename);
```

| **str_filename** | A string containing the path and the name of the file whose contents are to be sent to the output connector. |

**Description**

The `OutputFile` function is used in a Process to send the contents of a file directly to an output connector, without changing or modifying the contents. It can also be used for concatenating or encapsulating already existing data.

**Note:** You must not specify any Device driver for the output connector to which the file is sent. The `OutputFile` function does not work with PageOUT, since PageOUT requires a device to be set on the connector.

See also the `OutputString` function.
Returns

<table>
<thead>
<tr>
<th></th>
<th>The contents were successfully sent to the connector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>An error occurred. This may result if the file cannot be opened or read.</td>
</tr>
</tbody>
</table>

Example

Assume that you have an XMLOUT Process that produces the following output data:

```html
<html>
<head>Tender</head>
<body>
<h1>Vitamins Inc. kindly offers the following products</h1>
</body>
</html>
```

You also have a file called fruits.txt that contains a price list:

```html
<table>
<tr><td>Oranges</td><td>17.95</td></tr>
<tr><td>Bananas</td><td>14.95</td></tr>
<tr><td>Lemons</td><td>22.25</td></tr>
</table>
```

Instead of creating the table every time the XMLOUT Process is executed, you can include the price list in the output by calling the `fruits.txt` file from within the XMLOUT Process. Enter the following line in a script that is executed after the `<h1>` element:

```script
OutputFile ("fruits.txt");
```

The output sent to the connector will be:

```html
<html>
<head>Tender</head>
<body>
<h1>Vitamins Inc. kindly offers the following products
<table>
<tr><td>Oranges</td><td>17.95</td></tr>
<tr><td>Bananas</td><td>14.95</td></tr>
<tr><td>Lemons</td><td>22.25</td></tr>
</table>
</body>
</html>
```
OutputLXFJobResource

**Syntax**

```javascript
OutputLXFJobResource(num_jobID, str_name, num_index);
```

<table>
<thead>
<tr>
<th>num_jobID</th>
<th>The ID of the job in which the LXF job resource was created.</th>
</tr>
</thead>
<tbody>
<tr>
<td>str_name</td>
<td>The name of the job resource that contains the LXF overlay you want to add to the PageOUT Process.</td>
</tr>
<tr>
<td>num_index</td>
<td>The job resource index.</td>
</tr>
</tbody>
</table>

**Description**

Adds an LXF overlay from a job resource to a PageOUT Process.

Currently, this function only works with PageOUT.

See also the Job Resource output connector settings, and the OutputLXFJobResourcePrnOffs, GetJobResourceIndex and DeleteJobResource functions.

**Note:** Not supported in Post-processor scripting.

**Returns**

<table>
<thead>
<tr>
<th>num&gt;0</th>
<th>The LXF overlay has been successfully added to the Process.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Failure.</td>
</tr>
</tbody>
</table>

**Example**

Assume that the LXF Resource was created within job 122, and stored in a job resource named MyLXFResourceConnector, and that the job resource index is 0.

Enter the following in the PageOUT field where you want to add the LXF file:

```
OutputLXFJobResource(122, "MyLXFResourceConnector", 0);
```

OutputLXFJobResourcePrnOffs

**Syntax**

```javascript
OutputLXFJobResourcePrnOffs(num_jobID, str_name, num_index);
```

<table>
<thead>
<tr>
<th>num_jobID</th>
<th>The ID of the job in which the LXF job resource was created.</th>
</tr>
</thead>
<tbody>
<tr>
<td>str_name</td>
<td>The name of the job resource that contains the LXF overlay you want to add to the PageOUT Process.</td>
</tr>
<tr>
<td>num_index</td>
<td>The job resource index.</td>
</tr>
</tbody>
</table>
Script functions reference

OutputLXFJobResourcePrnOffs

Description
Adds an LXF overlay from a job resource to a PageOUT Process, and enables the use of SetPrnXOffs and SetPrnYOffs to adjust the overlay offset.

Currently, this function only works with PageOUT.

See also the Job Resource output connector settings, and the GetJobResourceIndex and DeleteJobResource functions.

Note: Not supported in Post-processor scripting.

Returns

<table>
<thead>
<tr>
<th>num&gt;0</th>
<th>The LXF overlay has been successfully added to the Process.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Failure.</td>
</tr>
</tbody>
</table>

Example

SetPrnXOffs (5);
SetPrnYOffs (2);
OutputLXFJobResourcePrnOffs (getInJobID(), "LXFJobResource", $pageNr);

OutputString

Syntax

OutputString(str);

| str    | A string or variable containing a string of text.       |

Description

Outputs a line of text directly to the output buffer of a StreamOUT or XMLOUT Process. The script can be placed before and after Processes and inside Processes.

Note: Only available for StreamOUT and XMLOUT Processes. Any other Process type will ignore the script and a log message will be written.

In StreamOUT, the character separator inside records will be ignored before and after the script string, so if the script is executed between two fields in a record that is not fixed positioned then no separator will be written between the fields. If you want to include the separator, it will have to be included in the script.

Note: Special characters (such as carriage return, line feed, tabs etc.) can be written to the stream using the hex values for these characters.
Returns

<table>
<thead>
<tr>
<th></th>
<th>String was written successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>String was not written successfully.</td>
</tr>
</tbody>
</table>

Example

The script can be used in an XMLOUT Process to output a variable in the XML prologue. Insert the script after the XML Header or the XML document type. For example:

```plaintext
outputstring($fieldvariable);
```

The `$fieldvariable` variable is assigned the value of a field which is displayed as text in the XML prologue.

OverlayGetNumPages

Syntax

```plaintext
OverlayGetNumPages(filename);
```

| filename | Path and name of the overlay file. |

Description

Returns the total number of pages in a multi-page overlay.

Returns

<table>
<thead>
<tr>
<th>num</th>
<th>The number of pages.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Error. The number of pages could not be retrieved.</td>
</tr>
</tbody>
</table>

Example

```plaintext
$totnum=OverlayGetNumPages("MultiPageOverlay.tiff");
```

Page

Syntax

```plaintext
Page();
```

Description

Returns the number of the current page.
Returns

<table>
<thead>
<tr>
<th>num</th>
<th>The number of the current page.</th>
</tr>
</thead>
</table>

Example

```javascript
$pg=page();
if(num($pg)=1)
{
    $text="First";
}
```

PageOfPages

**Syntax**

`PageOfPages();`

**Description**

Returns a string specifying the current page number and the total number of pages.

**Returns**

A string in the format `current_page(total_pages)`.

**Example**

```javascript
$pop=pageofpages();
```

Pages

**Syntax**

`Pages();`

**Description**

Returns the number of pages in the Process. Not applicable in the StreamServe pre-process phase.

For StoryTeller Processes, the function must be called within a Content Dependent Fragment, otherwise there is no way to know how many pages the document will contain.

**Returns**

<table>
<thead>
<tr>
<th>num</th>
<th>The number of pages in the Process.</th>
</tr>
</thead>
</table>

**Example**

```javascript
$pgs=pages();
```
if(num($pgs)=3)
{
    $count=3;
}

PatternResult

Syntax
PatternResult(pattern_id);

| pattern_id | A string specifying a pattern ID. |

Description
You can use PatternResult to check for a pattern in a Message or in a block. Only applicable to PageIN.

The PatternResult function must be invoked within a function in a Rule in a PageIN Event. You can invoke a function in the function file using the syntax func=funcname.

Returns

| 1 | The pattern was found. |
| 0 | The pattern was not found. |

Example
If you are checking for a pattern which is in a Message, enter a Rule for the Message in the PageIN Event:

//Invoke a function within a rule
func=patterncheck()

//patterncheck is a custom function in the function file

//In function file
func patterncheck()
{
    $result = patternresult ("id_invoice");
    log(9, "Pattern match for id_invoice = " + $result);
    //returns 1 if pattern is matched else 0
    $result = patternresult ("id_article");
    log(9, "Pattern match for id_article = " + $result);
}
PPDocCount

**Syntax**

```
PPDocCount();
```

**Description**

Returns the number of Documents sent to the output connector.

**Returns**

<table>
<thead>
<tr>
<th>num</th>
<th>The number of Documents sent to the output connector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Error</td>
</tr>
</tbody>
</table>

**Example**

```
$doc_count = PPDocCount();
```

PreProc

**Syntax**

```
PreProc();
```

**Description**

Determines whether or not StreamServer is running in pre-process phase.

**Returns**

<table>
<thead>
<tr>
<th>1</th>
<th>True</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>False</td>
</tr>
</tbody>
</table>

**Example**

```c
//!multicolumn!
agatha christie 1890
alfred hitchcock 1899

//Check if the server runs in preprocess phase
//If it does, perform the substitution
if (PreProc() = 1)
{
    $newborn = AddSubst("author.tbl", "alfred", 1, "100");
}
```

This changes the table to show that Alfred was born 1999 if the server runs in the preprocess phase.
PreProcLog

Syntax

PreProcLog(num_loglevel, str);

<table>
<thead>
<tr>
<th>num_loglevel</th>
<th>Log level that specifies when to append a string to the log file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 corresponds to log level Severe error messages in the Platform.</td>
</tr>
<tr>
<td>1</td>
<td>1 corresponds to log level All error messages in the Platform.</td>
</tr>
<tr>
<td>2-3</td>
<td>2-3 corresponds to log level All error and warning messages in the Platform.</td>
</tr>
<tr>
<td>4-8</td>
<td>4-8 corresponds to log level All error, warning and information messages in the Platform.</td>
</tr>
<tr>
<td>9</td>
<td>9 corresponds to log level All error, warning and extended information messages in the Platform.</td>
</tr>
</tbody>
</table>

The string is appended only if the log level specified in the Platform is greater than, or equal to, num_loglevel.

| str          | Text to be appended to the log file. |

Description

During the pre-process phase, appends a string to the log file.

Returns

<table>
<thead>
<tr>
<th>0</th>
<th>OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Failed</td>
</tr>
</tbody>
</table>

Example

preproclog(1,"Jan 2000");

If the StreamServer was started with loglevel=1 or greater, the string "Jan 2000" is added to the log.

Pt2In

Syntax

Pt2In(num)

| num | A number in points to convert to inches. |

Description

Converts a number given in points unit to inches.
Returns
A number in inches as a floating point value.

Example
$pt = 100;
$inches = pt2in($pt);

Pt2Mm

Syntax
Pt2Mmm(num)

| num | A number in points unit to convert to millimeters. |

Description
Converts a number given in points unit to millimeters.

Returns
A number in points unit as a floating point value.

Example
$pt = 100;
$mm = mm2in($pt);

Q

QueuePage
QueuePage is replaced by ConnectorPage.

QueuePages
QueuePages is replaced by ConnectorPages.
ReadSubst

Syntax
ReadSubst(tbl_file);

tbl_file  A string specifying the path and filename of the substitution table.

Description
Re-reads a substitution table from disk. The substitution table is re-read only if
the physical file has been changed since it was last read to memory.

Returns

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not re-read</td>
</tr>
<tr>
<td>1</td>
<td>Re-read</td>
</tr>
<tr>
<td>-1</td>
<td>Error (the path could not be found)</td>
</tr>
</tbody>
</table>

Example
In this example, the following substitution table (author.tbl) is used.

```plaintext
//!CodePage UTF8!
//!multicolumn!
agatha christie 1890
alfred hitchcock 1899
```

In the following script, ReadSubst is used to determine whether author.tbl has
changed. If it has, $test will be set to 1, and $nameSur will be set to christie.
If the table is not change, $test will be set to 0, and $nameSur will be set to an
empty string.

```plaintext
if (preproc() = 0)
{
    $test = ReadSubst("../data/tables/author.tbl");
    if (Num($test) = 1)
    {
        $nameSur = Subst("../data/tables/author.tbl", "agatha");
    }
}
```

Notes
- A substitution table is kept in memory until the script function EraseSubst is
called for that table.
• You must set up a condition to only execute the function when not in pre-process phase.

ReplaceJobIDAttribute

ReplaceJobIDAttribute is an old function and is removed.

ReplaceJobIDDateAttribute

ReplaceJobIDDateAttribute is an old function and is removed.

ReplaceJobIDNumAttribute

ReplaceJobIDNumAttribute is an old function and is removed.

RGBcolor

Syntax

RGBcolor(red, green, blue);

<table>
<thead>
<tr>
<th>red</th>
<th>A number specifying the level of red color (0-255).</th>
</tr>
</thead>
<tbody>
<tr>
<td>green</td>
<td>A number specifying the level of green color (0-255).</td>
</tr>
<tr>
<td>blue</td>
<td>A number specifying the level of blue color (0-255).</td>
</tr>
</tbody>
</table>

Description

Specifies an RGB color. For example, the color of a font or an area.

Returns

An RGB color, defined in an internal StreamServe format.

Example

if ($country="ENG")
   $color1=RGBcolor(255,0,0);
else
   $color1=RGBcolor(0,0,255);

Round

Syntax

Round(num, decimals);

<table>
<thead>
<tr>
<th>num</th>
<th>A number to be rounded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>decimals</td>
<td>A number of decimals.</td>
</tr>
</tbody>
</table>
Description
Returns the first argument rounded to the number of decimals specified in the second argument.

Returns
A number.

Example
$a=12.5567;
$b=round($a,2);

Result:
$b=12.56

SaveCurrMetadata

Syntax
SaveCurrMetadata(metadata_name);

| metadata_name | A string specifying the name of the metadata associated with the current document. |

Description
Saves the metadata for the current document in the repository. The function can be used at the following post-processing scripting levels:
Before and after document, process and page.

Note: If the metadata is not used with enveloping or sorting, the metadata must be declared with the DeclareMetadata() function at Before Job level.

Returns

<table>
<thead>
<tr>
<th>num</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Metadata is saved</td>
</tr>
<tr>
<td>num&lt;&gt;0</td>
<td>Error</td>
</tr>
</tbody>
</table>

Example
$ret = SaveCurrMetadata("zip");
SavePPMetadata

**Syntax**

```
SavePPMetadata(doc_handle, metadata_name);
```

<table>
<thead>
<tr>
<th>doc_handle</th>
<th>Identifies the document.</th>
</tr>
</thead>
<tbody>
<tr>
<td>metadata_name</td>
<td>A string specifying the name of the metadata associated with the document specified in the <code>doc_handle</code> parameter.</td>
</tr>
</tbody>
</table>

**Description**

Saves the metadata for the document specified in the `doc_handle` parameter returned by either the `GetFirstPPDoc()` or the `GetNextPPDoc()` function.

If the metadata is not used with enveloping or sorting, the metadata must be declared with the `DeclareMetadata()` function at Before Job level.

**Returns**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The metadata is saved</td>
</tr>
<tr>
<td><code>num&lt;&gt;0</code></td>
<td>Error</td>
</tr>
</tbody>
</table>

**Example**

```
$doc_id = GetFirstPPDoc();
while(num($doc_id) > 0)
{
    $ret = SetSegMetadata($doc_id, "city", "Prague");
    $ret = SavePPMetadata($doc_id, "city");
    $doc_id = GetNextPPDoc($doc_id);
}
```

SaveSegMetadata

**Syntax**

```
SaveSegMetadata(doc_handle, metadata_name);
```

<table>
<thead>
<tr>
<th>doc_handle</th>
<th>Identifies the document.</th>
</tr>
</thead>
<tbody>
<tr>
<td>metadata_name</td>
<td>A string specifying the name of the metadata associated with the specified document.</td>
</tr>
</tbody>
</table>

**Description**

Saves the metadata for the document specified in the `doc_handle` parameter returned by either the `GetFirstSegDoc()`, `GetCurrSegDoc()`, or the `GetNextSegDoc()` function.
Note: If the metadata is not used with enveloping or sorting, the metadata must be declared with the `DeclareMetadata()` function at Before Job level.

## Returns

<table>
<thead>
<tr>
<th>0</th>
<th>The metadata is saved.</th>
</tr>
</thead>
<tbody>
<tr>
<td>num&lt;&gt;0</td>
<td>Error</td>
</tr>
</tbody>
</table>

### Example

```plaintext
$doc_id = GetFirstSegDoc();
while(num($doc_id) > 0)
{
    $ret = SetSegMetadata($doc_id, "city", "Prague");
    $ret = SaveSegMetadata($doc_id, "city");
    $doc_id = GetNextSegDoc($doc_id);
}
```

## SessionEnd

### Syntax

```plaintext
SessionEnd(str_session_name);
```

### Description

Ends a session in the StreamServer, and removes all associated variables.

### Returns

<table>
<thead>
<tr>
<th>1</th>
<th>OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Failed</td>
</tr>
</tbody>
</table>

### Example

Ends a session called `My_Session`:
```plaintext
sessionend("My_Session");
```

## SessionGetVariable

### Syntax

```plaintext
SessionGetVariable(str_session_name, str_var_name);
```

### Description

`str_session_name` A string specifying the session name.
Script functions in alphabetical order

Script functions reference

---

### SessionGetVariable

#### Description
Retrieves the value of a given variable in a given session.

#### Returns
A string containing the variable value.

#### Example
$var_string = sessiongetvariable("My_session", "My_var");

If the variable My_var has the value 12345, the variable var_string will be assigned the string value 12345.

---

#### SessionSetVariable

#### Syntax
SessionSetVariable(str_session_name, str_var_name, str_var_value);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>str_session_name</td>
<td>A string specifying the name of the session.</td>
</tr>
<tr>
<td>str_var_name</td>
<td>A string specifying the variable name.</td>
</tr>
<tr>
<td>str_var_value</td>
<td>A string specifying the variable value.</td>
</tr>
</tbody>
</table>

#### Description
Specifies a value for a given variable in a given session. If the variable does not exist, it will be created.

#### Returns

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OK</td>
</tr>
<tr>
<td>0</td>
<td>Failed</td>
</tr>
</tbody>
</table>

#### Example
Specifies the value 12345 for the variable My_var in a running session called My_Session:

sessionsetvariable("My_Session", "My_var", "12345");
SessionStart

Syntax

SessionStart(str_session_name, num_timeout);

<table>
<thead>
<tr>
<th>str_session_name</th>
<th>A string specifying the name of the session.</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_timeout</td>
<td>A positive integer specifying the timeout length (minutes).</td>
</tr>
</tbody>
</table>

Description

Starts a new session in the StreamServer, and specifies the length of the session (minutes). The session will be removed if it has not been used for the specified number of minutes.

Returns

<table>
<thead>
<tr>
<th>num</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OK</td>
</tr>
<tr>
<td>0</td>
<td>Failed</td>
</tr>
</tbody>
</table>

Example

Starts a session called My_Session, and specifies the timeout to 25 minutes:

sessionstart("My_Session", 25);

SetCopies

Syntax

SetCopies(num);

<table>
<thead>
<tr>
<th>num</th>
<th>A number specifying the number of copies.</th>
</tr>
</thead>
</table>

Description

Sets the number of copies of the current Process.

Note: The setcopies function does only work with PCL drivers. The device must be able to print multiple copies.

Returns

<table>
<thead>
<tr>
<th>num</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OK</td>
</tr>
<tr>
<td>-1</td>
<td>Failed</td>
</tr>
</tbody>
</table>

Example

setcopies(5);
SetCurrMetadata

Syntax

SetCurrMetadata(metadataName, value);

<table>
<thead>
<tr>
<th>metadataName</th>
<th>A string specifying the metadata name for the current document.</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>A string specifying the value of the metadata to set for the current document.</td>
</tr>
</tbody>
</table>

Description

Sets string metadata for the current document. The function can be used at the following post-processing scripting levels:

Before and after document, process and page.

**Note:** If the metadata is not used with enveloping or sorting, the metadata must be declared with the `DeclareMetadata()` function at Before Job level.

Returns

| num<>0 | Error. |

Example

$$ret = SetCurrMetadata("zip", "Prague");$$

SetCurrMetadataNum

Syntax

SetCurrMetadataNum(metadataName, value);

<table>
<thead>
<tr>
<th>metadataName</th>
<th>A string specifying the metadata name for the current document.</th>
</tr>
</thead>
<tbody>
<tr>
<td>num</td>
<td>A number specifying the value of the metadata to set for the current document.</td>
</tr>
</tbody>
</table>

Description

Sets numeric metadata for the current document. The function can be used at the following post-processing scripting levels:

Before and after document, process and page.

**Note:** If the metadata is not used with enveloping or sorting, the metadata must be declared with the `DeclareMetadata()` function at Before Job level.
Returns

<table>
<thead>
<tr>
<th>num</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The metadata value is set.</td>
</tr>
<tr>
<td>num&lt;&gt;0</td>
<td>Error.</td>
</tr>
</tbody>
</table>

Example

```$ret = SetCurrMetadata("invoice_no", 1234);```

**SetCurrX**

**Syntax**

```SetCurrX(num);```

**num**  
A number specifying the new X position in millimeters.

**Description**

Sets a new X position for an object in PageOUT, such as a field, line, barcode, or text.

Once `SetCurrX` has been used to set the x position, you can no longer change the position by using variables in the Positions dialog box in PageOUT.

See also `GetCurrX` and `SetCurrY`.

**Returns**

A number specifying the old X position in millimeters.

**Example**

```$xpos = 10;  
setcurrx($xpos);```

**SetCurrY**

**Syntax**

```SetCurrY(num);```

**num**  
A number specifying the new Y position in millimeters.

**Description**

Sets a new Y position for an object in PageOUT.

`SetCurrY` is only applicable in output frames.

See also `SetCurrX`, `GetCurrBlockY` and `GetCurrObjY`
Returns
A number specifying the old Y position in millimeters.

Example
$ypos = 10;
setcurry($ypos);

SetDestPath

Syntax
SetDestPath(path_1[,path_2...], [str_1 [,str_2...str_N]]);

<table>
<thead>
<tr>
<th>path</th>
<th>One or more paths, separated by semicolon, that replace a % sign in a file specification.</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>One or more strings that replace %1 etc in the file specification string resulting after the % path substitution above.</td>
</tr>
</tbody>
</table>

Note: You can only use % on the output connector.

Description
Sets a destination path, and optionally, a destination file.

⚠️ This function forces the StreamServer application to run in Job mode only, no matter what is set on the connector.

Returns
N/A

Example
//Sets a destination path
setdestpath(\network\printer);
//Creates two files: Test1.pdf and Test2.pdf
setdestpath("Test1.pdf; Test2.pdf");
//Creates two files: adam.pdf and adum.pdf
setdestpath("a%1.pdf; a%2.pdf", "dam", "dum");
SetExtJobId

Syntax
SetExtJobID(str_extjobid);

| str_extjobid | A string containing the external ID to be set for the current job. |

Description
Sets the external ID of the current job to the value of str_extjobid.
This function is usually called from a Before Process script.
See also GetExtJobID, GetIntJobID, GetJobStatus.

Returns

| num>0  | The external ID was set successfully. |
| 0      | Failure. |

Example

//Set the external ID of the current job to ql_invoices
setextjobid ($ql_invoices);

SetFontProperties

Syntax
SetFontProperties(var,"font","weight","angle","underline");

| var | A variable that defines a font. You specify the attributes that is to be changed, and leave the other attributes empty (""). If no variable is available, you can use a dummy variable. The dummy variable does not have to have a value. For this variable, you must specify all attributes. |
| font | The basic font name as specified in the driver file. **Note:** The font name must not include any information about the style of the font, such as weight, angle or underline. |
| weight | Specifies whether the font is bold (True) or regular (False). |
| angle | Specifies whether the font is italic (True) or regular (False). |
| underline | Specifies whether the font is underlined (True) or not underlined (False). |
**Description**

Specifies a font. For example, when using a variable alias for font formatting, you use this scripting function to assign a font to the font variable.

When the script is executed, the specified font (*.ttf) must be available on the system on which the StreamServer is running. To make the font available, you can either create a text with the specified font in PageOUT and export your Project, or you can manually copy the file to the `<Export directory>\data\fonts` directory.

**Returns**

A font name, defined in an internal StreamServe format.

**Example**

```javascript
$font=SetFontProperties($dummy,"Arial","False","False","False");
// Sets $font to Arial, regular weight and angle, not underlined.
if ($country="ENG")
   $fontENG=SetFontProperties($font,"","True","True","");
// Changes the weight and angle of $font.
if ($country="SWE")
   $fontSWE=SetFontProperties($font "","","","True");
// Underlines $font.
```

### SetJobDescr

**Syntax**

```javascript
SetJobDescr(str descr);
```

<table>
<thead>
<tr>
<th>str _descr</th>
<th>The description of the current job.</th>
</tr>
</thead>
</table>

**Description**

Sets the description of the current job. The job description is used in the log and as the title of spooled job (Windows/NetWare).

**Returns**

N/A

**Example**

```javascript
setjobdescr("inv_job");
```

### SetJobFailed

**Syntax**

```javascript
SetJobFailed();
```
Description
Specifies the current job status as failed.

Returns
N/A

Example
setjobfailed();

SetJobOwner

Syntax
SetJobOwner(owner);

<table>
<thead>
<tr>
<th>owner</th>
<th>A string specifying the job owner.</th>
</tr>
</thead>
</table>

Description
Sets the owner of the print job created by the Spool output connector.

The default job owner is the user that sends the job to the StreamServer, for example the file owner if the job is retrieved using a Directory input connector or the authenticated user if using an HTTP input connector.

You also need to specify the keyword SetJobOwner on the output connector.

Note: This function does not affect the job metadata in the queue database. To change the metadata in the queue you must specify a job sender in Design Center.

See also CurrJobOwner.

Returns
N/A

Example
setjobowner("Susan");

SetLanguage

Syntax
SetLanguage(lang);

<table>
<thead>
<tr>
<th>lang</th>
<th>A string specifying a language.</th>
</tr>
</thead>
</table>

Description
Sets the language to be used for the current job. This setting is used with the StreamServe Language Set (*.sls) file.
Note: Not supported in Post-processor scripting.

Returns
N/A

Example
if($language="SWE")
{
    setlanguage($language);
}

SetMetaDataJob

Syntax
SetMetaDataJob(key, value);

<table>
<thead>
<tr>
<th>key</th>
<th>The name of the metadata attribute.</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>The value to which to set the metadata attribute for this job.</td>
</tr>
</tbody>
</table>

Description
Creates a new metadata attribute for the input job.

Returns

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The metadata attribute was successfully added.</td>
</tr>
<tr>
<td>-1</td>
<td>An error occurred.</td>
</tr>
</tbody>
</table>

Example
$result=SetMetaDataJob("type", "print");

SetPPMetadata

Syntax
SetPPMetadata(doc_handle, metadata_name, value);

<table>
<thead>
<tr>
<th>doc_handle</th>
<th>Identifies a document</th>
</tr>
</thead>
<tbody>
<tr>
<td>metadata_name</td>
<td>A string specifying the metadata name associated with the current document.</td>
</tr>
<tr>
<td>value</td>
<td>A string specifying the value of the metadata associated with the current document.</td>
</tr>
</tbody>
</table>
Description
Sets metadata of string type for the document specified in the `doc_handle` parameter returned by either the `GetFirstPPDoc()` or the `GetNextPPDoc()` function.

**Note:** If the metadata is not used with enveloping or sorting, the metadata must be declared with the `DeclareMetadata()` function at Before Job level.

Returns

<table>
<thead>
<tr>
<th>num</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The metadata value is set.</td>
</tr>
<tr>
<td>num&lt;&gt;0</td>
<td>Error.</td>
</tr>
</tbody>
</table>

Example

```bash
$doc_id = GetFirstPPDoc();
while(num($doc_id) > 0)
{
    $ret = SetPPMetaData ($doc_id, "city", "Prague");
    $doc_id = GetNextPPDoc($doc_id);
}
```

SetPPMetadataNum

**Syntax**

`SetPPMetadataNum(doc_handle, metadataName, value);`

<table>
<thead>
<tr>
<th><code>doc_handle</code></th>
<th>Identifies a document</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>metadata_name</code></td>
<td>A string specifying the metadata name associated with the current document.</td>
</tr>
<tr>
<td><code>value</code></td>
<td>A number specifying the value of the metadata associated with the current document.</td>
</tr>
</tbody>
</table>

**Description**

Sets metadata of numeric type for the document specified in the `doc_handle` parameter returned by either the `GetFirstPPDoc()` or the `GetNextPPDoc()` function.

**Note:** If the metadata is not used with enveloping or sorting, the metadata must be declared with the `DeclareMetadata()` function at Before Job level.
Returns

<table>
<thead>
<tr>
<th>num</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The metadata value is set.</td>
</tr>
<tr>
<td>num&lt;0</td>
<td>Error</td>
</tr>
</tbody>
</table>

Example

```perl
$doc_id = GetFirstPPDoc();
while(num($doc_id) > 0)
{
    $ret = SetPPMetaDataNum ($doc_id, "invoice_no", 112233);
    $doc_id = GetNextPPDoc($doc_id);
}
```

SetPrnXoffs

**Syntax**

```perl
SetPrnXoffs(num);
```

**num**  
A number specifying an adjustment in millimeters.

**Description**

Adjusts X positions for overlay files (*.prn). Adjustments are valid until further adjustments.

**Note:** The overlay must have the Offset Adjust check box selected in the PageOUT Process.

**Returns**

A number specifying the old X adjustment.

**Example**

```perl
if($doctype="invoice")
{
    setprnxoffs(4);
}
```

SetPrnYoffs

**Syntax**

```perl
SetPrnYoffs(num);
```

**num**  
A number specifying an adjustment in millimeters.
Description
Adjusts Y positions for overlay files (*.prn). Adjustments are valid until further adjustments.

Note: The overlay must have the Offset Adjust check box selected in the PageOUT Process.

Returns
A number specifying the old Y adjustment.

Example
if($doctype="picklist")
{
    setprnyoffs(-3);
}

SetSegMetadata

Syntax
SetSegMetadata(doc_handle, metadataName, value);

doc_handle | Identifies a document
metadataName | A string specifying the metadata name associated with the document.
value | A string specifying the value of the metadata for the specified document.

Description
Sets string metadata for the document specified in the doc_handle parameter returned by either the GetFirstSegDoc(), GetCurrSegDoc(), or the GetNextSegDoc() function.

Note: If the metadata is not used with enveloping or sorting, the metadata must be declared with the DeclareMetadata() function at Before Job level.

Returns

| 0 | Metadata is set. |
| num<0 | Error |

Example
$doc_id = GetFirstSegDoc();
while(num($doc_id) > 0)
{
    $ret = SetSegMetadata($doc_id, "city", "Prague");
}
SetSegMetadataNum

**Syntax**

\[
\text{SetSegMetadataNum(doc\_handle, metadataName, value)};
\]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc_handle</td>
<td>Identifies a document</td>
</tr>
<tr>
<td>metadataName</td>
<td>A string specifying the metadata name associated with the document.</td>
</tr>
<tr>
<td>value</td>
<td>A number specifying the value of the metadata for the specified document.</td>
</tr>
</tbody>
</table>

**Description**

Sets numeric metadata for the document specified in the doc\_handle parameter returned by either the GetFirstSegDoc(), GetCurrSegDoc(), or the GetNextSegDoc() function.

**Note:** If the metadata is not used with enveloping or sorting, the metadata must be declared with the DeclareMetadata() function at Before Job level.

**Returns**

<table>
<thead>
<tr>
<th>num</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Metadata is set.</td>
</tr>
<tr>
<td>num&lt;&gt;0</td>
<td>Error</td>
</tr>
</tbody>
</table>

**Example**

```plaintext
$doc\_id = GetFirstSegDoc();
while(num($doc\_id) > 0)
{
    $ret = SetSegMetadataNum($doc\_id, "invoice\_no", 1234);
    $doc\_id = GetNextSegDoc($doc\_id);
}
```

**SetSerNo**

**Syntax**

\[
\text{SetSerNo(filename, str\_serial\_id, num\_value)};
\]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>A string specifying the path and filename of the file containing the number sequence.</td>
</tr>
</tbody>
</table>
SetSerNo

Syntax

SetSerNo("serno.txt","ID1",100);

Description

Sets the value of a serial number in a number sequence. See `GetSerNo` for additional information.

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OK</td>
</tr>
<tr>
<td>-1</td>
<td>Failed</td>
</tr>
</tbody>
</table>

Example

setserno("serno.txt","ID1",100);

SetSubst

Syntax

SetSubst(tbl_file, str_key, num_col, str_value);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tbl_file</td>
<td>A string specifying the path and filename of the substitution table.</td>
</tr>
<tr>
<td>str_key</td>
<td>The key of the entry to search for.</td>
</tr>
<tr>
<td>num_col</td>
<td>A number specifying the value column. The first value column number is 0.</td>
</tr>
<tr>
<td>str_value</td>
<td>A string representing a value to assign.</td>
</tr>
</tbody>
</table>

Description

Assigns `str_value` to the value column `num_col` of the specified substitution table entry. It does not affect the physical file stored on disk, only the representation of the table in the memory.

The substitution table file is specified by `tbl_file`, and the substitution table entry is specified by `str_key`. If the entry does not exist, it will be created.

Returns

N/A
Example
In this example, the following substitution table (author.tbl) is used.

```plaintext
//!CodePage UTF8!
//!multicolumn!
agatha christie 1890
alfred h 1899
```

The following script modifies the substitution table entry alfred, and adds the substitution table entry astrid:

```plaintext
SetSubst("../data/tables/author.tbl","alfred",0,"hitchcock");
SetSubst("../data/tables/author.tbl","astrid",0,"lindgren");
SetSubst("../data/tables/author.tbl","astrid",1,"1907");
WriteSubst("../data/tables/author.tbl");
```

After the script is executed, author.tbl is changed as follows:

```plaintext
//!CodePage UTF8!
//!multicolumn!
agatha christie 1890
alfred hitchcock 1899
astrid lindgren 1907
```

Notes
- A substitution table is kept in memory until the script function EraseSubst is called for that table.
- Try to use the AddSubst function instead of SetSubst when possible, as more than one job can be executed at the same time. If different jobs call SetSubst, they will overwrite each others’ modifications to the substitution table.

SetXoffs

Syntax

```
SetXoffs(num);
```

<table>
<thead>
<tr>
<th>num</th>
<th>A number specifying an adjustment in millimeters.</th>
</tr>
</thead>
</table>

Description

Adjusts X positions for all output fields, pictures, and overlays towards the right. The adjustment is valid within the current job only, or until another adjustment is specified.

**Note:** The overlay must have the Offset Adjust check box selected in the PageOUT Process.
Returns
A number specifying the old X adjustment.

Example
if($doctype="invoice")
{
    setxoffs(4);
}

SetYoffs

Syntax
SetYoffs(num);

<table>
<thead>
<tr>
<th>num</th>
<th>A number specifying an adjustment in millimeters.</th>
</tr>
</thead>
</table>

Description
Adjusts Y positions for all output fields, pictures, and overlays downwards. The adjustment is valid within the current job only, or until another adjustment is specified.

Note: The overlay must have the Offset Adjust check box selected in the PageOUT Process.

Returns
A number specifying the old Y adjustment.

Example
if($doctype="pickinglist")
{
    setyoffs(-8);
}

SinCount
Same as $MsgCountId$.

SinVal
Same as $MsgGetValue$.

SinValExist
Same as $MsgValueExist$. 
Skip

Syntax
Skip();

Description
Skips the current operation, i.e. the operation to which the script is associated. It can also be used to skip Events, Processes, blocks and fields.

The function is useful for more complicated tests than a rule allows for. For example, the function can be used for suppressing page numbers on one-page printouts.

Note: The script that contains the `skip()` function is fully executed including functions invoked after the `skip()` call.

Limitations
- For Retrieved scripts, the function can only be used in Event Retrieved scripts.
- The function is not supported in Post-processor scripting.
- You can not use this function to skip Free Blocks. To conditionally render Free Blocks, you can e.g. create a script After Block that uses `CallBlock("FreeBlock").`
- Pages can not be skipped after a frame overflow.

Returns

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OK</td>
</tr>
<tr>
<td>-1</td>
<td>Failed</td>
</tr>
</tbody>
</table>

Example

```java
if(num($count)>15)
{
    skip();
}
```

Sort

Syntax
Sort(sortkey_id, value_1[, value_2,..value_n]);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sortkey_id</td>
<td>A string containing the key identifier.</td>
</tr>
<tr>
<td>value_1,..value_n</td>
<td>Strings specifying the key values.</td>
</tr>
</tbody>
</table>
Each sort key requires a type definition line in a file named `sortdef` that defines the types (`str` or `num`) of the sort keys.

Syntax of type definition line in `sortdef` file:

```
sortdef "key identifier" type_1 [type_2 ...]
```

**Description**

Used in a Before Process script to assign one or more sort keys to a Message. StreamServe sorts the Messages in the job according to the specified sort keys.

**Note:** The `Sort` function requires a `sortdef` file that contains lists of data types. You must include the name of this file, `sortdef`, in the StreamServer startup argument file (`Start.arg`). The function is not supported in Post-processor scripting.

**Returns**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OK</td>
</tr>
<tr>
<td>-1</td>
<td>Failed</td>
</tr>
</tbody>
</table>

**Example**

```
sort("inv", $invno);
```

**SortSegDoc**

**Syntax**

```
SortSegDoc(value_1[, value_2,..value_n]);
```

<table>
<thead>
<tr>
<th>value_1,..</th>
<th>Strings specifying the metadata keys used for sorting documents.</th>
</tr>
</thead>
<tbody>
<tr>
<td>value_N</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

Sorts the documents in the current segment according to the specified sort keys.

The sorting order is defined by one of the following

- the `DeclareMetadata()` function
- the sort key when sorting documents
- the envelope key when creating an envelope definition.

When the `SortSegDoc()` function is used with an envelope key the `EnvelNr` metadata key should be part of the sort otherwise the envelope definition will not be retained.

The function can be used at the following post-processing scripting levels:

Before job.
Returns

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Documents are sorted.</td>
</tr>
<tr>
<td>num&lt;0</td>
<td>Error</td>
</tr>
</tbody>
</table>

Example

Job Begin script:
$ret = DeclareMetadata("zip", "N", "A");
Post-processor Job Begin script:
$doc_id = GetFirstSegDoc();
while(num($doc_id) > 0)
{
    $zip = 12345
    SetSegMetadata($doc_id, "zip", $zip);
    $doc_id = GetNextSegDoc($doc_id);
}
$ret = SortSegDoc("zip", "EnvelNr");

StartTimer

Syntax

StartTimer(int id);

| int id | The ID of the timer. Each job can have a maximum of 10 timers (0-9). |

Description

Starts a timer in the StreamServer to measure processing time. Use the StopTimer script function to stop the timer, see StopTimer on page 330.

Example

StartTimer (0)

StEvalXPath

Syntax

StEvalXPath(strXPath);

| strXPath | The XPath expression string to evaluate. |
Description
Evaluates an XPath expression.

Returns

| str | A string. If evaluation fails an empty string is returned. |

Example
$EvaluatedXPath = StEvalXPath("/data/message/b1/f2[position() <3]");

StGetParagraphPageXmm

Syntax
StGetParagraphPageXmm();

Description
StoryTeller tool function. Returns the X-coordinate in millimeters for the upper left corner of an object’s paragraph, calculated from the page origin.

Returns

<table>
<thead>
<tr>
<th>num</th>
<th>The X coordinate in millimeters.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The function was called in a bad context.</td>
</tr>
</tbody>
</table>

Example
$xcoord = StGetParagraphPageXmm();

StGetParagraphPageXpt

Syntax
StGetParagraphPageXpt();

Description
Returns the X-coordinate in points for the upper left corner of an object’s paragraph, calculated from the page origin.
Returns

<table>
<thead>
<tr>
<th>num</th>
<th>The X coordinate in points.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The function was called in a bad context.</td>
</tr>
</tbody>
</table>

Example

$xcoord = StGetParagraphPageXPt();

StGetParagraphPageYMM

Syntax

StGetParagraphPageYMM();

Description

Returns the Y-coordinate in millimeters for the upper left corner of an object’s paragraph, calculated from the page origin.

Returns

<table>
<thead>
<tr>
<th>num</th>
<th>The Y coordinate in millimeters.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The function was called in a bad context.</td>
</tr>
</tbody>
</table>

Example

$ycoord = StGetParagraphPageYMM();

StGetParagraphPageYPT

Syntax

StGetParagraphPageYPT();

Description

Returns the Y-coordinate in points for the upper left corner of an object’s paragraph, calculated from the page origin.

Returns

<table>
<thead>
<tr>
<th>num</th>
<th>The Y coordinate in points.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The function was called in a bad context.</td>
</tr>
</tbody>
</table>

Example

$ycoord = StGetParagraphPageYPT();
StGetParagraphXMMm

**Syntax**

StGetParagraphXMMm();

**Description**

Returns the X-coordinate in millimeters for the upper left corner of an object’s paragraph, relative to the text area origin.

**Returns**

<table>
<thead>
<tr>
<th>num</th>
<th>The X coordinate in millimeters.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The function was called in a bad context.</td>
</tr>
</tbody>
</table>

**Example**

$xcoord = StGetParagraphXMMm();$

StGetParagraphXPt

**Syntax**

StGetParagraphXPt();

**Description**

Returns the X-coordinate in points for the upper left corner of an object’s paragraph, relative to the text area origin.

**Returns**

<table>
<thead>
<tr>
<th>num</th>
<th>The X coordinate in points.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The function was called in a bad context.</td>
</tr>
</tbody>
</table>

**Example**

$xcoord = StGetParagraphXPt();$

StGetParagraphYMMm

**Syntax**

StGetParagraphYMMm();

**Description**

Returns the Y-coordinate in millimeters for the upper left corner of an object’s paragraph, relative to the text area origin.
Script functions in alphabetical order

Returns

<table>
<thead>
<tr>
<th>num</th>
<th>The Y coordinate in millimeters.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The function was called in a bad context.</td>
</tr>
</tbody>
</table>

Example

```javascript
$ycoord = StGetParagraphYMm();
```

StGetParagraphYPt

**Syntax**

```javascript
StGetParagraphYPt();
```

**Description**

Returns the Y-coordinate in points for the upper left corner of an object’s paragraph, relative to the text area origin.

Returns

<table>
<thead>
<tr>
<th>num</th>
<th>The Y coordinate in points.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The function was called in a bad context.</td>
</tr>
</tbody>
</table>

Example

```javascript
$ycoord = StGetParagraphYPt();
```

StGetProcessingProperty

**Syntax**

```javascript
StGetProcessingProperty(property);
```

**property** The StoryTeller processing property to retrieve the value for. For available properties, see the table below:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>language</td>
<td>The current language used during runtime.</td>
</tr>
</tbody>
</table>

**Note:** Property keys are case sensitive!

**Note:** To get the total number of pages in a document, use *Pages* on page 288.
**Description**
Retrieves the value of the specified non-processing property in StoryTeller.

**Returns**

| string | The value of the specified property. For invalid properties an empty string is returned. |

**Example**

```javascript
$locale = StGetProcessingProperty("language");
```

**StGetProperty**

**Syntax**

```javascript
StGetProperty(property);
```

<table>
<thead>
<tr>
<th>property</th>
<th>The non-dimensional StoryTeller property to retrieve the value for. For available properties, see the table below:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Note:</strong> Property keys are case sensitive!</td>
<td></td>
</tr>
<tr>
<td><strong>Note:</strong> To retrieve dimension properties, use <code>StGetPropertyMm</code> on page 324 or <code>StGetPropertyPt</code> on page 326.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The object name</td>
</tr>
<tr>
<td>Description</td>
<td>The object description</td>
</tr>
<tr>
<td>Visibility</td>
<td>The visibility of the object</td>
</tr>
<tr>
<td></td>
<td>0 – Normal</td>
</tr>
<tr>
<td></td>
<td>1 – Hidden (Object is suppressed)</td>
</tr>
<tr>
<td></td>
<td>2 – Invisible</td>
</tr>
<tr>
<td>Rotation</td>
<td>Clockwise rotation in degrees.</td>
</tr>
<tr>
<td>SizeMode</td>
<td>For images:</td>
</tr>
<tr>
<td></td>
<td>0 – Resize</td>
</tr>
<tr>
<td></td>
<td>1 – Original</td>
</tr>
</tbody>
</table>
| **Positioning** | For images:
| | 0 – Left top
| | 1 – Left center
| | 2 – Left bottom
| | 3 – Center top
| | 4 – Center center
| | 5 – Center bottom
| | 6 – Right top
| | 7 – Right center
| | 8 – Right bottom |

| **Alignment** | For images when SizeMode is set to Resize and ScaleMode is None or Proportional, this specifies the alignment of the image within the bounding box (or just refer to ST doc)
| | 0 – Left top
| | 1 – Left center
| | 2 – Left bottom
| | 3 – Center top
| | 4 – Center center
| | 5 – Center bottom
| | 6 – Right top
| | 7 – Right center
| | 8 – Right bottom |

| **Visibility** | This property represents the Visibility property in the Properties panel.
| | 0 – Visible
| | 1 – Hidden
| | 2 – Invisible |

| **TextHeightGrow** | This property represents the Vertical size property in the Properties panel.
| | 0 – Fixed
| | 1 – Limited

**Note:** Use the `TextHeightMin/TextHeightMax` property keys with `StSetPropertyMm()` or `StSetPropertyPt()` to set the limits. To use "no limit" for the text height, specify a very high number.
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LineColor</td>
<td>The line color. See StoryTeller documentation for supported formats.</td>
</tr>
<tr>
<td>FillType</td>
<td>This property represents the Fill &gt; Type property in Properties panel.</td>
</tr>
<tr>
<td></td>
<td>0 – Solid</td>
</tr>
<tr>
<td></td>
<td>1 – Linear gradient horizontal</td>
</tr>
<tr>
<td></td>
<td>2 – Linear gradient vertical</td>
</tr>
<tr>
<td></td>
<td>3 – Radial gradient</td>
</tr>
<tr>
<td>FillColor</td>
<td>The fill color. See StoryTeller documentation for supported formats.</td>
</tr>
<tr>
<td>FillSecondColor</td>
<td>The second fill color for gradient colors.</td>
</tr>
<tr>
<td>ScaleMode</td>
<td>0 – None</td>
</tr>
<tr>
<td></td>
<td>1 – Scale</td>
</tr>
<tr>
<td></td>
<td>2 – Proportional</td>
</tr>
<tr>
<td>ImageURI</td>
<td>The URI to the image.</td>
</tr>
<tr>
<td>ImageFormat</td>
<td>The image format. Read-only.</td>
</tr>
<tr>
<td>ImageBitDepth</td>
<td>The image bit depth. Read-only.</td>
</tr>
<tr>
<td>ImagePixelWidth</td>
<td>The image pixel width. Read-only.</td>
</tr>
<tr>
<td>ImagePixelHeight</td>
<td>The image pixel height. Read-only.</td>
</tr>
<tr>
<td>ImageResolutionX</td>
<td>The horizontal image resolution.</td>
</tr>
<tr>
<td>ImageResolutionY</td>
<td>The vertical image resolution.</td>
</tr>
<tr>
<td>ImagePage</td>
<td>The number of the page for a multi-page TIFF image.</td>
</tr>
<tr>
<td>OverlayURI</td>
<td>The URI to the overlay</td>
</tr>
<tr>
<td>OverlayPage</td>
<td>The number of the page for a multi-page overlay.</td>
</tr>
<tr>
<td>BarcodeName</td>
<td>The barcode type.</td>
</tr>
<tr>
<td>BarcodeData</td>
<td>The barcode data.</td>
</tr>
<tr>
<td>SubstitutionURI</td>
<td>The URI to the substitution</td>
</tr>
<tr>
<td>Cells[n]/FillColor</td>
<td>The fill color of the nth cell on the row, where n=0 is the left cell.</td>
</tr>
<tr>
<td>Cells[]/LineColor</td>
<td>The border color of the nth cell on the row, where n=0 is the left cell.</td>
</tr>
<tr>
<td>Cells[]/TopLineColor</td>
<td>The border color of the top border of the nth cell on the row, where n=0 is the left cell.</td>
</tr>
</tbody>
</table>
Script functions reference

**StGetPropertyMm**

**Syntax**

```plaintext
StGetPropertyMm(property);
```

<table>
<thead>
<tr>
<th>property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>the dimensional property to retrieve the value for. For available properties, see the table below.</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** Property keys are case sensitive!

**Note:** To retrieve a non-dimensional property, use `StGetProperty` on page 321.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>The horizontal position of the object’s anchor point relative to the origin (upper left corner) of the parent object, normally the page.</td>
</tr>
<tr>
<td><strong>Y</strong></td>
<td>The vertical position of the object’s anchor point relative to the origin (upper left corner) of the parent object, normally the page.</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Width</strong></td>
<td>The width of the object’s bounding box.</td>
</tr>
<tr>
<td><strong>Height</strong></td>
<td>The height of the object’s bounding box, or the height of a table row.</td>
</tr>
<tr>
<td><strong>TextHeightMin</strong></td>
<td>If Vertical size (or TextHeightGrow property if using scripting) is set to Limited, this is the minimum height of the text regardless of the height of the text area.</td>
</tr>
<tr>
<td><strong>TextHeightMax</strong></td>
<td>If Vertical size (or TextHeightGrow property if using scripting) is set to Limited, this is the maximum height of the text regardless of the height of the text area.</td>
</tr>
<tr>
<td><strong>ContentVShrink</strong></td>
<td>The maximum vertical shrink of the content within a shape.</td>
</tr>
<tr>
<td><strong>ContentVGrowth</strong></td>
<td>The maximum vertical growth of the content within a shape.</td>
</tr>
<tr>
<td><strong>ContentHShrink</strong></td>
<td>The maximum horizontal shrink of the content within a shape.</td>
</tr>
<tr>
<td><strong>ContentHGrowth</strong></td>
<td>The maximum horizontal growth of the content within a shape.</td>
</tr>
<tr>
<td><strong>ShapeVShrink</strong></td>
<td>The maximum vertical shrink of the shape.</td>
</tr>
<tr>
<td><strong>ShapeVGrowth</strong></td>
<td>The maximum vertical growth of the shape.</td>
</tr>
<tr>
<td><strong>ShapeHShrink</strong></td>
<td>The maximum horizontal shrink of the shape.</td>
</tr>
<tr>
<td><strong>ShapeHGrowth</strong></td>
<td>The maximum horizontal growth of the shape.</td>
</tr>
<tr>
<td><strong>LineThickness</strong></td>
<td>The line thickness.</td>
</tr>
<tr>
<td><strong>Columns[]/Width</strong></td>
<td>The column width of the ( n )th column of the table, where ( n=0 ) is the left column.</td>
</tr>
<tr>
<td><strong>RowHeight</strong></td>
<td>The row height.</td>
</tr>
<tr>
<td><strong>Cells[]/LineThickness</strong></td>
<td>The line thickness of the ( n )th cell on the row, where ( n=0 ) is the left cell.</td>
</tr>
<tr>
<td><strong>Cells[]/TopLineThickness</strong></td>
<td>The line thickness of the top border of the ( n )th cell on the row, where ( n=0 ) is the left cell.</td>
</tr>
<tr>
<td><strong>Cells[]/LeftLineThickness</strong></td>
<td>The line thickness of the left border of the ( n )th cell on the row, where ( n=0 ) is the left cell.</td>
</tr>
</tbody>
</table>

### Cells[]/RightLineThickness

The line thickness of the right border of the \( n \)th cell on the row, where \( n=0 \) is the left cell.

### Cells[]/BottomLineThickness

The line thickness of the bottom border of the \( n \)th cell on the row, where \( n=0 \) is the left cell.

#### Description

Retrieves the value of the specified property in StoryTeller. The value is returned in millimeters.

#### Returns

<table>
<thead>
<tr>
<th>num</th>
<th>The value of the specified property in millimeters. For invalid properties 0 is returned.</th>
</tr>
</thead>
</table>

#### Example

```javascript
$Height = StGetPropertyMm("Height");
```

### StGetPropertyPt

#### Syntax

```
StGetPropertyPt(property);
```

<table>
<thead>
<tr>
<th>property</th>
<th>The dimension property to retrieve the value for, in points. See \textit{StGetPropertyMm} on page 324 for available properties.</th>
</tr>
</thead>
</table>

**Note:** Property keys are case sensitive!

**Note:** To retrieve a non-dimensional property, use \textit{StGetProperty} on page 321.

#### Description

Retrieves the value of the specified dimension property in StoryTeller. The value is returned in points.

#### Returns

<table>
<thead>
<tr>
<th>num</th>
<th>The value of the property in points. For invalid properties 0 is returned.</th>
</tr>
</thead>
</table>

#### Example

```javascript
$height = StGetPropertyPt("Height");
```
StIsEmpty

Syntax
StIsEmpty();

Description
StoryTeller tool function. It checks whether an object is empty or not. The meaning of empty depends on the object, as follows:

<table>
<thead>
<tr>
<th>Object</th>
<th>...is empty if...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Composition Center Section</td>
<td>No fragment is returned for the section.</td>
</tr>
<tr>
<td>Substitution, Text object, Story frame</td>
<td>It contains no text, commands, or layout objects, and all content objects are empty.</td>
</tr>
<tr>
<td>Table row</td>
<td>No cell in the row contains text, commands, or layout objects, and all content objects are empty.</td>
</tr>
<tr>
<td>Table</td>
<td>Table has no body rows, or all body rows are empty.</td>
</tr>
<tr>
<td>Page</td>
<td>No Story frames on the page contains text, commands, or layout objects, and all content objects are empty.</td>
</tr>
</tbody>
</table>

Returns

bool

Returns true if the content of an object is empty, for example an empty substitution.

Example

$boolean = StIsEmpty();

StLanguageLookup

Syntax
StLanguageLookup(language, key);

<table>
<thead>
<tr>
<th>language</th>
<th>The language to translate into</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>The key to identify the entry in the language file.</td>
</tr>
</tbody>
</table>
**Description**  
Retrieves a translation from the available .sls files, using specified language and key. Useful if you e.g. want to make a decision based on how a substitution is translated.

**Example**  
```php  
$value = stLanguageLookup(stGetProcessingProperty("language"), &data_field);  
if($value==")  
{  
   $value=&data_field;  
}  
```

**Note:** This function call generates a log message when value is not found for the key.

---

**StMsgAddNode**

**Syntax**  
```php
StMsgAddNode(strXPath, name);
```

<table>
<thead>
<tr>
<th><strong>strXPath</strong></th>
<th>The position in the tree where the node is appended</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>name</strong></td>
<td>The name of the node to append.</td>
</tr>
</tbody>
</table>

**Description**  
StoryTeller tool function.  
Appends a node `name` to a Message. The position in the Message tree where the `name` node is placed is specified by `strXPath`.

**Returns**  
N/A

**Example**  
```php
StMsgAddNode("/", "/New_node");
```

---

**StMsgAttachXML**

**Syntax**  
```php
StMsgAttachXML(strXPath, xml, src_XPath);
```
### StMsgAttachXML

**Syntax**

```plaintext
StMsgAttachXML(strXPath, xml, src.XPath);
```

<table>
<thead>
<tr>
<th><strong>strXPath</strong></th>
<th>XPath to a Message node where a DOM node is appended.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>xml</strong></td>
<td>An XML string from which an XML DOM node is loaded.</td>
</tr>
<tr>
<td><strong>src.XPath</strong></td>
<td>DOM node that is appended to a Message node</td>
</tr>
</tbody>
</table>

**Description**

StoryTeller tool function.

Appends an XML DOM node – loaded from an XML string – to a Message node.

**Returns**

N/A

**Example**

```plaintext
StMsgAttachXML("/","<MyString>", "/Resources");
```

### StMsgAttachXMLFile

**Syntax**

```plaintext
StMsgAttachXMLFile(strXPath, xml_file, src.XPath);
```

<table>
<thead>
<tr>
<th><strong>strXPath</strong></th>
<th>XPath to a Message node where a DOM node is appended.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>xml_file</strong></td>
<td>An XML file from which an XML DOM node is loaded.</td>
</tr>
<tr>
<td><strong>src.XPath</strong></td>
<td>DOM node that is appended to a Message node</td>
</tr>
</tbody>
</table>

**Description**

StoryTeller tool function.

Appends an XML DOM node – loaded from an XML file – to a Message node.

**Returns**

N/A

**Example**

```plaintext
StMsgAttachXMLFile("/", ".\resources.xml", "/Resources");
```
StMsgSaveToFile

Syntax

StMsgSaveToFile(xml_file);

| xml_file          | The XML file to drop Message data into for an Event. |

Description

StoryTeller tool function, mainly to be used for debugging. The function saves Message data for an Event into an XML file.

Returns

N/A

Example

StMsgSaveToFile("../file_name.xml");

StopTimer

Syntax

StopTimer(int id, int logIt);

<table>
<thead>
<tr>
<th>int id</th>
<th>The ID of the timer you want to stop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>int logIt</td>
<td>0 – Do not log elapsed time.</td>
</tr>
<tr>
<td></td>
<td>1 – Log elapsed time.</td>
</tr>
<tr>
<td></td>
<td>2 – Log elapsed time and write to file.</td>
</tr>
</tbody>
</table>

Description

Stops the specified timer in the StreamServer, and if required displays the elapsed time in the log and/or writes it to a file.

For information about starting a timer, see StartTimer on page 316.

Example

StopTimer(0, 1);
StSetProperty

Syntax

StSetProperty(property, value);

<table>
<thead>
<tr>
<th>property</th>
<th>The non-dimensional property to set the value for. See StGetProperty on page 321 for available properties.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Note:</strong> Property keys are case sensitive!</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> To set dimension properties, use StSetPropertyMm on page 331 or StSetPropertyPt on page 332.</td>
</tr>
<tr>
<td>value</td>
<td>The value to set on the specified property.</td>
</tr>
</tbody>
</table>

Description

Sets a value on a non-dimensional property in StoryTeller.

Returns

| 0       | Success. For invalid properties -1 is returned.                                                          |

Example

$visibility = StSetProperty("Visibility", "1");

StSetPropertyMm

Syntax

StSetPropertyMm(property, value);

<table>
<thead>
<tr>
<th>property</th>
<th>The dimension property to set the value for. See StGetPropertyMm on page 324 for available properties.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Note:</strong> Property keys are case sensitive!</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> To set non-dimension properties, use StSetProperty on page 331.</td>
</tr>
<tr>
<td>value</td>
<td>The value in millimeters to set on the specified property.</td>
</tr>
</tbody>
</table>

Description

Sets the specified value in millimeters on the specified dimension property in StoryTeller.
### StSetPropertyMm

**Syntax**

```
StSetPropertyMm(property, value);
```

**Description**

Sets a specified value in millimeters for the specified dimension property in StoryTeller.

**Returns**

| num | The value set in millimeters. For invalid properties -1 is returned. |

**Example**

```
$new_height = StSetPropertyMm("Height", "10");
```

### StSetPropertyPt

**Syntax**

```
StSetPropertyPt(property, value);
```

<table>
<thead>
<tr>
<th>property</th>
<th>The dimension property to set the value for. See <code>StGetPropertyMm</code> on page 324 for available properties.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Note: Property keys are case sensitive!</td>
<td></td>
</tr>
<tr>
<td>Note: To set non-dimension properties, use <code>StopTimer(0, 1)</code> on page 330.</td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>The value in points unit to set on the specified property.</td>
</tr>
</tbody>
</table>

**Description**

Sets a specified value in points for the specified dimension property in StoryTeller.

**Returns**

| num | The value set in points. For invalid properties -1 is returned. |

**Example**

```
$box_width = StSetPropertyPt("Width", "100");
```

### StoreJobIDAttribute

`StoreJobIDAttribute` is an old function and is removed.

### StoreJobIDDateAttribute

`StoreJobIDDateAttribute` is an old function and is removed.

### StoreJobIDNumAttribute

`StoreJobIDNumAttribute` is an old function and is removed.
Str

Syntax
Str(num);

| num | A number. |

Description
Converts a numerical value to a string.

Returns
A string.

Example
str(10);
Result:
"10"

StrBlk

Syntax
StrBlk(str);

| str | A string to scan. |

Description
Removes leading white space (spaces and tabs) in a string.

Returns
A copy of the string with leading white space removed.

Example
$a=strblk(" In the beginning...");
Result:
$a="In the beginning..."

StridX

Syntax
StridX(str2scan, search_str);

| str2scan | The string to scan. |
| search_str | The string to search for. |
Script functions in alphabetical order

Script functions reference

Description
Locates the first occurrence of a sub-string in a string.

Returns

| num | If search_str was found in str2scan, a number indicating the position of the first character of the first occurrence of search_str is returned. The first character position in the string is 1. If search_str was not found in str2scan, 0 is returned. If search_str is an empty string, 1 is returned. |

Example
$a="Intelligent Output Management";
$b=stridx($a,"tell");
Result:
$b="3"

StrLadj

Syntax
StrLadj(str, num_strlen);

<table>
<thead>
<tr>
<th>str</th>
<th>A string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_strlen</td>
<td>Required length of string.</td>
</tr>
</tbody>
</table>

Description
Left-aligns and truncates (or extends) a string to a specified length. If the specified length is greater than the current string length, the string is extended with spaces.

Returns
Left-aligned and truncated (or extended) string according to the specified required length.

Example
$a="Parrot";
$b=strladj($a,10);
Result:
$b="Parrot   "

StrLen

**Syntax**

```
StrLen(str);
```

| str  | A string. |

**Description**

Returns the length of a string.

**Returns**

| num  | A number indicating the length of the input string. |

**Example**

```php
$a=strlen("hello");
```

Result:

```
$a=5
```

StrQTok

**Syntax**

```
StrQTok(str, char_sep, quote_char, esc_char, result_array);
```

<table>
<thead>
<tr>
<th>str</th>
<th>The string to be tokenized.</th>
</tr>
</thead>
<tbody>
<tr>
<td>char_sep</td>
<td>A character separator.</td>
</tr>
<tr>
<td>quote_char</td>
<td>A string quote character.</td>
</tr>
<tr>
<td>esc_char</td>
<td>A quote escape character.</td>
</tr>
<tr>
<td>result_array</td>
<td>The array to which the tokens will be assigned.</td>
</tr>
</tbody>
</table>

**Description**

Takes four strings and a one-dimensional array and returns a numeric value indicating the number of tokens passed into the array.

**Returns**

A numeric value indicating the number of tokens passed into the array.

**Example**

```php
$var1="123:abc:'A, B and '/'C'/'";
```

//Single quotes are used for strings containing separators.
// The escape character for single quotes
// in quoted strings is "/".
$num_of_tokens=StrQTok($var1, ",", ",", "/", $arr1);

Result:
$num_of_tokens=3
$arr1[0]="123"
$arr1[1]="abc"

StrrBlk

Syntax
StrrBlk(str);

| str | A string. |

Description
Removes trailing white spaces (spaces and tabs) from the input string.

Returns
A copy of the input string with all trailing white spaces removed.

Example
$a=strrblk("At length!      ");

Result:
$a="At length!"

StrTok

Syntax
StrTok(str, char_sep, result_array);

| str | The string to be tokenized. |
| char_sep | A character separator. |
| result_array | The array to which the tokens will be assigned. |

Description
Takes two strings and a one-dimensional array and returns a numeric value indicating the number of tokens passed into the array.

Returns
A numeric value indicating the number of tokens passed into the array.
Example

```bash
$var1="Mon:Tue:Wed:Thu:Fri:Sat:Sun";
$num_of_tokens=StrTok($var1,":","arr1");

Result:

$num_of_tokens=7
$arr1[0]="Mon"
$arr1[1]="Tue"
...
```

**StURIExist**

**Syntax**

```bash
StURIExist (strURL);
```

| strURL | Input URL string to be checked whether it exists or not. |

**Description**

Checks whether a file at the specified URL exists. Can be used for StoryTeller protocols, for example `dcresource:<id>, local:<id>` etc, as well as HTTP, FTP, and File protocols.

**Returns**

| 1 | The URL exists. |
| 0 | The URL does not exist. |

**Example**

```bash
$validbool = StURIExist("dcresource:e16150ac-e5bf-434e");
```

**Subst**

**Syntax**

```bash
Subst (tbl_file, str_key);
```

| tbl_file | A string specifying the path and filename of the substitution table. |
| str_key | The key of the entry to search for. |

Leading and trailing spaces are removed from the two strings on each line in the table. Strings that contain spaces must be enclosed in double quotes.

**Description**

Returns the first value (value column 0) in the specified substitution table entry.
**Returns**

Returns the first value (value column 0) of the entry specified by `str_key`. If the value does not exist, an empty string is returned.

**Example**

In this example, the following substitution table (`author.tbl`) is used.

```
//!CodePage UTF8!
//!multicolumn!
agatha christie 1890
alfred hitchcock 1899
```

The following script returns the first value from the entry with key `agatha`:

```perl
$surName = Subst("../data/tables/author.tbl","agatha");
```

When the script is run, the variable `$surName` gets the value "christie".

**Notes**

A substitution table is kept in memory until the script function `EraseSubst` is called for that table.

---

**SubstArr**

**Syntax**

`SubstArr(tbl_file, str_key, array);`

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tbl_file</code></td>
<td>A string specifying the path and filename of the substitution table.</td>
</tr>
<tr>
<td><code>str_key</code></td>
<td>A string specifying a key in the table.</td>
</tr>
<tr>
<td><code>array</code></td>
<td>A variable that receives the array of columns from the substitution entry.</td>
</tr>
</tbody>
</table>

**Description**

Looks up a key in a substitution table, and creates an array containing the entry values. Returns the number of entry values.
Returns

<table>
<thead>
<tr>
<th>num</th>
<th>If the key was found, returns the number of items returned in array.</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>The key was not found.</td>
</tr>
</tbody>
</table>

Example

In this example, the following substitution table (author.tbl) is used.

```plaintext
//!CodePage UTF8!
//!multicolumn!
agatha  christie  1890
alfred   hitchcock 1899
```

The following script creates an array for the entry with key agatha:

```plaintext
$nCol = SubstArr("../data/tables/author.tbl", "agatha", $addrArr);
$surName = $addrArr[0];
$birthDay = $addrArr[1];
```

Result:

$\text{nCol} = 2$

$s\text{urName} = \text{christie}$

$\text{birthDay} = 1890$

Notes

- A substitution table is kept in memory until the script function \text{EraseSubst} is called for that table.

SubStr

**Syntax**

```
SubStr(str, num_pos, num_len);
```

<table>
<thead>
<tr>
<th>str</th>
<th>An input string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_pos</td>
<td>A number specifying the first position of sub-string in the string.</td>
</tr>
<tr>
<td>num_len</td>
<td>A number specifying the length of sub-string in the string. If this number is not specified, the end of the sub-string is set to the end of the input string.</td>
</tr>
</tbody>
</table>

**Description**

Returns a sub-string of the input string. Equivalent to \$var1(num_1 [,num_2]) where \$var is a string variable.
Returns
A sub-string of the input string.

Example
$a="Intelligent Output Management";
$b=substr($a,13);
$c=$a(13);
$d=substr($a,13,6);

Result:
$b="Output Management"
$c="Output Management"
$d="Output"
$a(13,6)                   //also returns "Output"

SubStrRepl

Syntax
SubStrRepl(str, search_str, repl_str);

<table>
<thead>
<tr>
<th>str</th>
<th>An input string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>search_str</td>
<td>A search string to be replaced.</td>
</tr>
<tr>
<td>repl_str</td>
<td>A replacement sub-string.</td>
</tr>
</tbody>
</table>

Description
Replaces a search string with a sub-string in an input string.

Returns
A string.

Example
$a="Intelligent Output Management";
$b=substrrepl($a, "Out", "In");

Result:
$b="Intelligent Input Management"
Terminate

Syntax

Terminate(num_exitcode);

| num_exitcode | A numeric value between 0 and 127. Default is 0. Values other than 0-127 are not recommended since they may have different meanings depending on the operating system. |

Description

Makes the StreamServer shutdown gracefully when all jobs being processed have been completed, including the one that called the Terminate function. Once the Terminate function has been called, no new jobs will be started. You can use the exit code to inform the application that started the StreamServer why the StreamServer has been shutdown.

In UNIX bash shell scripts the exit code is stored in `$?` after StreamServe has exited.

**Note:** If several job threads call the Terminate function at the same time the exit code is undefined. When there are several calls to the Terminate function within the same job, the last call will set the exit code.

Returns

The Terminate function always returns 1.

Examples

**Example 91 Using Terminate**

To shutdown the StreamServer when all jobs have been completed:

```
Terminate(0);
```

**Example 92 Using Terminate**

To use the exist code to leave a message to the application that started the StreamServer:

```
if ($encounteredWords="stop for two weeks")
    Terminate(2);
else if ($encounteredWords="stop for three weeks")
    Terminate(3);
```
Example 93 Using Terminate

In a Windows command file you can use the `%ERRORLEVEL%` variable with the exit code:

```bash
strsc -a start.arg
@echo off
if %ERRORLEVEL% EQU 0 goto :END
if %ERRORLEVEL% GEQ 4 goto :END
goto return%ERRORLEVEL%

:return1
   echo The server stopped with an error
   goto :END
:return2
   echo The server encountered the words "stop for two weeks" in the data stream
   goto :END
:return3
   echo The server encountered the words "stop for three weeks" in the data stream
   goto :END
:END
```

ToLower

**Syntax**

```
ToLower(str);
```

| **str** | A string. |

**Description**

Converts all characters in a string to lower case.

**Returns**

A copy of the input string with all characters in lower case.

**Example**

```
$var1="STREAMSERVE";
$var2=ToLower($var1);
Result:
$var2="streamserve"
```
**ToUpper**

**Syntax**

ToUpper(str);

| str | A string. |

**Description**

Converts all characters in a string to upper case.

**Returns**

A copy of the input string with all characters in upper case.

**Example**

```plaintext
$var1="StreamServe";
$var2=ToUpper($var1);
Result:
$var2="STREAMSERVE"
```

**U**

**UpdateDocStatus**

**Syntax**

UpdateDocStatus(status[, error_code]);

<table>
<thead>
<tr>
<th>status</th>
<th>A string specifying the new document status. The following values are available: stored, processing, processed, delivering, delivered, terminated, suspended, deleted, inprogress, partiallyprocessed, custom, custom1, custom2, custom3, custom4, custom5, custom6, custom7, custom8, custom9.</th>
</tr>
</thead>
<tbody>
<tr>
<td>error_code</td>
<td>The error code to update the repository with when the status is error.</td>
</tr>
</tbody>
</table>

**Description**

Updates status of the current document in the repository. The function can be used at the following post-processing scripting levels:

Before and after document, process and page.
Returns

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Status updated.</td>
</tr>
<tr>
<td>num&lt;&gt;0</td>
<td>Error</td>
</tr>
</tbody>
</table>

Example

```php
$ret = UpdateDocStatus("processed");
```

UpdateExternalCompletionStatus

UpdateExternalCompletionStatus is an old function and is removed.

UpdateJobStatus

**Syntax**

```php
UpdateJobStatus(status[, error_code]);
```

<table>
<thead>
<tr>
<th>status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A string specifying the new job status. The following values are available: stored, processing, processed, delivering, delivered, terminated, suspended, deleted, inprogress, partiallyprocessed, custom, custom1, custom2, custom3, custom4, custom5, custom6, custom7, custom8, custom9.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>error_code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>The error code to update the repository with when the status is error.</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

Updates the status of the current post-processor job in the Post-processor repository. The function can be used at the following post-processing scripting levels:

Before and after document, process and page.

Returns

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Status updated.</td>
</tr>
<tr>
<td>num&lt;&gt;0</td>
<td>Error</td>
</tr>
</tbody>
</table>

Example

```php
$ret = UpdateJobStatus("processed");
```
UpdatePPDocStatus

**Syntax**

```javascript
UpdatePPDocStatus(doc_handle, status[, error_code]);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>doc_handle</code></td>
<td>Identifies the document.</td>
</tr>
<tr>
<td><code>status</code></td>
<td>A string specifying the new job status. The following values are available: stored, processing, processed, delivering, delivered, terminated, suspended, deleted, inprogress, partiallyprocessed, custom, custom1, custom2, custom3, custom4, custom5, custom6, custom7, custom8, custom9.</td>
</tr>
<tr>
<td><code>error_code</code></td>
<td>The error code to update the repository with when the status is error.</td>
</tr>
</tbody>
</table>

**Description**

Updates repository status of the document specified in the `doc_handle` parameter returned by either the `GetFirstPPDoc()` or the `GetNextPPDoc()` function.

**Returns**

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Status updated</td>
</tr>
<tr>
<td>num&lt;&gt;0</td>
<td>Error</td>
</tr>
</tbody>
</table>

**Example**

```javascript
$doc_id = GetFirstPPDoc();
while (num($doc_id) > 0) {
    $ret = UpdatePPDocStatus(num($doc_id), "processed");
    $doc_id = GetNextPPDoc($doc_id);
}
```

UpdatePPJobStatus

**Syntax**

```javascript
UpdatePPJobStatus(status[, error_code]);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>status</code></td>
<td>A string specifying the new job status. The following values are available: stored, processing, processed, delivering, delivered, terminated, suspended, deleted, inprogress, partiallyprocessed, custom, custom1, custom2, custom3, custom4, custom5, custom6, custom7, custom8, custom9.</td>
</tr>
<tr>
<td><code>error_code</code></td>
<td>The error code to update the repository with when the status is error.</td>
</tr>
</tbody>
</table>
Description
Updates status of the current post-processor job in the repository.

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Status updated.</td>
</tr>
<tr>
<td>num&lt;&gt;0</td>
<td>Error</td>
</tr>
</tbody>
</table>

Example
```bash
$doc_id = GetFirstPPDoc();
while(num($doc_id) > 0)
{
    $ret = UpdatePPJobStatus(num($doc_id), "processed");
    $doc_id = GetNextPPDoc($doc_id);
}
```

UpdateSegDocStatus

Syntax
```bash
UpdateSegDocStatus(doc_handle, status[, error_code]);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc_handle</td>
<td>An identifier specifying a document.</td>
</tr>
<tr>
<td>status</td>
<td>A string specifying the new job status. The following values are available:</td>
</tr>
<tr>
<td></td>
<td>stored, processing, processed, delivering, delivered, terminated, suspended,</td>
</tr>
<tr>
<td></td>
<td>deleted, inprogress, partiallyprocessed, custom, custom1, custom2, custom3,</td>
</tr>
<tr>
<td></td>
<td>custom4, custom5, custom6, custom7, custom8, custom9.</td>
</tr>
<tr>
<td>error_code</td>
<td>The error code to update the repository with when the status is error.</td>
</tr>
</tbody>
</table>

Description
Updates repository status of the document specified in the `doc_handle` parameter returned by either the `GetFirstSegDoc()`, `GetCurrSegDoc()`, or the `GetNextSegDoc()` function.

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Status updated.</td>
</tr>
<tr>
<td>num&lt;&gt;0</td>
<td>Error</td>
</tr>
</tbody>
</table>

Example
```bash
$doc_id = GetFirstSegDoc();
```
while (num($doc_id) > 0)
{
    $ret = UpdateSegDocStatus(num($doc_id), "processed");
    $doc_id = GetNextSegDoc($doc_id);
}

### UpdateSegJobStatus

**Syntax**

UpdateSegJobStatus(doc_handle, status[, error_code]);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>doc_handle</code></td>
<td>Identifies the document.</td>
</tr>
<tr>
<td><code>status</code></td>
<td>A string specifying the new job status. The following values are available:</td>
</tr>
<tr>
<td></td>
<td>• stored</td>
</tr>
<tr>
<td></td>
<td>• processing</td>
</tr>
<tr>
<td></td>
<td>• processed</td>
</tr>
<tr>
<td></td>
<td>• delivering</td>
</tr>
<tr>
<td></td>
<td>• delivered</td>
</tr>
<tr>
<td></td>
<td>• terminated</td>
</tr>
<tr>
<td></td>
<td>• suspended</td>
</tr>
<tr>
<td></td>
<td>• deleted</td>
</tr>
<tr>
<td></td>
<td>• inprogress</td>
</tr>
<tr>
<td></td>
<td>• partiallyprocessed</td>
</tr>
<tr>
<td></td>
<td>• custom</td>
</tr>
<tr>
<td></td>
<td>• custom1</td>
</tr>
<tr>
<td></td>
<td>• custom2</td>
</tr>
<tr>
<td></td>
<td>• custom3</td>
</tr>
<tr>
<td></td>
<td>• custom</td>
</tr>
<tr>
<td></td>
<td>• custom5</td>
</tr>
<tr>
<td></td>
<td>• custom6</td>
</tr>
<tr>
<td></td>
<td>• custom7</td>
</tr>
<tr>
<td></td>
<td>• custom8</td>
</tr>
<tr>
<td></td>
<td>• custom9</td>
</tr>
<tr>
<td><code>error_code</code></td>
<td>The error code to update the repository with when the status is error.</td>
</tr>
</tbody>
</table>
Description
Updates repository status of the post-processor job that contains the document specified in the doc_handle parameter returned by either the GetFirstSegDoc(), GetCurrSegDoc(), or GetNextSegDoc() function.

Returns

<table>
<thead>
<tr>
<th>num</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Status updated.</td>
</tr>
<tr>
<td>num&lt;&gt;0</td>
<td>Error</td>
</tr>
</tbody>
</table>

Example

```$doc_id = GetFirstSegDoc();
while(num($doc_id) > 0)
{
    $ret = UpdateSegJobStatus(num($doc_id), "processed");
    $doc_id = GetNextSegDoc($doc_id);
}$$```

URLClose

Syntax

`URLClose(connectionID);`

**connectionID** A string specifying the connection ID you specified when opening the file. The connectionID is specified in the URLOpen() function.

Description
Closes the specified connection opened with URLOpen().

Returns
A number indicating whether the file was successfully closed.

<table>
<thead>
<tr>
<th>num</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The file was successfully closed.</td>
</tr>
<tr>
<td>1</td>
<td>Failure.</td>
</tr>
</tbody>
</table>

Example

```//Close "myconnection" and store the returned value in $closeok
$closeok = URLclose("myconnection");```
URLExist

Syntax

```text
URLExist(strURL);
```

| strURL | Input URL string to be checked whether it exists or not. |

Description

Checks whether a file at the specified URL exists. Can be used for HTTP, FTP and File protocols.

Returns

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The URL exists.</td>
</tr>
<tr>
<td>0</td>
<td>The URL does not exist.</td>
</tr>
</tbody>
</table>

Example

```
$validbool = URLExist("http://myusername:mypassword@myserver/file.txt");
```

URLOpen

Syntax

```text
URLOpen(connectionID, strURL, "r");
```

<table>
<thead>
<tr>
<th>connectionID</th>
<th>A name you specify which will be used in all the following URL functions to identify which URL to access.</th>
</tr>
</thead>
<tbody>
<tr>
<td>strURL</td>
<td>A string specifying the URL of the file.</td>
</tr>
<tr>
<td>&quot;r&quot;</td>
<td>Opens the connection in read mode.</td>
</tr>
</tbody>
</table>

Description

Opens a file at the specified URL for reading. The URL format must be one of the following:

- `http://[<username>:<password>@]<httpserver>[:<port>]/<path>`
- `ftp://[<username>:<password>@]<ftpserver>[:<port>]/<path>`
- `file://[[\]\]<fileserver>]/<path>`

**Note:** The data within square brackets is optional.

You can also specify just a path, this will default to a `file://<path>` string.
Returns
A number indicating whether the file was successfully opened.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The file was successfully opened.</td>
</tr>
<tr>
<td>1</td>
<td>Error</td>
</tr>
</tbody>
</table>

Example

//Open myfile.txt for reading

$readok = URLopen("my_conn_ID", "ftp://myserver/myfile.txt", "r");

URLReadLn

Syntax

URLReadLn(connectionID, var);

<table>
<thead>
<tr>
<th>connectionID</th>
<th>The connection ID you specified when opening the file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>var</td>
<td>A string specifying the variable in which to store the data.</td>
</tr>
</tbody>
</table>

Description

Reads a line from the specified file until a new line or an end of file is reached, and returns the line to a specified variable. The new line character is not included in the result. You must open the file to be read with URLOpen before calling URLReadLn. Use the URLClose function to close the file after reading.

Returns
A number indicating whether the line was successfully read.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The line was successfully read.</td>
</tr>
<tr>
<td>1</td>
<td>Failure</td>
</tr>
</tbody>
</table>

Example

while(URLreadln("my_connection", $inbuf) = 0){
}

Reads from the file connected via my_connection until EOF is reached.
Utf8DecodeString

Syntax

Utf8DecodeString(InputStr);

**InputStr**  Input string to be decoded.

**Description**

Decodes an UTF8 encoded text string.

**Returns**

**str**  The decoded string. If decoding fails an empty string is returned.

**Example**

$DecodedString = utf8DecodeString("string to be decoded");

Utf8EncodeString

Syntax

Utf8EncodeString(InputStr);

**InputStr**  Input string to be UTF8 encoded.

**Description**

Encodes a text string to UTF8 format.

**Returns**

**str**  The encoded string. If encoding fails an empty string is returned.

**Example**

$Utf8_EncodedString = utf8EncodeString("string to be encoded");

W

WeekOfYear

Syntax

WeekOfYear([yy] yymmd);  

**[yy] yymmd**  A date string.
Description
Returns the year and the number of a week within the year.

Returns
A string \( (str) \) specifying the year and number of the week within the year in \( yyww \) format.

Example
\$a="990512";
\$b=weekofyear($a);
Result:
\$b="9919"

WriteSubst

Syntax
WriteSubst(tbl_file);

Description
Writes a substitution table to a specified path. Any external modifications to the file after it has been loaded are overwritten. See also ReadSubst.

Returns

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>File was written successfully.</td>
</tr>
<tr>
<td>0</td>
<td>File was not written.</td>
</tr>
</tbody>
</table>

Example
The following script writes a substitution file to \( .../data/tables/author.tbl \):
SetSubst("..../data/tables/author.tbl","agatha",0,"christie");
SetSubst("..../data/tables/author.tbl","agatha",1,"1890");
SetSubst("..../data/tables/author.tbl","alfred",0,"hitchcock");
SetSubst("..../data/tables/author.tbl","alfred",1,"1899");
WriteSubst("..../data/tables/author.tbl");
Result (author.tbl):

//!multicolumn!
agatha christie 1890
alfred hitchcock 1899
Notes

- A substitution table is kept in memory until the script function `EraseSubst` is called for that table.

- For backwards compatibility, `Subst` and `SubstArr` do not remember comments made in the substitution file. If you want `WriteSubst` to keep your comments, call `GetSubst` before any other substitution function accesses the substitution file.

- The column separator in the saved file will always be a tab-character, regardless of an earlier column separator.

- If a column index larger than zero has been set using scripts in a non `// !multicolumn!` marked file, the file will be converted into multi-column when saved, and key values containing white spaces will be converted adding quotes or `<hex>` notations if necessary.

- `<hex>` notations other than `<0d><0a><09>` are not kept when saving.

- If `//!codepage!` is specified in the substitution table file, the remaining file will be saved using the given codepage.

- The `WriteSubst` function will automatically add double quotes (""") around empty columns in the table. This is to make sure that the correct value is returned by the `GetSubst` function, as this function does not take the column separators into account.
Lotus Notes Fetch Wizard

You can retrieve data from a Lotus Notes database, for example a name or an address, to a StreamServe Process. Instead of updating all your StreamServe Projects, you can update the Lotus Notes database and retrieve the correct information from the database.

The Lotus Notes Fetch wizard inserts a code block in a script in the Process. The code block generated by the wizard retrieves data from the Lotus Notes database. It uses a variable in the Event to identify the value to be retrieved from the Lotus Notes database, and stores the retrieved data in a variable in the Process.

Requirements

A Lotus Notes client, version 4.5 or higher, must be installed on the same computer as Design Center. The notes.ini file in the Lotus Domino or Lotus Notes installation refers to the Lotus ID file which must contain data for authenticating read access to the Lotus Notes database.

Specifying the search criteria

If more than one document in the Lotus Notes database matches the specific criteria, data will only be returned from the first document that is found. Increasing the number of search criteria will result in a more accurate search. The Lotus Notes Fetch wizard cannot check the validity of the criteria you specify. If you use invalid search criteria, the Lotus Notes Fetch Wizard will return a blank line, or the variable name, instead of the variable in the output document or data flow.

Retrieving data from more than one document

Each Lotus Notes Fetch wizard operation will return data from one document. If you want to retrieve data from more than one document, you will have to specify a Lotus Notes fetch for each document.
The generated code block

The syntax of the generated code block follows the StreamServe scripting language.

You can move an entire code block, but not specific sections of a generated code block. If you do, the generated code block may not work properly.

You can have multiple code blocks in the same script. However, the fetch wizard does not support generated blocks inside blocks, for example

```java
if(...)
{
   // Generated block 1
}
else
{
   // Generated block 2
}
// Generated block 3
```

Editing a code block generated by the wizard

You should always use the Lotus Notes Fetch wizard to edit generated code blocks. Editing code blocks any other way may result in failure to retrieve data from the Lotus Notes database.

Limitation

The Lotus Notes Fetch Wizard does not support Unicode, you can only use ASCII.
Generating a code block with the Lotus Notes Fetch wizard

To generate a code block using the Lotus Notes Fetch wizard, you need to do the following:

- Assign a variable in the Event to identify the data to be fetched
  Create a variable in the Event to identify the data to be retrieved from the Lotus Notes database. Avoid using the same names for manually created variables and generated variables. This might cause problems when using the wizard since the wizard creates its own variables for the password, server name, and the database.

- Create a variable in the Process to store the fetched data
  Create a variable in the Process in which to store the data retrieved from the Lotus Notes database.

- Generate the code block that will retrieve data from the Lotus Notes database
  You use the Lotus Notes Fetch wizard to specify which data the code block should retrieve from the Lotus Notes database, and where to store the retrieved data.
  See Information required by the Lotus Notes Fetch wizard on page 358, and Generating the code block on page 358.

- Specify the -lotusnotes argument in the Project
  You must specify the -lotusnotes argument when you export the StreamServe Project.
Generating a code block with the Lotus Notes Fetch wizard

Lotus Notes Fetch Wizard

Information required by the Lotus Notes Fetch wizard

- The name and the password to the Lotus Notes server
- The name of the Lotus Notes database
- The document view and the form in the Lotus Notes database that contain the data to be retrieved
- The field to be retrieved from the Lotus Notes database. The code block uses the value of the variable you specified in the Event (Criteria) to search for a matching field entry (Field name).
- The variable in the Process in which to store the retrieved data.

<table>
<thead>
<tr>
<th>Document Search Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field name</td>
</tr>
<tr>
<td>Criteria</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field to variable mappings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field name</td>
</tr>
<tr>
<td>Variable name</td>
</tr>
</tbody>
</table>

Generating the code block

See also Information required by the Lotus Notes Fetch wizard on page 358.

1. In the Process, right-click the variable to which you want to add the code block that will retrieve data from the Lotus Notes database, and select Edit Script. The Script editor opens.
2. Select Before from the Trigger drop down list.
4. Select Lotus Notes Fetch from the Available wizards drop down list.
5. Enter a name for the code block if you want a specific name to be used in the generated code block. Otherwise leave blank.
6. Click Insert. The Lotus Notes Fetch wizard opens.
7. Follow the wizard to generate the code block.
8. When the wizard is finished, the generated code block is displayed in the Script editor.
Editing a generated code block

If, for example, the search criteria or the fields in the Notes database have been changed, you may need to edit or update an existing code block with the new values. Always use the Lotus Notes Fetch wizard to edit a generated code block.

See also Information required by the Lotus Notes Fetch wizard on page 358.

To edit a generated code block

1. In the Process, right-click the variable with the code block that you want to edit, and select Edit Script. The Script editor opens.
2. Select Edit from the Code Wizard menu. The Edit block dialog box opens.
3. Click Edit. The Lotus Notes Fetch wizard carries out an integrity check on the generated code block.

If the code block passes the integrity check, a confirmation message is shown. Click OK to open the wizard and edit the information as required.

If the code block does not pass the status check, an error message is displayed. The error message describes the type of error that was found in the validation.
Deleting a generated code block

1. In the Process, right-click the variable with the code block that you want to delete, and select Edit Script. The Script editor opens.
2. Select Delete from the Code Wizard menu. A warning message is shown.
3. Click Yes to delete the generated code block. The code block is deleted.

Note: If you click Cancel in the Script window, the generated code block will be restored.