StreamServe™ Persuasion™

StreamServe Persuasion SP5 Scripting Reference

Reference Manual

Rev C

StreamServe Persuasion SP5 Scripting Reference Reference Manual
Rev C

Contents

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

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

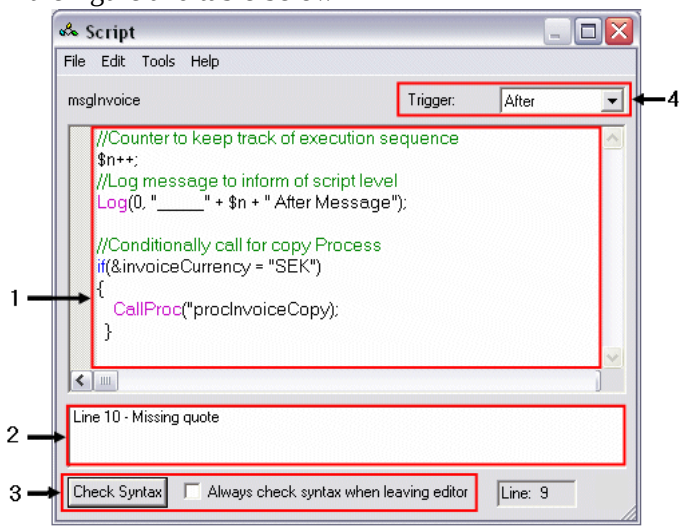


Figure 1 The script editor.

1	The script view where you type in the scripts. You can enter as many scripts as you need in this view.
2	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.
3	Syntax check. You can check the syntax manually (click **Check Syntax**) or select to always check the syntax before closing the script editor.
4	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.

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.

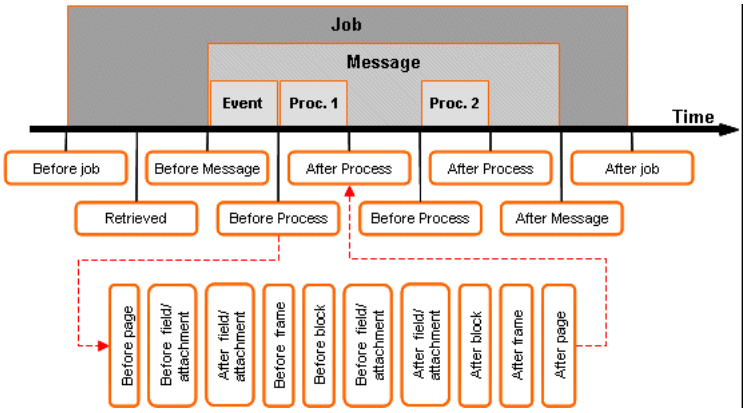


Figure 2 Script execution order.

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

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

In this section

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.

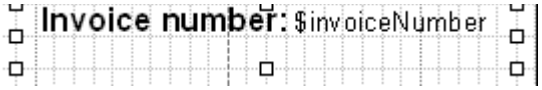
You can create field variables in Events. For example, you can assign the global variable `$countryCode` to the field `CountryCode`. You can also create global variables in scripts.

Examples

Example 1 *Global variable $countryCode used in a script.*

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

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



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.

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.

String	M	r		M	i	k	e		J	o	n	e	s
Position	1	2	3	4	5	6	7	8	9	10	11	12	13

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:

```
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*

```
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*

```
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*

```
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*

```
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.*

```
if(&countryCode = "USA")
{
    $overlay = "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.*

```
$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.

-	**Subtraction** Subtracts the value of one expression from another. `$var = expression1 - expression2`
+	**Addition** Adds the value of one numeric expression to another, or concatenates two strings. `$var = expression1 + expression2`
*	**Multiplication** Multiplies the value of two expressions. `$var = expression1 * expression2`
/	**Division** Divides the value of two expressions. `$var = expression1 / expression2`
++	**Incrementation** Increments the value of a variable by one. `$var++`

`--`	**Decrementation** Decrements the value of a variable by one. `$var--`
`-=`	**Subtraction Assignment** Subtracts the value of an expression from the value of a variable, and assigns the result to the variable. `$var -= expression` Equivalent to `$var = $var - expression`. Works only with numeric values.
`+=`	**Addition Assignment** Adds the value of an expression to the value of a variable, and assigns the result to the variable. `$var += expression` Equivalent to `$var = $var + expression`. Works only with numeric values.
`=`	**Assignment** Assigns a value to a variable. `$var = expression`
`==`	**Equality** Returns a Boolean value indicating the result of the comparison. `expression1 == expression2`
`!=`	**Inequality** Returns a Boolean value indicating the result of the comparison. `expression1 != expression2`
`>`	**Greater than** Returns a Boolean value indicating the result of the comparison. `expression1 > expression2`
`<`	**Less than** Returns a Boolean value indicating the result of the comparison. `expression1 < expression2`
`>=`	**Greater than or equal to** Returns a Boolean value indicating the result of the comparison. `expression1 >= expression2`
`<=`	**Less than or equal to** Returns a Boolean value indicating the result of the comparison. `expression1 <= expression2`

AND	**Logical AND** Performs a logical conjunction on two expressions. `$var = expression1 AND expression2`
OR	**Logical OR** Performs a logical disjunction on two expressions. `$var = expression1 OR expression2`
NOT	**Logical NOT** Performs logical negation on an expression. `$var = NOT expression1`

Delimiters

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

;	Terminates statements.
" "	Delimiter for text strings.
,	Separates items in lists.
.	Decimal separator.
( )	Surrounds expressions, and parameter lists in functions.
{ }	Surrounds consecutive statements, making them appear as one simple statement. Equivalent to `begin … end`.
[ ]	Surrounds indexes in an array.
<>	Surrounds hexadecimal numbers, and lists of hexadecimal numbers.

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.*

```
$value = Num($amount) - 1;
```

Example 17 *String expression.*

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

Example 18 *Boolean expression.*

```
if (Num($cost) < 100)
{
   $value = Num($amount) - 1;
}
```

Variables in numeric expressions

You must use the script function `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.*

```
$a = 20;
$b = Num($a) + 30;
Result: $b=50
```

Example 20 *Not using the Num script function.*

```
$a = 20;
$b = $a + 30;
Result: $b=2030
```

String Extraction Expression (PageIN only)

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

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:

```
$strg = 37 14 7;
```

Assignment statements

Assignment statements are used to assign values to variables.

Syntax

```
variable=expression;
```

Examples

Example 21 *Assigning 1 to $counter.*

```
$counter = 1;
```

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

```
$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

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

Examples

Example 23 *if statement.*

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

The if...else statement

Syntax

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

Examples

Example 24 *if...else statement.*

```
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

```
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.*

```
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

```
for (statement1; expression; statement2) statement[s]3
```

Examples

Example 28 *for loop statement.*

```
{
 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.*

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

Example 30 Section comment

```
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 a 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

```
$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

```
$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

```
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 a 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.

```
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:

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

Functions returning values

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

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

This function can be called from a script:

```
$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:

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

This function can be called from a script:

```
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*

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

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

Result

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*

```
{
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`.

```
//!multicolumn!
//key       value 0     value 1
alfred      hitchcock   1899
```

Script functions reference

The script functions reference contains descriptions of all built-in script functions.

In this chapter

General functions

In this section

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.

Function name	Description
ArraySize	Counts the number of elements stored in an array.
FindInArray	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.
FindStringExactInArray	Searches for a value in an array. Returns the position of the array element that contains exactly the specified value.

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.

Function name	Description
CallBlock	Calls a Free Block.
FirstBlockInst	Determines if the current block is the first of its kind in a frame.

Function name	Description
FirstBlockOnPage	Determines if the current block is the first of its kind in a frame on each page.
LastBlockInst	Determines if the current block is the last of its type.
LastBlockOnPage	Determines if the current block is the last block in a frame on each page.

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.

Function name	Description
GetExtJobId	Returns the external ID of the current job.
GetIntJobId	Returns the internal ID of the current job.
GetTopJobId	Returns the internal ID of the input job (top job).
SetExtJobId	Sets the external ID of the current job.

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.

Function name	Description
DeleteJobResource	Marks job resources to be removed. The marked resources are deleted when all references to the job resource are released.
GetJobResourceIndex	Retrieves the index of a job sent to a Job Resource output connector.
OutputLXFJobResource	Adds an LXF overlay from a job resource to a PageOUT Process.
OutputLXFJobResourcePrnOffs	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.

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.

Function name	Description
GetJobStatus	Returns the status of the current job.
SetJobFailed	Specifies the current job status as failed.

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.

Function name	Description
Log	Appends a string to the log file.
PreProcLog	During the pre-process phase, appends a string to the log file.

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.

Function name	Description
OutputLXFJobResource	Adds an LXF overlay from a job resource to a PageOUT Process.
OutputLXFJobResourcePrnOffs	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.
OverlayGetNumPages	Returns the total number of pages in a multi-page overlay.

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.

Function name	Description
ConnectorPage	Returns the number of pages (in the current job) that have been processed so far on the current connector.
ConnectorPages	Returns the total number of pages (in the current job) that will be processed on the current connector.
DocPage	Returns the current page number of the current document.
DocPages	Returns the total number of pages in the current document.
FrameOverflow	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.
NewPage	Generates a new page according to the context in which it is called.

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.

Function name	Description
Mat	Copies characters from the input matrix. Removes leading and trailing blanks.
MatBlk	Copies characters from the input matrix. Keeps leading and trailing blanks.
MaxCol	Returns the number of columns in the input matrix.
MaxRow	Returns the number of rows in the input matrix.
PatternResult	Used to check for a pattern in a Message or in a block.

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.

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

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.

Function name	Description
AtJobEnd	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.
CallProc	Invokes a Process in the Message from which it is called.
CancelJob	Cancels the current job. The cancellation is executed after pre-processing, i.e. all Events are pre-processed before the job is interrupted.
Clear	Clears the specified variables.
ClearVars	Clears all variables in a job.
CreateGlobalSerNo	Creates a serial number key in the repository.
CurrJobOwner	Returns the current job owner.
DumpVariables	Writes all variables that have values to a file.
EndDocument	Forces a Document Begin at the end of the Process from which the script function is called.

Function name	Description
EndMessage	Used in a Retrieved script to set Event Order Last for the Event.
Eval	Evaluates all variables in the argument string.
Execute	Executes an external program, script, bat file or process.
Exist	Checks if a directory or file exists.
FlushOutput	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.
ForcePoll	Signals an input connector to poll data.
GetConnectorValue	Returns the value of a named field from a connector.
GetCurrModuleinfo	Returns the current Event or Process status, depending on where the script is executed.
GetGlobalSerNo	Calculates the next counter value in a number sequence according to a specified key stored in the repository.
GetGlobalSerNoRange	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.
GetHTTPHeaderValue	Returns the value of a named field in the HTTP header of a job created by a request on an HTTP input connector.
GetMetaDataMessage	Returns the current value of a metadata.
GetSerNo	Calculates the next counter value in a number sequence according to a specified key stored in a file.
GetTextFormattingInfo	Enables splitting of LXF text into separate areas, for example to create multi-column output.
GetXoffs	Returns the current X offset value in millimeters.
GetYoffs	Returns the current Y offset value in millimeters.
InConnectorName	Returns the name of the current input connector.
IncProcStatCounter	Increases the value of the argument with 1.
NewJob	Replaces the `JobBegin` and `JobEnd` functions, and is used to split large input jobs into smaller jobs.
NoFormFeed	Suppresses the form feed sequence output at end of page.
NumberOfEvents	Counts the number of Events in a job.
NumberOfEventsEx	Returns the number of the specified Event.

Function name	Description
OutputString	Outputs a line of text directly to the output buffer of a StreamOUT or XMLOUT Process.
PreProc	Determines whether or not StreamServer is running in pre-process phase.
RGBcolor	Specifies an RGB color. For example, the color of a font or an area.
StartTimer	Starts a timer in the StreamServer to measure processing time. See also *StopTimer* on page 330.
StopTimer	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 *StartTimer* on page 316.
SetCopies	Sets the number of copies of the current Process.
SetDestPath	Sets a destination path, and optionally, a destination file.
SetFontProperties	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.
SetJobDescr	Sets the description of the current job. The job description is used in the log and as the title of spooled job (Windows/ NetWare).
SetJobOwner	Sets the owner of the print job created by the Spool output connector.
SetLanguage	Sets the language to be used for the current job. This setting is used with the StreamServe Language Set (`*.sls`) file.
SetSerNo	Sets the value of a serial number in a number sequence.
Skip	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.
Sort	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.
Terminate	Makes the StreamServer shutdown gracefully when all jobs being processed have been completed, including the one that called the Terminate function.

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 consists 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.

Function name	Description
Mod10	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.
Mod10L	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.
Mod11	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`.

COM functions

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_UI2
  VT_UI4
  VT_INT
  VT_UINT
  VT_VARIANT

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

COM 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.

Function name	Description
COMCreateObject	Loads the COM component with the specified ProgID and associates it with the specified handle.
COMDestroyObject	Unloads the COM component associated with the specified handle.
COMInvoke	Invokes the specified method for the COM component associated with the specified handle.
COMSetProperty	Sets the specified property for the COM component associated with the specified handle.
COMGetProperty	Retrieves the value of the specified property for the COM object associated with the specified handle.

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.

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

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.

Function name	Description
GetDistributionChannelsForRole	Returns the distribution channels for a role, for a specific document type.
GetDistributionChannelsForUser	Returns the distribution channels for a user, for a specific document type.
GetRolesForUser	Returns the roles associated with a user.

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.

Function name	Description
ConnectorPageActual	Returns the number of logical pages sent to the output connector, including inserts but excluding skipped pages.
CurrDocProperty	Returns the specified document property for the current document.
CurrJobProperty	Returns the specified job property.
CurrMetadata	Returns the value of the specified metadata for the current document.
CurrReposProperty	Returns the specified repository property.
DeclareMetadata	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.
DocInsertedPages	Returns the number of pages inserted in the output queue within the current document.
DocPage	Returns the current page number of the current document.
DocPageActual	Returns the number of document pages sent to the output connector, including inserts but excluding skipped pages.
EnableEnvelopeCheck	Enables or disables a check for an inconsistent number of sheets when using enveloping.
GetCurrSegDoc	Returns an index that identifies the current document, in the current segment, in the set of documents sent to the output connector.
GetDocActualInserts	Returns the inserts assigned to the document.
GetDocEnvelopeCount	Returns the number of envelopes for the current document.
GetDocInsertCount	Returns the number of inserts for the current document.
GetDocInsertEquivals	Returns the insert equivalent count for the current document.
GetDocSheetCount	Returns the number of sheets within the current document.
GetEnvelNr	Returns the envelope number of the current document.
GetFirstPPDoc	Returns a counter index that identifies the first document, in a set of documents sent to the output connector in a Post-processor job.
GetFirstSegDoc	Returns the index that identifies the first document in the current segment, in the set of documents sent to the output connector.

Function name	Description
GetMailMachine	Returns the name of the enveloping machine to which the current document was assigned.
GetNextPPDoc	Returns a counter index that identifies the next document in a set of documents in the post-processor job sent to the output connector.
GetNextSegDoc	Returns an index that identifies the next document, in the current segment, in the set of documents sent to the output connector.
GetPPDocId	Returns the document ID of a document stored in a Post-processor repository.
GetPPDocProperty	Returns the value of the specified property for the specified document.
GetPPError	Returns the last error that occurred during the Post-processor job.
GetPPJobId	Returns the job ID of a job stored in a Post-processor repository.
GetPPJobProperty	Returns the value of the specified property for the job containing the specified document.
GetPPMetadata	Returns the value of the specified metadata for the specified document.
GetPPReposProperty	Returns the value of the specified repository property storing the specified document.
GetSegDocProperty	Returns the specified document property for the specified document.
GetSegJobProperty	Returns the specified job property for the specified document.
GetSegment	Returns the current segment number. This is the same number that replaces the `%{SEGMENT}` variable in the output file name.
GetSegMetadata	Returns the value of the specified metadata for the document in the specified segment.
GetSegReposProperty	Returns the specified repository property for the specified document.
InsertOverlay	Adds an overlay to the current page.
InsertPage	Inserts a new page in the output stream.
IsFirstDocInEnvelope	Checks if the current document is the first document in the envelope.
IsFirstPageInEnvelope	Checks if the current page is the first page in the envelope.
IsFirstSegment	Checks if the current segment is the first segment of post-processor job.
IsInsertingPage	Checks if the current page is being inserted to the output queue.
IsLastDocInEnvelope	Checks if the current document is the last document in the envelope.
IsLastPageInEnvelope	Checks if the current page is the last page in the envelope.
IsLastSegment	Checks if the current segment is the last segment of post-processor job.

Function name	Description
JobInsertedPages	Returns the number of pages inserted to the output queue within the current job.
NextDoc	Skips the current document and processes the next document.
NextPage	Skips the current page and processes the next page.
NextProc	Skips the rest of the current Process, and processes the next Process.
NextSegment	Skips the rest of the current segment, and processes the next segment.
PPDocCount	Returns the number of documents sent to the output connector.
SaveCurrMetadata	Saves the metadata for the current document in the repository.
SavePPMetadata	Saves the metadata for the document specified in the *doc_handle* parameter returned by either the `GetFirstPPDoc` or `GetNextPPDoc` function.
SaveSegMetadata	Saves the metadata for the document specified in the *doc_handle* parameter returned by either the `GetFirstSegDoc`, `GetCurrSegDoc`, or `GetNextSegDoc` function.
SetCurrMetadata	Sets string metadata for the current document.
SetCurrMetadataNum	Sets numeric metadata for the current document.
SetSegMetadata	Sets string metadata for the document specified in the *doc_handle* parameter returned by either the `GetFirstSegDoc`, `GetCurrSegDoc`, or `GetNextSegDoc` function.
SetSegMetadataNum	Sets numeric metadata for the document specified in the *doc_handle* parameter returned by either the `GetFirstSegDoc`, `GetCurrSegDoc`, or `GetNextSegDoc` function
SortSegDoc	Sorts the documents in the current segment according to the specified sort keys.
UpdateDocStatus	Updates status of the current document in the repository.
UpdateJobStatus	Updates the status of the current post-processor job in the Post-processor repository.
UpdatePPDocStatus	Updates repository status of the document specified in the *doc_handle* parameter returned by either the `GetFirstPPDoc` or `GetNextPPDoc` function.
UpdatePPJobStatus	Updates status of the current post-processor job in the repository.
UpdateSegDocStatus	Updates repository status of the document specified in the *doc_handle* parameter returned by either the `GetFirstSegDoc`, `GetCurrSegDoc`, or `GetNextSegDoc` function.

Function name	Description
UpdateSegJobStatus	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.

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.

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

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.

Function name	Description
Base64DecodeFile	Decodes a base64 encoded file and stores the result in an output file.
Base64EncodeFile	Encodes an input file to base64 format. The encoded data is stored in an output file.
CurrInFileName	Returns the name of the input file, if there is one, else an empty string.
CurrInRealPath	Returns the temporary file name of the input file, if there is one.
CurrInSavePath	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.
FileClose	Closes the specified file.
FileCopy	Copies the specified file, and leaves the original file unchanged.
FileDelete	Deletes the specified file.
FileMove	Moves the specified file from one location to another.
FileOpen	Opens the specified file for reading or writing according to mode specified.
FileReadLn	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.
FileSize	Returns the size of the specified file.
FileWrite	Writes data to the specified file.
FileWriteLn	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.
IoErrText	Returns the error text for IO functions.
MkDir	Creates a new directory using the specified path.
OutputFile	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.
URLClose	Closes the file at the specified URL.
URLExist	Checks if a file exists at the specified URL.
URLOpen	Opens a file at the specified URL for reading.

Function name	Description
URLReadLn	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..

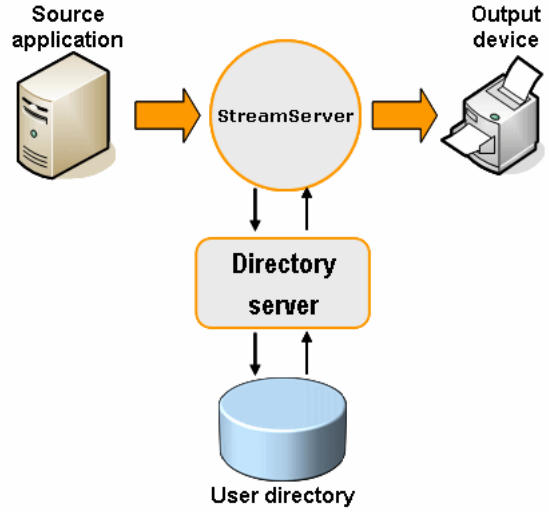


Figure 3 Connection to a directory server.

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.

Function name	Description
LdapAddAttrBinaryValue	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.
LdapAddAttrValue	Adds the specified value to the specified attribute. The changes are not written to the directory until the `LdapUpdateEntry` function is called.
LdapConnect	Connects to the directory server.
LdapConnectPersistent	Connects to the directory server.
LdapConnectSSL	Connects to the directory server using an encrypted connection.

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

Function name	Description
LdapWriteAttrBinaryValue	Writes the contents of the attribute to a file.

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:

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

connName	A name for the connection.
hostName	The host to connect to. The default port is 389, but can be overridden using *hostName*:*port*.
user	The user ID to log on to the server.
password	The password to log on to the server.

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.

```
// 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.

```
/* 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*/
    $groupmember = ldapgetobjectbydn("con", ldapgetattrvalue($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.

Function name	Description
LotusNotesAttachFile	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.
LotusNotesAddNote	Adds a Lotus Notes Note to a Lotus Notes database.
LotusNotesConnect	Establishes a connection to a Lotus Notes database on a Lotus Domino server.
LotusNotesGet	Retrieves data from a field in a Lotus Notes Note.
LotusNotesDisconnect	Disconnects an existing connection to a Lotus Notes database.
LotusNotesFind	Searches the database for Lotus Notes Notes that match the specified search criteria.
LotusNotesFirst	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.
LotusNotesLast	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.
LotusNotesNext	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.
LotusNotesPrev	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.

Function name	Description
LotusNotesSetDateTime	Sets the value of an item in a Note to the date and, optionally, the time specified.
LotusNotesSetNumber	Sets the value of an item in a Lotus Notes Note to the specified value.
LotusNotesSetText	Updates an item in a Lotus Notes Note with the specified text.

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.

```
// Domino Server
$server_name = "StuNt01/streamserve";
// The Lotus Notes database where you want to create a new document
$database_file = "AGFax.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.

Function name	Description
MsgCountId	Counts the number of instances of the specified block in the Message.
MsgGetValue	Returns the value of the specified field.
MsgValueExist	Checks if the specified field contains a value.

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.

Function name	Description
MsgProcCountId	Counts the number of instances of the specified block in the current Process.
MsgProcGetValue	Returns the value of a specified field within the current Process.
MsgProcValueExist	Checks if a specific field exists in the current Process.

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.

Function name	Description
MsgFrameCountId	Counts the number of instances of the specified block in the current frame.
MsgFrameGetValue	Returns the value of the specified field.
MsgFrameValueExist	Checks if the specified field contains a value.

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.

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

Message traversing examples

Example 44 Message traversing example

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

```
$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++;
      }
   }
}
```

Numeric operation 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.

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

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.

Function name	Description
GetCurrX	Returns the current position of a PageOUT object along the X axis in millimeters.
GetCurrBlockY	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.
GetCurrObjY	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.
SetCurrX	Sets a new X position for an object in PageOUT, such as a field, line, barcode, or text.
SetCurrY	Sets a new Y position for an object in PageOUT.
SetPrnXoffs	Adjusts X positions for overlay files (`*.prn`).
SetPrnYoffs	Adjusts Y positions for overlay files (`*.prn`).
SetXoffs	Adjusts X positions for all output fields, pictures, and overlays towards the right.
SetYoffs	Adjusts Y positions for all output fields, pictures, and overlays downwards.

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.

Function name	Description
IsPreview	Checks if the current job is a preview job or not.
GetPreviewType	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.
GetRequestedPreviewContentType	*Only available for preview calls from Ad Hoc Correspondence, Correspondence Reviewer, or Composition Center.* 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.

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.

```
$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.

Function name	Description
GetDesignCenterVersion	Returns the Design Center version that was used to create the Project.
GetPhysicalPlatform	Returns the name of the physical platform layer deployed in Control Center.
GetProjectName	Returns the name of the Design Center Project.
GetProjectRevision	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.
GetSender	Returns the name of the sender of the job. You specify the sender in the Runtime configuration in Design Center.

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.

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

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.

Function name	Description
SessionEnd	Ends a session in the StreamServer, and removes all associated variables.
SessionGetVariable	Retrieves the value of a given variable in a given session.
SessionSetVariable	Specifies a value for a given variable in a given session. If the variable does not exist, it will be created
SessionStart	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.

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.

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

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.

Function name	Description
Base64DecodeString	Decodes a base64 encoded text string.
Base64DecodeStringUsingCodepage	Decodes a base64 encoded text string that was encoded with the `Base64EncodeStringUsingCodepage` function.
Base64EncodeString	Encodes a text string to base64 format.
Base64EncodeStringUsingCodepage	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.
Str	Converts a numerical value to a string.
StrBlk	Removes leading white space (spaces and tabs) in a string.
StridX	Locates the first occurrence of a sub-string in a string.
StrLadj	Left-aligns and truncates (or extends) a string to a specified length.
StrLen	Returns the length of a string.
StrQTok	Takes four strings and a one-dimensional array and returns a numeric value indicating the number of tokens passed into the array.
StrrBlk	Removes trailing white spaces (spaces and tabs) from the input string.
StrTok	Takes two strings and a one-dimensional array and returns a numeric value indicating the number of tokens passed into the array.
SubStr	Returns a sub-string of the input string.
SubStrRepl	Replaces a search string with a sub-string in an input string.
ToLower	Converts all characters in a string to lower case.
ToUpper	Converts all characters in a string to upper case.
Utf8DecodeString	Decodes an UTF8 encoded text string.
Utf8EncodeString	Encodes a text string to UTF8 format.

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

```
//!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:

`../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`.

```
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.

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

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.

```
/*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.

Function name	Description
AmountValue	Converts a string to a numerical value using a specified format.
ConvCurrMsgToUC	Converts the encoding of input data from the specified code page to UCS-2.
HexStr	Copies a string and interprets numbers in angle brackets as two-digit hexadecimal numbers (useful for Escape sequences). Other characters are returned unchanged.
IsAmountValue	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`
IsNum	Checks if a string expression consists of digits.
NFormat	Converts a number to a string, formatting the string according to the specified format syntax.
Num	Converts a string expression to a numerical value.

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.

Function name	Description
OdbcBeginTrans	Begins a new transaction, which opens two possibilities: • To commit the transaction, call the `ODBCCommitTrans` function. • To end the transaction, call the `ODBCRollbackTrans` function.

Function name	Description
OdbcCommitTrans	Commits a transaction that you have opened with `ODBCBeginTrans`. The changes within the transaction are saved to the database and the transaction is ended.
OdbcConnect	Connects the user to the data source.
OdbcConnectW	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.
OdbcCloseQuery	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`.
OdbcDate2Date	Extracts the date from a date and time string.
OdbcDate2Time	Extracts the time from a date and time string.
OdbcDisconnect	Disconnects the server from the data source.
OdbcExecute	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.
OdbcExecuteEx	Same as `OdbcExecute`. The difference is that this function can be configured to accept that a `CREATE TABLE` statement fails.
OdbcGetOne	Executes an SQL statement and copies the values from the first row in the result set to script variables.
OdbcGetFirst	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.
OdbcGetNext	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.
OdbcRollbackTrans	Rolls back and ends a transaction that you have opened with `ODBCBeginTrans`.
OdbcSetCodepage	Sets the code page.

ODBC startup argument

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

Argument	Description
`-odbcTimeOut` *value*	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.

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);
```

num	A numeric expression.

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.

```
$c = Abs(Num($a) - 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);
```

tbl_file	A string specifying the path and filename of the substitution table.
str_key	A string specifying the key of the substitution table entry to search for.
num_col	A number specifying the value column. The first value column number is `0`.

str_value	A string representing a numeric value to add to the value found in column *num_col*. The number may be given with or without a decimal point. The function rounds the value to the maximum number of decimals given previously or in this parameter. You can use a negative number to subtract.

Description

Adds *str_value* to the value in *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 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 *str_value*.

Example

In this examples, the following substitution table (`author.tbl`) is used.

```
//!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.

```
$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 `astrid`:

```
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:

```
//!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);`

str_amount	A string specifying the amount.
thousand_sep	A string specifying the thousand separator.
decimal_sep	A string specifying the decimal separator.

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`*`);`

`array_name`	A string specifying the name of the array.

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")
   {
```

```
        $retSize=ArraySize($arrTele);
        $arrTele[$retSize]=$noDialled;
        $arrAmount[$retSize]=$callCharge;
    }
else
    $arrAmount[$retIndex]=$callCharge;
```

AtJobEnd

Syntax

`AtJobEnd(`*`eventname`*`);`

`eventname`	A string specifying the name of the Event.

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.

`1`	The Event will be triggered.
`0`	The Event will not be triggered.

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.

Output mode	Description
Process	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.
Document	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.
Job	Any attachment from Processes within the same job as the email connector Process can be attached to the email connector output.

Example

Before Process script defining the beginning of the attachment:

```
$attatchmentfile = AttachmentBegin();
```

After Process script defining the end of the attachment:

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

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

AttachmentEnd

Syntax

`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.

Output mode	Description
Process	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.
Document	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.
Job	Any attachment from Processes within the same job as the email connector Process can be attached to the email connector output.

Example

Before Process script defining the beginning of the attachment:

```
$attatchmentfile = AttachmentBegin();
```

After Process script defining the end of the attachment:

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

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

B

Base64DecodeFile

Syntax

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

InputFile	Full path and name of the input file.
OutputFile	Full path and name of the output file with decoded data. If the file already exists, it will be overwritten.

Description

Decodes a base64 encoded file and stores the result in an output file.

Returns

A number indicating whether the decoding was successful.

`1`	Decoding succeeded.
`0`	Decoding failed.

Example

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

Base64DecodeString

Syntax

```
Base64DecodeString(InputStr);
```

InputStr	Input string to be decoded.

Description

Decodes a base64 encoded text string.

Returns

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

Example

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

Base64DecodeStringUsingCodepage

Syntax

```
Base64DecodeStringUsingCodepage(InputStr);
```

InputStr	Input string to be decoded.

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);
```

InputFile	Full path and name of input file.
OutputFile	Full path and name of output file with encoded data. If the file already exists it will be overwritten.

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.

1	Encoding succeeded.
0	Encoding failed.

Example

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

Base64EncodeString

Syntax

```
Base64EncodeString(InputStr);
```

InputStr	Input string to be base64 encoded.

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);
```

InputStr	Input string to be base64 encoded.
Codepage	The codepage to convert the input string to before encoding it to base64 format.

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)
```

num_1	Numeric expression.
num_2	Numeric expression.

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)`

num_1	Numeric expression.
num_2	Numeric expression.

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)`

num_1	Numeric expression.
num_2	Numeric expression.

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.

-1	Failed. The specified Process was not found.

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

```
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

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

Result:

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

ClearSubst

Syntax

```
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.

Returns

`1`	Table cleared

Example

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

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

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

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

ClearVars

Syntax

```
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

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

`strObjHandl`	The handle to be associated with the component.
`strProgId`	The ProgID of the component to be loaded.

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.

`0`	The component was successfully loaded.
`-1`	Failed to load the component.

Example

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

COMDestroyObject

Syntax

```
COMDestroyObject(strObjHandl);
```

strObjHandl	The handle associated with the component to be unloaded.

Description

Unloads the COM component associated with the specified handle.

Returns

A number indicating whether the component was successfully unloaded.

`0`	The component was successfully unloaded.
`-1`	Failed to unload component.

Example

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

COMInvoke

Syntax

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

strObjHandle	The handle associated with the component.
strMethodName	The method to be invoked.

var_1, var_2,...	One or more variables from StreamServe scripts to be used as arguments.
ReturnVar	The variable that will be set to the return value from the invoked method.

Description

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.

Returns

A number indicating whether the method was successfully invoked.

`0`	The method was successfully invoked.
num`<0`	Failed to invoke the method.

Example

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

COMSetProperty

Syntax

`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.

`0`	The property was successfully set.
num`<0`	Failed to set the property.

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.

`0`	The property was successfully retrieved.
num`<0`	Failed to retrieve the property.

Example

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

ConnectorPage

Syntax

```
ConnectorPage();
```

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

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

ConnectorPageActual

Syntax

```
ConnectorPageActual();
```

Description

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

Returns

num	The number of logical pages.
-1	Error.

Example

```
$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`*`);`

codepage	A string (case sensitive) that specifies the code page used to encode input data.
text_order	`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).

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);
```

key	A string uniquely identifying a serial number key, representing for example a document type.
initial	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.
min	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`.
max	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.

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 succesfully or not, or if it already exists..

`0`	The serial number key was created successfully.
`1`	The serial number key already exists in the repository,
`-1`	Error.

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

```
CurrDocProperty(propertyName);
```

propertyName	The document property to return. Can be one of the following: • `ppdocid` – the Post-processor document ID. • `ppjobid` – the Post-processor job ID. • `status` – the document status. • `priority` – the document priority. • `creationtime` – the document creation time. • `processtime` – the document process time. • `error` – the document error. • `pages` – the number of logical pages in the document. • `resrepository` – the resource repository. • `resserver` – the resource repository server.

Description

Returns the specified document property for the current document.

Returns

The specified document property.

Example

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

Result:

```
$docid = 123
```

CurrInFileName

Syntax

```
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

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

CurrInRealPath

Syntax

```
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

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

CurrInSavePath

Syntax

```
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);
```

propertyName	The job property to return. Can be one of the following: • `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 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);
```

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

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
arr[2]=7
```

CurrReposProperty

Syntax

```
CurrReposProperty(propertyName);
```

propertyName	The name of the repository property. Can be one of the following: • `server` – the FO server name. • `dbname` – the FO database name. • `physicalserver` – the physical server for the FO server. • `dblogicalname` – the database logical name.

Description

Returns the specified repository property.

Returns

The specified repository property.

Example

```
$db_server = CurrReposProperty("server");
```

D

DayOfWeek

Syntax

```
DayOfWeek([yy]yymmdd);
```

[*yy*]*yymmdd*	A date string.

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]);
```

metadataName	A string specifying the metadata name.
type	A string specifying one of the following types: `S` – String (default) `N` – Numeric
sort_order	A string specifying sorting order if metadata is used to sort documents. You can specify one of the following: `A` – Ascending (default) `D` – Descending
behaviour	A string specifying the result of the `DeclareMetadata` function if metadata is not found for the doucments. You can specify one of the following: `F` – Fail (default) `E` – Empty `D` – Default. See *default_value* argument.
defaultValue	Specifies the default value used when *behaviour* is set to default.

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

`0`	OK
num<>`0`	Error

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*`);`

num_jobID	The ID of the job in which the LXF job resource was created.
str_name	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.

num_index	The job resource index. If *num_index* is `-1`, all job resources within the specified job will be deleted.

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.

`>0`	OK. The job resources have been successfully marked and will be deleted.
`0`	Failure.

Example

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

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

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

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

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

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

DelSubstKey

Syntax

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

tbl_file	A string specifying the path and filename of the table.
str_key	The key of the entry to delete.

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.

Returns

A number indicating whether the entry was deleted successfully.

`1`	Entry deleted successfully.
`0`	Entry not deleted successfully.

Example

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

```
//!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`):

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

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

```
//!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

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

`str_dateformat`	A string specifying the date format.
`[`*`yy`*`]`*`yymmdd`*	A number specifying the a date, where: • `yyyy` – year in four digits, e.g. 2005. • `yy` – year in two digits, e.g. 05. • `mm` – month. • `dd` – day of the month

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 yyyy", 980923);
```

Result: `$d="mon 09 year 1998"`

DiffDate

Syntax

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

[yy]yymmdd	A date string.

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)
```

num_1	A numeric expression.
num_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

num	The number of pages inserted.
`-1`	Error

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

num	The number of pages sent to the output connector.
-1	Error

Example

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

Result:

```
$page_count = 5
```

DocPage

Syntax

```
DocPage();
```

Description

Returns the current page number of the current document.

Returns

num	The current page number.

Example

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

DocPages

Syntax

```
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

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

DtisoFormat

Syntax

```
DtisoFormat(source, format, timezone);
```

source	A string containing the date and time.

format	A string specifying the format of the source string.
	• For date only, a simple string is used, for example
	`"MM/DD/YYYY"` if source string is `"09/15/2007"`, or
	`"DD-MM-YY"` if source string is `"15-09-07"`
	• For date and time, the following format must be used:
	`"{date{<date>}{<internediate character>}time{<time>}}"`
	where
	• <*date*> is the date format of the source string and contains *YYYY* or *YY* for the year, *MM* for the month, and *DD* for the day.
	• <*time*> is the time format of the source string and contains *HH* for the hour and *MM* for the minute.
	• <*internediate character*> is the separator between the date and time, for example blank space.
	For example:
	`"{date{MM/DD/YYYY}{ }time{HH:MM}}"` if source string is `"09/15/2007 12:30"`

timezone	A string containing timezone (e.g. `"+01:00"` or `"-01:00"`). A return value including both date and time will be shifted according to the specified timezone. If you do not want to specify the timezone, you can specify an empty string,`""`. If you specify a timezone value for date conversion only, it will be ignored.

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:

```
$formatteddatetime = "2007-06-24T08:30:00"
```

DumpVariables

Syntax

```
DumpVariables(file_name);
```

file_name	A string specifying the name of the file to create. A file path can be specified. **Note:** If a file path you specify does not exist no file will be created.

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

1	OK

Example

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

E

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.

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.

0	Check is set.
num<>0	Error

Example

```
EnableEnvelopeCheck("ip", 0);
```

EndDocument

Syntax

```
EndDocument();
```

Description

Forces a Document Begin at the end of the Process from which the script function is called. The `EndDocument` function can be called from a Before or an After Process script, or from inside a Process. This function is not supported in Post-processor scripting.

Returns

N/A

Example

```
EndDocument();
```

EndMessage

Syntax

```
EndMessage();
```

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

`1`	Table removed

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.

```
$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);
```

Eval

Syntax

```
Eval(str);
```

str	A string containing one or several variables.

Description

Evaluates all variables in the argument string.

Returns

str	A string in which all variables have been replaced by their values.

Example

```
$question = "life, the universe, and everything";

$answer = "42";

$result = Eval ("The answer to the question of $question is
$answer");
```

`$result` will then contain

```
"The answer to the question of life, the universe, and everything
is 42"
```

Execute

Syntax

```
Execute(command, timeout, input, output);
```

command	The command to execute.
timeout	Timeout in seconds. StreamServer will wait the specified amount of time before it tries to shutdown the called process and returns to operation. Timeout set to -1 means StreamServer will "wait forever". This is the default value. 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".
input	Input data to send to the called process.
output	Array of three output variables: • `output[0]` – STDOUT output from the called process. • `output[1]` – STDERR output from the called process. • `output[2]` – Exit code from the called process.

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

0	OK.
1	Command not found.
2	Timeout reached.
3	Maximum number of spawned process reached.
4	Other error.

Examples

Example 54 Calling an external Java application to download document

```
$command = eval("java -cp <22>..\data<22> FileDownloader $link
$lPath");
$return = execute($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

1	True
0	False

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`).

F

FileClose

Syntax

```
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.

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

Example

```
//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

```
FileCopy(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

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

Returns

A number indicating whether the file was successfully copied.

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

Example

```
//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.

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

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.

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

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`*`);`

filename	A string specifying the path and name of the file.
mode	A string specifying the mode of opening: • `w` – Opens an empty file for writing. If the given file exists, its contents are destroyed. • `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. • `r` – Opens for reading. If the file does not exist or cannot be found, the call fails. • `b` – Open in binary mode; translations involving carriage-return and line-feed characters are suppressed.

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

```
while(filereadln(".\file.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

```
$size = FileSize("c:\my_file.txt");
```

FileWrite

Syntax

```
FileWrite(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);
}
```

FileWriteLn

Syntax

```
FileWriteLn(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, 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.

`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,
//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 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);
}
```

FindInArray

Syntax

```
FindInArray(array_name, str);
```

array_name	A string specifying the name of the array.
str	A variable or a string specifying the value to search for.

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

num	An integer specifying the position of the array element.
-1	Failure.

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.

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

FindStringExactInArray

Syntax

```
FindStringExactInArray(array_name, str);
```

array_name	A string specifying the name of the array.
str	A variable or a string specifying the value to search for.

Description

Searches for a value in an array. Returns the position of the array element that contains exactly the specified value.

Returns

num	An integer specifying the position of the array element.
-1	Failure.

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

`1`	True.
`0`	False.

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

`1`	True.
`0`	False.

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

`0`	OK. Action successful.
`-1`	Failed. Action NOT successful.

Example

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

ForcePoll

Syntax

`ForcePoll(`*`str_connectorName`*`);`

`str_connectorName`	The name of the input connector to activate.

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

`1`	OK. Action successful.
`0`	Failed. Action NOT successful.

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

```
GetAttachmentContentType(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 content-type of an attachment saved to disk via an EmailIN connector.

Returns

str	The content-type of the attachment

Example

```
$encoding = GetAttachmentContentType(1);
```

GetAttachmentCount

Syntax

```
GetAttachmentCount();
```

Description

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

Returns

num	The number of attachments in the current email that have been saved to disk by the EmailIN connector.

Example

```
$count = GetAttachmentCount();
```

GetAttachmentFile

Syntax

```
GetAttachmentFile(num_file);
```

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

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

```
$subject=GetConnectorValue("subject");
```

Result:

```
$subject="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

`0`	Current status OK (running)
`1`	Current status Warning
`2`	Current status Failed

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.

Returns

num	An index identifying the current document in the current output segment document array.
-1	Error.

Example

```
$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

```
$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 specifed by the picture clause.

Example

```
$a=getdatetime("");
```

Result:

```
$a="2009-04-03T11:03:30.456+01:00"
//
$b=getdatetime("MMMM 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);
```

role_name	The name of the role for which you want to retrieve the distribution channels.
doc_type	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.
directory	`0` = internal directory, `1` = external directory
channels_var	The variable that will contain the returned values. If you specify $<*channels_variable*>[0], all channels are returned (if your counter starts on `0`, see example below). If you specify $<*channels_variable*>, only the first channel is returned.

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

```
$channels[0] = "";
$cnt =
getDistributionChannelsForRole("administrator","invoice",0,$chann
els[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.

doc_type	The document type for which the user want to use the distribution channels. The document type must be defined in the Project and the StreamServer application started.
directory	0 = internal directory, 1 = external directory
channels_var	The variable that will contain the returned values. If you specify $<*channels_variable*>[0], all channels are returned (if your counter starts on 0, see example below). If you specify $<*channels_variable*>, only the first channel is returned.

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

```
$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

```
$inserts = GetDocInsertCount();
```

Result:

```
$inserts = 3
```

GetDocInsertEquivals

Syntax

```
GetDocInsertEquivals();
```

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

num	A number specifying the inserts equivalent count.
-1	Failure. Enveloping is not used or the **Count inserts** options is not selected.

Example

```
$equivals = GetDocInsertEquivals();
Result:
$equivals = 3.5
```

GetDocSheetCount

Syntax

```
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

num	Counter index identifying the first document in the set of documents.
-1	Error.

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

num	An index identifying the first document in the current output segment document array.
-1	Error.

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);
```

key	A string uniquely identifying a serial number key, representing for example a document type.

Description

Returns the next serial number available for the specified key.

Returns

num	A number specifying the next available serial number.
-1	Error.

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:

```
$ser = getglobalserno("invoices");
```

`$ser` is set to `1234` and the table entry contains:

```
invoices   1235   1000   2000
```

GetGlobalSerNoRange

Syntax

```
GetGlobalSerNoRange(key, range);
```

key	A string uniquely identifying a serial number key, representing for example a document type.
range	The range of serial numbers you want to reserve, starting from the current value.

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

num	The first value in the specified range.
`-1`	Error.

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:

```
$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

```
//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.

num<>0	The job has been completed successfully.
0	Failure.

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`*`);`

key	The name of the metadata attribute.

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`*`);`

key	A string, uniquely identifying the GUID or the name of the metadata (case sensitive). **Note:** If name is used, make sure the name is not updated in the Design Center configuration.

Description

Returns the current value of a metadata. The metadata must have the Message context.

Using the `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

```
$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);
```

doc_index	index that identifies the previous document in the set of Documents sent to the output connector.

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

num	A counter index identifying the next document.
0	There are no more Documents.
-1	Error

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);
```

doc_index	Index of the previous document in the current segment in the set of documents sent to the output connector.
num	An counter index identifying the next document.

Description

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

Returns

num	An counter index identifying the next document.
0	There are no more documents.
-1	Error.

Example

```
$doc_index = GetFirstSegDoc();
while(num($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

```
$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

num	The document ID.
-1	Error.

Example

```
$doc_id = GetPPDocId();
```

GetPPDocProperty

Syntax

```
GetPPDocProperty(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 document property. The *propertyName* can be: • `ppdocid` – the Post-processor document ID. • `ppjobid` – the Post-processor job ID. • `status` – the document status. • `priority` – the document priority. • `creationtime` – the document creation time. • `processtime` – the document process time. • `error` – the document error. • `pages` – the number of logical pages in the document. • `resrepository` – the resource repository. • `resserver` – the resource repository server.

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

num	Error code of the last error that occurred during post-processor job.
0	No error exists.
-1	The function did not succeed.

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();
```

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

num	The job ID.
-1	Error

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);
```

docHandle	An identifier specifying a document. You can retrieve the identifier by using either the `GetFirstPPDoc()` or the `GetNextPPDoc()` function.
metadataName	The metadata name connected to the current document

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.

propertyName	The name of the repository property. The *propertyName* can be: • `server` – the FO server name. • `dbname` – the FO database name. • `physicalserver` – the physical server for the FO server. • `dblogicalname` – the database logical name.

Description

Returns the value of the specified repository property storing the document specified by the *docHandle* parameter.

Returns

The value of *propertyName*.

Example

```
$doc_handle=GetFirstPPDoc();
$subject=GetPPReposProperty($doc_handle, "server");
```

GetPreviewType

Syntax

```
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

0	The job is not a preview job.
1	The job is a preview job, invoked from Ad Hoc Correspondence, Correspondence Reviewer, or Composition Center.
2	The job is a preview job for which one of the following applies: • The job is invoked from a Preview input connector. • The job is invoked from a connecter, containing an `strs-tm-preview` metadata (as HTTP header value).
3	The job is a preview job, invoked from a Design Center Project (for example, from a PageOUT Process).

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

```
$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);
```

docHandle	An identifier specifying a document. You can retrieve this identifier by using either the `GetFirstSegDoc()`, `GetCurrSegDoc()`, or `GetNextSegDoc()` function.
propertyName	The name of the document property. The *propertyName* can be: • `ppdocid` – the Post-processor document ID. • `ppjobid` – the Post-processor job ID. • `status` – the document status. • `priority` – the job priority. • `creationtime` – returns the time when the job was stored. • `processtime` – the time when the job was processed. • `error` – the job error code. • `pages` – the number of pages in the job. • `resrepository` – the resource repository name. • `strsid` – the resource server name.

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);
```

docHandle	An identifier specifying a document. You can retrieve this identifier by using either the `GetFirstSegDoc()`, `GetCurrSegDoc()`, or `GetNextSegDoc()` 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 time when the job was stored. • `processtime` – the time when the job was processed. • `error` – the job error code. • `pages` – the number of pages in job. • `name` – the job name. • `strsid` – the StreamServe job ID. • `user` – the user name. • `description` – the description of the job.

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.

propertyName	The name of the repository property. The *propertyName* can be: • `server` – the Fast objects server name. • `dbname` – the database name. • `physicalserver` – the physical server name. • `dblogicalname` – the database logical name.

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);
```

filename	A string specifying the path and filename of the file containing the number sequence.
str_key	A string specifying the key.

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`:

```
test0    0
test     10 50
test1    57 50 100
Result:      $c=11
```

Each line in the file can contain 2, 3, or 4 columns.

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);
```

tbl_file	A string specifying the path and filename of the substitution table.
str_key	A string specifying the key of the substitution table entry to search for.
num_col	A number specifying the value column. The first value column number is `0`.

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.

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

The following script returns the second value from the entry with key `agatha`:

```
$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

num	The number of value columns in the entry specified by *str_key*.
0	The key does not exist.

Examples

Example 59 *GetSubstColCount – examle 1.*

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 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.

```
//!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`:

```
$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

```
GetSubstKey(tbl_file, num_key_index);
```

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

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`*`);`

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

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`:

```
$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

```
GetTextFormattingInfo(base, lxf_data, width, height);
```

base	A string specifying the prefix of the returned variable names.
lxf_data	The LXF formatted data to be split into separate areas.
width	A number specifying the width of the first text area.
height	A number specifying the height of the first text area.

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

$*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

Example

```
$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" );
```

```
  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=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=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=getyoffs();
```

H

HexStr

Syntax

```
HexStr(str);
```

str	A string.

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"
```

(Where "..." are three converted numbers which may be non-printable.)

```
$b="xyz 09"
```

I

In2Mm

Syntax

```
In2Mm(num)
```

num	A number in inches to convert to millimeters.

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`*`);`

`counter_name`	A string or a variable.

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

`0`	OK
`1`	Error

Example

```
if (PreProc()=0 {
   $result = incProcStatCounter("department1");
}
```

InQueueName

`InQueueName` is replaced by *InConnectorName*.

InsertOverlay

Syntax

```
InsertOverlay(overlay_name, x_offset, y_offset);
```

overlay_name	The name of the overlay to use on the page.
x_offset	The horizontal distance of the overlay, calculated in points from the upper left corner of the page.
y_offset	The vertical position of the overlay, calculated in points from the upper left corner of the page.

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 the 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

0	The page overlay was successfully added.
num>0	Error.

Example

```
InsertOverlay("SummaryOverlay", 0, 5);
```

InsertPage

Syntax

```
InsertPage(overlay_name, strProcesslink);
```

overlay_name	The name of the overlay to use on the inserted page.
strProcesslink	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.

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

`0`	The page was successfully inserted.
num`>0`	Error

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

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

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

`0`	OK. Record added successfully.
`-1`	Failed. Record NOT added successfully.

Example

```
$return = isamadd("order", "1234       $45.50");
```

IsamClose

Syntax

```
IsamClose(isam_file);
```

`isam_file`	A string specifying the path and name of the Isam file.

Description

Closes the specified Isam file.

Note: All open Isam files are closed at Job End.

Returns

`0`	OK.
`-1`	Failed.

Example

```
isamclose("order");
```

IsamCreate

Syntax

```
IsamCreate(isam_file, num_reclen, key num_pos num_len);
```

`isam_file`	A string specifying the name of the Isam file, without the file extension.
`num_reclen`	A number specifying the record length.
`num_pos`	A number specifying the position of the key in the record (starting at `0`).
`num_len`	A number specifying the length of the key.

Description

Creates an Isam file with the specified record length and the specified key position and length.

Returns

`0`	OK
`-1`	Failed

Example

```
$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

`IsamErase(`*`isam_file`*`);`

`isam_file`	A string specifying the name of the Isam file, without the file extension.

Description

Removes the specified Isam file.

Returns

`0`	OK
`-1`	Failed

Example

```
isamerase("order");
```

IsamGet

Syntax

`IsamGet(`*`isam_file`*`, `*`rec_key`*`);`

`isam_file`	A string specifying the name of the Isam file, without the file extension.
`rec_key`	A string specifying the record key.

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

`0`	OK
`-1`	Failed

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.

decimal_sep	A string specifying the decimal separator.

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

`1`	True
`0`	False

Example

```
$var="123,123.45";
if(isamountvalue($var,",",".","."))
{
    $ok=1;
}
```

IsDate

Syntax

`IsDate(yyyymmdd);`

yyyymmdd	A date string.

Description

Checks if a string expression is a date string.

Returns

`1`	True
`0`	False

Example

```
$d="19990101";
if(isdate($d))
{
    $ok=1;
}
```

IsFirstDocInEnvelope

Syntax

```
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

`1`	True
`0`	False

Example

```
if(IsFirstDocInEnvelope())
{
    $ok=1;
}
```

IsFirstPageInEnvelope

Syntax

```
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

1	True
0	False

Example

```
if(IsFirstPageInEnvelope())
{
    $ok=1;
}
```

IsFirstSegment

Syntax

```
IsFirstSegment();
```

Description

Checks if the current segment is the first segment of post-processor job.

Returns

1	True
0	False

Example

```
if(IsFirstSegment())
{
   $ok = 1;
}
```

IsInsertingPage

Syntax

```
IsInsertingPage();
```

Description

Checks if the current page is being inserted to the output queue.

Returns

`1`	True
`0`	False

Example

```
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

`1`	True
`0`	False

Example

```
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

`1`	True
`0`	False

Example

```
if(IsLastPageInEnvelope())
{
    $ok=1;
}
```

IsLastSegment

Syntax

```
IsLastSegment();
```

Description

Checks if the current segment is the last segment of post-processor job.

Returns

`1`	True
`0`	False

Example

```
if(IsLastSegment())
{
   $ok = 1;
}
```

IsNum

Syntax

```
IsNum(str);
```

str	A string specifying a number.

Description

Checks if a string expression consists of digits.

Returns

`1`	True
`0`	False

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.

`0`	False. The current job is not a preview job.
`>0`	True. The current job is a preview job.

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

num	The number of inserted pages.
`-1`	Error

Example

```
$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

`1`	True
`0`	False

Example

```
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

```
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

`1`	True
`0`	False

Example

```
$lb=lastblockonpage();
if(num($lb)=1)
{
    $end=1;
}
```

LdapAddAttrBinaryValue

Syntax

`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

`1`	The value was successfully added to the attribute.
`-1`	Error. The value could not be added.

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`*`);`

`entryID`	The identifier of an entry.
`attrName`	The name of the attribute of interest.
`value`	The new value to be added to the attribute.

Description

Adds the specified value to the specified attribute. The changes are not written to the directory until the `LdapUpdateEntry` function is called.

Returns

`1`	The value was successfully added to the attribute. If the attribute does not already exist it is created.
`-1`	Error. The value could not be added.

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);
```

connName	A name you specify which will be used in all the following LDAP functions to identify which server they will access.
hostName	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.
user	The user ID to log on to the server.
password	The logon password to the server.

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

`1`	A connection to the server has been established successfully.
`-1`	Error. No connection to the server could be established.

Example

Loading a local file into a binary attribute:

```
$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

`LdapConnectPersistent(`*`connName`*`, `*`hostName`*`, `*`user`*`, `*`password`*`);`

`connName`	A name you specify which will be used in all the following LDAP functions to identify which server they will access.
`hostName`	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.
`user`	The user ID to log on to the server.
`password`	The logon password to the server.

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

`1`	A connection to the server has been established successfully.
`-1`	Error. No connection to the server could be established.

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

```
LdapConnectSSL(connName, hostName, user, password);
```

connName	A name you specify which will be used in all the following LDAP functions to identify which server they will access.
hostName	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.
user	The user ID to log on to the server.
password	The logon password to the server.

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

`1`	A connection to the server has been established successfully.
`-1`	Error. No connection to the server could be established.

Example

```
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);
```

connName	A name you specify which will be used in all the following LDAP functions to identify which server they will access.
hostName	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.
certNick	Nickname of the Client Certificate used to authenticate the user.
password	The logon password to the server.

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

`1`	A connection to the server has been established successfully.
`-1`	Error. No connection to the server could be established.

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);
```

connName	A name you specify which will be used in all the following LDAP functions to identify which server they will access.
hostName	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.
certNick	Nickname of the Client Certificate used to authenticate the user.
password	The logon password to the server.

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`	A connection to the server has been established successfully.
`-1`	Error. No connection to the server could be established.

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);
```

connName	A name you specify which will be used in all the following LDAP functions to identify which server they will access.
hostName	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.
user	The user ID to log on to the server.
password	The logon password to the server.

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:

`-ldapSslCertDb <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

`1`	A connection to the server has been established successfully.
`-1`	Error. No connection to the server could be established.

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);
```

connName	The identifier specified in *LdapConnect*.

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

`1`	OK
`-1`	Error. Could not disconnect.

Example

```
LdapDisconnect("con");
```

In this example, the connection `con` is disconnected from the directory server.

LdapFind

Syntax

`LdapFind(`*`connName`*`, `*`base`*`, `*`search`*`);`

`connName`	The identifier specified in the `LdapConnect` call.
`base`	The base entry from which to start searching for the specified user ID.
`search`	A search string in LDAP search syntax.

Description

Finds entries in the LDAP directory using an LDAP query.

Returns

`resultID`	The function call returns the `resultID` of the result set, i.e. the set of entries that match the search string. The `resultID` can be used in `LdapGetEntry` calls to retrieve the individual entries from the result set.
`""`	An empty string is returned if the search fails.

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

```
LdapGetAttrValue(entryID, attrName, valueIdx);
```

entryID	The identifier of an entry in a result set. The entry may contain several attributes.
attrName	The name of the attribute to be retrieved.
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

```
$name=ldapGetAttrValue($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

```
LdapGetAttrValueCount(entryID, attrName);
```

entryID	The ID of a specific entry.
attrName	The name of the attribute of interest.

Description

Returns the number of values of a specific attribute on an entry.

Returns

num	The number of values for the specific attribute.
-1	The attribute does not exist.

Example

```
$num=LdapGetAttrValueCount($theuser,"mail");
```

In this example, the number of values associated with the `mail` attribute of the `entryID` (`theuser`) is stored in the `num` variable.

LdapGetEntry

Syntax

```
LdapGetEntry(resultID, entryNumber);
```

resultID	The ID representing the result of a search.
entryNumber	The number of the entry in the result set to be retrieved. Note that indexing is 0 based, i.e. *entryNumber* 0 is the first entry in the result set, *entryNumber* 1 is the second entry, etc.

Description

Retrieves a specific entry from a result set (the result of a query).

Returns

entryID	The function call returns the `entryID` of the string found. The `entryID` can be used with subsequent calls to access its attributes.
""	An empty string is returned if the search fails.

Example

```
$theuser=ldapGetEntry($theuserlist, 0);
```

In this example, the value of the first entry (index 0) in `theuserlist` is assigned to `theuser` variable.

LdapGetEntryCount

Syntax

```
LdapGetEntryCount(resultID);
```

resultID	The ID represinting the result of a search.

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`*`);`

connName	The identifier specified in the `LdapConnect` call.
dn	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.

Description

Searches the LDAP directory for a specific object.

Returns

entryID	If the search is successful the `entryID` of the object found. The `entryID` can be used with subsequent calls to access entry attributes.
""	An empty string is returned if the search fails.

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);
```

connName	The identifier specified in the `LdapConnect` call.
base	The base entry from which to start searching for the specified user.
user	The user to find.
attribute	The ldap attribute used to identify user. Empty string defaults to `uid`.
objectClass	The ldap object class used to identify user. Empty string defaults to `person`.

Description

Searches the LDAP directory for a specific user.

Returns

entryID	If the search is successful an `entryID` is returned. The `entryID` can be used with subsequent calls to access the attributes of the entry.
""	An empty string is returned if the search fails.

Example

```
$user=ldapGetUser("con","cn=Users,dc=dev,dc=streamserve,dc=com",
"Administrator", "cn", "user" );
```

LdapNewEntry

Syntax

```
LdapNewEntry(connName, dn);
```

connName	The identifier specified in the `LdapConnect` call.

dn	The distinguished name of the new object (entry) to be created. For example, the distinguished name might consist of the objects `uid`, `ou`, and `o`. Note that the parameters must be given in exact order from the leaf entry to the root entry. `o` 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 group to which the object belongs, and the `uid` is the ID of the specific user or object. In this example, the `o` and `ou` objects must have been created prior to this function call. The new entry created will contain the unique attribute `uid` and become a leaf entry in the directory tree.

Description

Creates a new entry in the LDAP directory. Changes are not written to the directory until the `LdapUpdateEntry` function is called.

Returns

entryID	An `entryID` is returned if the new entry was created successfully. The `entryID` can be used with subsequent calls to access entry attributes.
""	An empty string is returned if the search fails.

Example

```
$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

`1`	The result set was released successfully.
`-1`	Error: The result set could not be released.

Example

```
$myresults = ldapNewEntry($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

`1`	The result set was released succesfully
`-1`	Error: The result set could not be released

Example

```
$myresults=ldapFind($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

`1`	The values were replaced successfully.
`-1`	Error. The value could not be replaced.

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);
```

`entryID`	The identifier of an entry.
`attrName`	The name of the attribute whose set of values is to be replaced.
`value`	The new value to replace the existing value(s).

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

`1`	The values were replaced successfully. If the attribute does not already exist it is created.
`-1`	Error. The value could not be replaced.

Example

```
ldapReplaceAttrValue($theuser,"description",$insert);
```

In this example, the value of the `description` attribute in `theuser` 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

`LdapSort(`*`resultId`*`, `*`sortType`*`, `*`attrName_1`*`[,`*`attrName_2...attrNameN`*`]);`

resultId	The result set id returned from LdapFind
sortType	A string of maximum three characters that specifies the sort type to be performed. You must specify one of the following: `N` – Numeric sort `S` – String sort `C` – String collation You can specify the sort order by adding one of the following characters to the string: `A` – Ascending (Default) `D` – Descending If you specify sort type `S` you can add the following to the string: I – Case insensitive
attrNameN	The attribute to compare.

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

`1`	The result set was sorted successfully.
`-1`	Error. The result set was not sorted successfully.

Example

```
// Sort the search result by string (case insensitive) in ascending
// order using the "common name" ("cn") attribute.
ldapsort($myResultId, "SAI", "cn");
```

LdapUpdateEntry

Syntax

```
LdapUpdateEntry(entryID);
```

entryID	The identifier of an entry.

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

`1`	The changes made to the entry were successfully written back to the LDAP server.
`-1`	Error. The entry could not be updated n the LDAP server.

Example

```
ldapUpdateEntry($theuser);
```

In this example, all changes made to `theuser` variable and its attributes are written back to the LDAP server.

LdapWriteAttrBinaryValue

Syntax

```
LdapWriteAttrBinaryValue(entryID, attrName, fileName);
```

entryID	The identifier of an entry.
attrName	The name of the attribute of interest.
fileName	A filename, with optional path.

Description

Writes the contents of the attribute to a file.

Returns

`1`	The value was successfully added to the attribute.
`-1`	Error. The value could not be written to a file.

Example

```
ldapWriteAttrBinaryValue($theuser,"photo",my_file);
```

In this example, the contents of the `photo` attribute of `theuser` entry are written to `my_file`.

Log

Syntax

`Log(num_loglevel, str);`

num_loglevel	Log level that specifies when to append a string to the log file. • `0` corresponds to log level `Severe error messages` in the Platform. • `1` corresponds to log level `All error messages` in the Platform. • `2-3` corresponds to log level `All error and warning messages` in the Platform. • `4-8` corresponds to log level `All error, warning and information messages` in the Platform. • `9` corresponds to log level `All error, warning and extended information messages` in the Platform. 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

`0`	OK
`-1`	Failed

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

`0`	The Lotus Notes Note was successfully created in the Lotus Notes database.
`-1`	Failed to add the Lotus Notes Note to the Lotus Notes database. Possible causes include no connection to the database.

Example

```
$ret = LotusNotesAddNote();
```

LotusNotesAttachFile

Syntax

```
LotusNotesAttachFile(str_path, str_compress);
```

`str_path`	The file to be attached. You must specify the full path.
`str_compress`	A boolean option for file compression. If the value is `true` (not case sensitive), the file will be compressed in the Lotus Notes Note.

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

`0`	The file was successfully attached to the Lotus Notes Note.
`-1`	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.

Example

```
$ret = LotusNotesAttachFile("C:\Files\My_file.pdf", "True");
```

LotusNotesConnect

Syntax

```
LotusNotesConnect(str_server, str_database, str_password);
```

str_server	The Lotus Domino server.
str_database	The filename of the Lotus Notes database on the Lotus Domino server. 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. 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. Only a Lotus Domino administrator can enable read or write access to the Lotus Notes database.
str_password	The password for unlocking the Lotus ID file which authenticates access to a Lotus Notes database. 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. If you connect to the database via a Lotus Domino server, without access to a Lotus Notes client installation, you must: • Obtain a suitable Lotus ID file. • Ensure that the Lotus ID file is recertified and contains a valid password. • Rename the Lotus ID file to `ssdominouser.id` • Store the Lotus ID file in the export directory of the Project. Ask your Lotus Domino administrator for assistance.

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

`0`	A connection was successfully established.
`-1`	Failed to establish a connection to the Lotus Notes database.

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

`0`	The connection was successfully disconnected.
`-1`	Failed to disconnect.

Example

```
$ret = LotusNotesDisconnect();
```

LotusNotesFind

Syntax

`LotusNotesFind(`*`str_search`*`);`

`str_search`	The search criteria. The criteria must be given in Lotus Notes syntax.

Description

Searches the database for Lotus Notes Notes that match the specified search criteria.

Returns

n	The number of Lotus Notes Notes found that match the search criteria.
-1	Invalid search syntax.
-2	The search could not be performed. Possible causes include no connection to the database.

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

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 Note.

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

```
$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

```
$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

`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

```
$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

`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

```
$ret = LotusNotesPrev();
```

LotusNotesSetDateTime

Syntax

`LotusNotesSetDateTime(`*`str_fieldname`*`, `*`str_date`*`);`

`str_fieldname`	The item to be updated or added.
`str_date`	The date and time with which to update the field.

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)
- `DDMMYYYY[HHMMSS]`
- `MMDDYYYY[HHMMSS]`

Returns

`0`	The Note was successfully updated.
`-1`	Failed to update the Note. Possible causes include no connection to the database.

Example

```
$ret = LotusNotesSetDateTime("created", "20001231101555");
```

LotusNotesSetNumber

Syntax

`LotusNotesSetNumber(`*`str_fieldname`*`, `*`str_value`*`);`

`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

`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 = LotusNotesSetNumber("invoiceNumber", "123456");
```

LotusNotesSetText

Syntax

```
LotusNotesSetText(str_fieldname, str_value);
```

str_fieldname	The item to be updated or added.
str_value	The text to use a data for the item.

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");
```

M

Mat

Syntax

```
Mat(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. Removes leading and trailing blanks. Only applicable to PageIN.

Returns

String copied from the matrix.

Example

```
mat(4, 7, 25);
```

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`*`);`

num_pos	Number specifying the position from which the length of data will be returned.
num_length	Number specifying the length of the data to be returned.

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);
```

path	A path.

Description

Creates a new directory using the specified path.

Returns

0	OK
-1	Failed

Example

```
mkdir(C:\StreamServe\Company_ABC);
```

Mm2In

Syntax

```
Mm2In(num)
```

num	A number in millimeters to convert to inches.

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)
```

num	A number in millimeters to convert to points.

Description

Converts a number given in millimeters to points unit.

Returns

A number in points unit as a floating point value.

Example

```
$mm = 100;
$points = mm2pt($mm);
```

Mod

Syntax

`Mod(num_1, num_2)`

num_1	A number being divided.
num_2	A number being the divisor.

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

`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

`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`*`)`

str	A string specifying a number.

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:

```
$modlending is "23"
```

```
$newnoterefnr is "123456789023"
```

Mod11

Syntax

`Mod11(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

```
$total = "11234567";
$chk = mod11($total);
```

MoveFile

`MoveFile` is an old name for the *FileMove* function.

MsgBlockId

Syntax

`MsgBlockId(travhandl);`

travhandl	The handle used to identify this specific traversing pass. The handle is specified using the *MsgOpen*, *MsgFrameOpen*, or *MsgFrameOpen* function.

Description

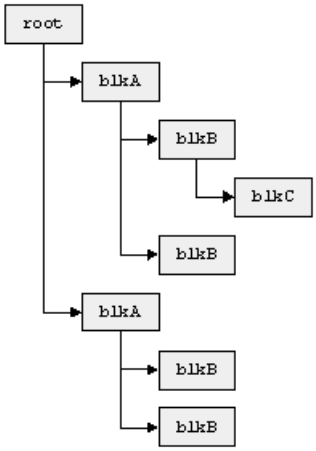
Returns the name of the current block.

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.



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

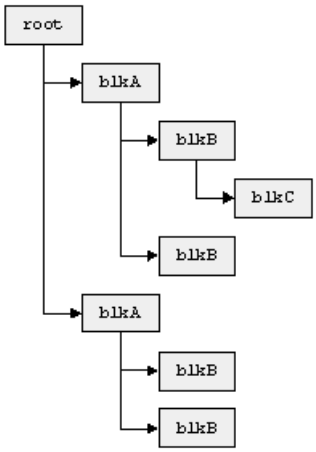
Counts the number of instances of the specified block in the Message.

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`:

```
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`:

```
...
$counter = 1;
$blkBinA = MsgCountId("blkA:" + $counter + "/blkB");
$counter++;
...
```

MsgFieldId

Syntax

`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

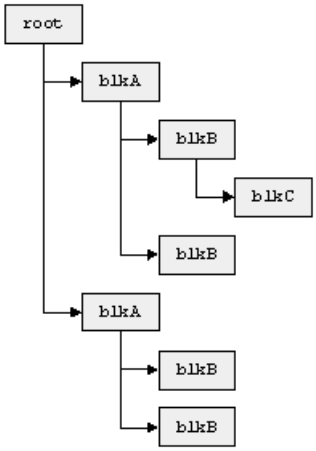
Retrieves the name of a field (not field value) in the current block.

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.



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

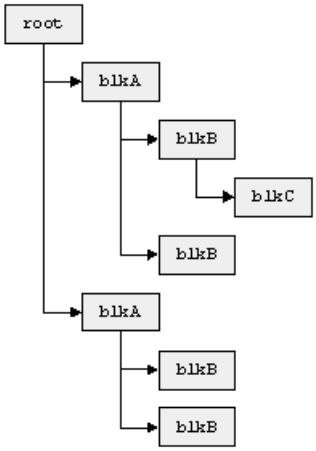
Returns the value of a field in the current block.

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`:

```
$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`:

```
$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`:

```
$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

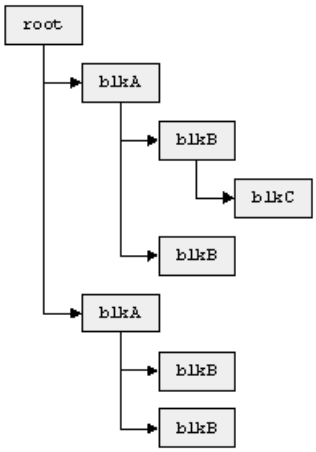
Counts the number of instances of the specified block in the current frame.

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;
$blkBinA = MsgFrameCountId("blkA:" + $counter + "/blkB");
$counter++;
...
```

MsgFrameGetValue

Syntax

`MsgFrameGetValue(`*`field`*`);`

field	A string specifying the name of a field in the current frame.

Description

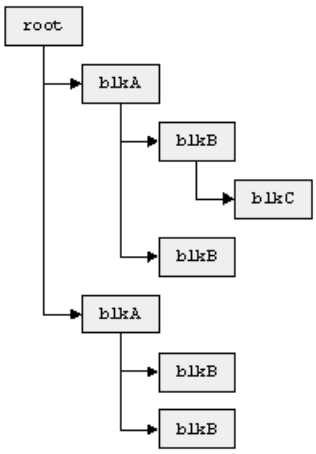
Retrieves the value of a specified field in the current frame.

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 frame contains the blocks `blkA`, `blkB`, and `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 `idType` in the second instance of block `blkA`:

```
$fieldValue=MsgFrameGetValue("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`:

```
$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`:

```
...
$counter = 1;
$fieldValue=MsgFrameGetValue("blkA:" + $counter +"/blkB:2/
idType");
$counter++;
...
```

MsgFrameOpen

Syntax

`MsgFrameOpen(`*`str_block_address`*`, `*`num_traverse_level`*`);`

`str_block_address`	A string specifying the block where to start traversing the Message.
	Empty string
	If you specify an empty string (““), the Message traversing will start at the current location.
	Wildcards
	The following three examples illustrate how you can use wildcards in the block address string.
	Match the first block in the frame:
	`*`
	Match the first child block of the block `blkA`:
	`blkA/*`
	Match the 5:th instance of the block `blkA`:
	`blkA:5`

num_traverse_level	A number specifying the number of levels to traverse, starting from the specified block: 0 - Traverse the entire subtree. 1 - Include blocks at the same level as specified block. 2 - Same as 1, plus child blocks. 3 - Same as 2, plus child blocks of child blocks. etc.

Description

Defines where in the current frame to start traversing the Message. This function can only be used in PageOUT.

Returns

travhandl	A handle is returned if the frame was successfully traversed. The handle identifies this specific traversing pass.
-1	Failed to perform the traversing pass. Possible reasons include no block found, or that the function was called in the wrong context.

Example

```
$handle=MsgFrameOpen("blkA/blkB", 0);
```

MsgFrameValueExist

Syntax

```
MsgFrameValueExist(field);
```

field	A string specifying the name of a field in the current frame.

Description

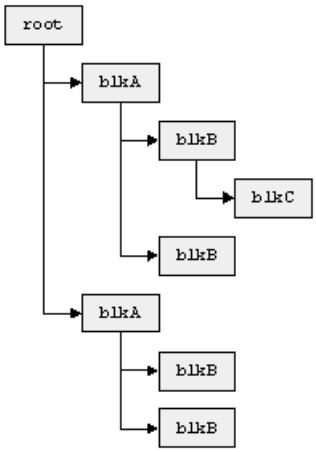
Checks if the specified field contains a value.

Returns

`1`	The field value exists.
`0`	The field value does not exist.

Example

In the following examples, the frame contains the blocks `blkA`, `blkB`, and `blkC` structured as shown in the figure below.



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);
```

field	A string specifying the name of a field.

Description

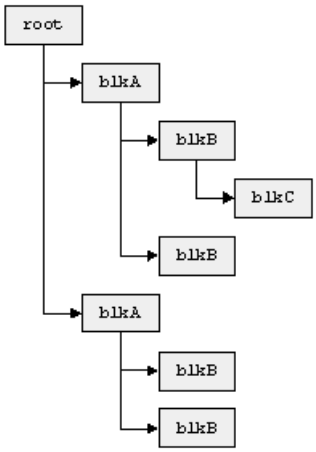
Retrieves the value of a specified field.

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 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

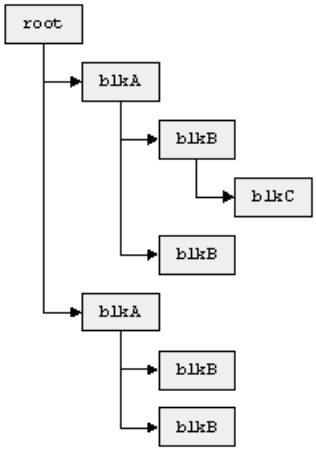
Moves to the next block.

Returns

`1`	Successfully moved to the next block.
`0`	Failed to move to the next block. Possible reasons include end of Message or invalid handle.

Example

In this example, the Message contains the blocks `blkA`, `blkB`, and `blkC` structured as shown in the figure below.



The following Before Frame script in a PageOUT Process returns the name of the first instance of block `blkB` in the first instance of block `blkA`. Then it moves to the next block in the Message tree, and returns the name of this block (`blkC`):

```
$handle=MsgFrameOpen("blkA:1/blkB:1", 0);
$blockID1=MsgBlockId($handle);
MsgNextBlock($handle);
$blockID2=MsgBlockId($handle);
```

MsgNumFields

Syntax

`MsgNumFields(`*`travhandl`*`);`

`travhandl`	The handle used to identify this specific traversing pass. The handle is specified using the *MsgOpen*, *MsgFrameOpen*, or *MsgFrameOpen* function.

Description

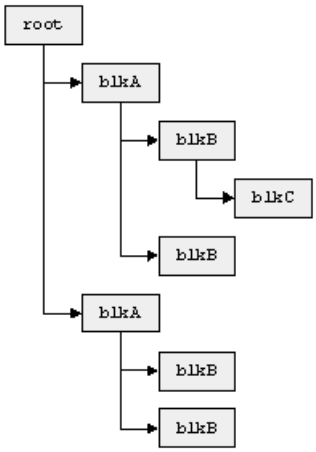
Returns the number of fields in the current block.

Returns

num_fields	The number of fields in the current block.

Example

In this example, the Message contains the blocks `blkA`, `blkB`, and `blkC` structured as shown in the figure below.



The following Before Frame script in a PageOUT Process returns the number of fields in the first instance of block `blkB` in the first instance of block `blkA`:

```
$handle=MsgFrameOpen("blkA:1/blkB:1", 0);
$numFields=MsgNumFields($handle);
```

MsgOpen

Syntax

```
MsgOpen(str_block_address, num_traverse_level);
```

str_block_address	A string specifying the block where to start traversing the Message.
	Empty string
	If you specify an empty string (““), the Message traversing will start at the current location.
	Wildcards
	The following three examples illustrate how you can use wildcards in the block address string.
	Match the first block in the Message:
	`*`
	Match the first child block of the block `blkA`:
	`blkA/*`
	Match the 5:th instance of the block `blkA`:
	`blkA:5`

num_traverse_level	A number specifying the number of levels to traverse, starting from the specified block: 0 - Traverse the entire subtree. 1 - Include blocks at the same level as specified block. 2 - Same as 1, plus child blocks. 3 - Same as 2, plus child blocks of child blocks. etc.

Description

Defines where to start traversing the Message.

Returns

travhandl	A handle is returned if the frame was successfully traversed. The handle identifies this specific traversing pass.
-1	Failed to perform the traversing pass. Possible reasons include no block found, or that the function was called in the wrong context.

Example

```
$handle=MsgOpen("blkA/blkB", 0);
```

MsgProcCountId

Syntax

```
MsgProcCountId(block);
```

block	A string specifying a block in the current Process.

Description

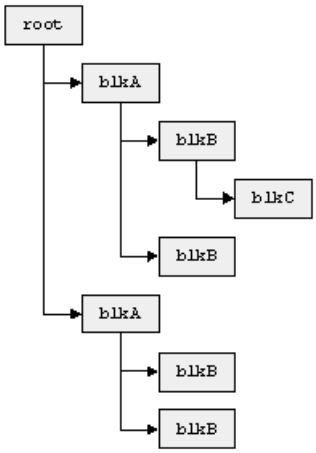
Counts the number of instances of the specified block in the current Process.

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.



Example 79 Script returning the number of blkB blocks.

The following script returns the number of `blkB` blocks in the Process:

```
MsgProcCountId("blkB"); //Returns 4
```

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`:

```
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`:

```
...
$counter = 1;
$blkBinA = MsgProcCountId("blkA:" + $counter + "/blkB");
$counter++;
...
```

MsgProcGetValue

Syntax

`MsgProcGetValue(`*`field`*`);`

field	A string specifying the name of a field in the current Process.

Description

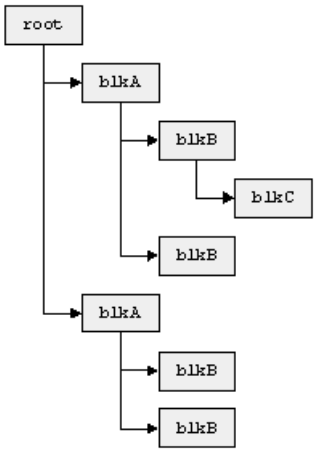
Retrieves the value of a specified field in the current Process.

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);
```

str_block_address	A string specifying the block where to start traversing the Message.
	Empty string
	If you specify an empty string (““), the Message traversing will start at the current location.
	Wildcards
	The following three examples illustrate how you can use wildcards in the block address string.
	Match the first block in the Process:
	`*`
	Match the first child block of the block `blkA`:
	`blkA/*`
	Match the 5:th instance of the block `blkA`:
	`blkA:5`

num_traverse_level	A number specifying the number of levels to traverse, starting from the specified block: 0 - Traverse the entire subtree. 1 - Include blocks at the same level as specified block. 2 - Same as 1, plus child blocks. 3 - Same as 2, plus child blocks of child blocks. etc.

Description

Defines where in the current Process to start traversing the Message. This function cannot be used in PageOUT.

Returns

travhandl	A handle is returned if the frame was successfully traversed. The handle identifies this specific traversing pass.
-1	Failed to perform the traversing pass. Possible reasons include no block found, or that the function was called in the wrong context.

Example

```
$handle=MsgProcOpen("blkA/blkB", 0);
```

MsgProcValueExist

Syntax

```
MsgProcValueExist(field);
```

field	A string specifying the name of a field in the current Process.

Description

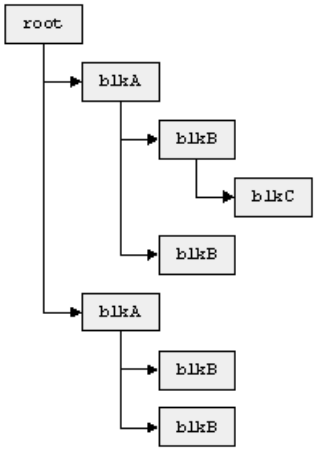
Checks if the specified field contains a value.

Returns

1	The field value exists.
0	The field value does not exist.

Example

In the following examples, the Process contains the blocks `blkA`, `blkB`, and `blkC` structured as shown in the figure below.



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

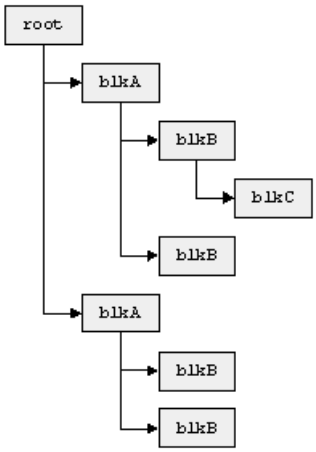
Checks if the specified field contains a value.

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:

```
$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:

```
$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:" + $counter +"/blkB:2/idType");
$counter++;
```

...

N

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

`0`	The document was successfully skipped.
num`<>0`	Error

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

`0`	The page was successfully skipped.
num`<>0`	Error

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

`0`	The Process was successfully skipped.
num<>0	Error

Example

```
NextProc();
```

NextSegment

Syntax

```
NextSegment();
```

Description

Skips the rest of the current segment, and processes the next segment if it exists.

Returns

`0`	The segment was successfully skipped.
num<>0	Error

Example

```
NextSegment();
```

NewDate

Syntax

```
NewDate([yy]yymmdd, num_days);
```

[*yy*]*yymmdd*	A number specifying a date.
num_days	A number specifying a negative or positive number of days.

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

```
$a="980427";
$b=newdate($a,15);
$c=newdate($a,-10);
```

Result:

```
$b="19980512"
$c="19980417"
```

NewJob

Syntax

```
NewJob();
```

Description

The `NewJob` function replaces the `JobBegin` and `JobEnd` functions, and is used to split large input jobs into smaller jobs.

The `NewJob` function must be called from a Retrieved script. The `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($count)>500)
{
    NewJob();
}
$count++;
```

NewPage

Syntax

```
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

```
if($var="end")
{
    newpage();
}
```

NFormat

Syntax

```
NFormat(syntax, num);
```

syntax	A string specifying the format syntax. Z – A digit displayed if required. If the value of a leading or trailing digit is zero, the digit is replaced by a space, " ". B – A digit displayed if required. If the value of a leading or trailing digit is zero, the digit is removed. 9 – A digit that is always displayed. # – A digit on either side of a decimal separator, or the last sign if no decimal separator exists. The digit is always displayed. Other characters, including spaces, remain unchanged. **Note:** Z and B 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

```
$a=1234.5;
$b=nformat("Z,ZZ#.##",$a);
$c=nformat("9999#.##",$a);
$d=nformat("ZZ.ZZ",0);
$e=nformat("ZZ.ZZ",0.07);
```

Result:

```
$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

`0`	OK
`-1`	Failed

Example

```
if(num($var)<20)
{
    noformfeed();
}
```

Num

Syntax

`Num(str);`

str	A string specifying a number.

Description

Converts a string expression to a numerical value.

Returns

A number.

Example

```
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 Post-processor 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:

```
proc1, proc n/2, proc2, procn/2+1, etc
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

```
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

```
$eventnumber = numberOfEventsEx("invoice");
```

O

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

`1`	The transaction was successfully saved to the database.
`-1`	Error. Problems occurred while trying to save the transaction to the database.

Example

```
ODBCCommitTrans("my_identifier");
```

OdbcConnect

Syntax

```
OdbcConnect(str_dbidentifier, str_data_source, str_userID,
str_password);
```

str_dbidentifier	A name you specify which will be used in all the following ODBC functions to identify which data source they will access.
str_data_source	The name of the ODBC data source to which you want to establish a connection.
str_userID	The user ID to login to the data source.
str_password	The login password to the data source.

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

`1`	A connection to the data source has been established successfully.
`-1`	Error. No connection to the data source could be established.

Example

```
OdbcConnect("my_identifier","address.dsn","sa","abc_1");
```

OdbcConnectW

Syntax

```
OdbcConnectW(str_dbidentifier, str_data_source, str_userID,
str_password, str_codepage);
```

str_dbidentifier	A name you specify which will be used in all the following ODBC functions to identify which data source they will access.
str_data_source	The name of the ODBC data source to which you want to establish a connection.
str_userID	The user ID to login to the data source.
str_password	The login password to the data source.
str_codepage	The code page corresponding to the retrieved data.

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

`1`	A connection to the data source has been established successfully.
`-1`	Error. No connection to the data source could be established.

Example

```
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 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

1	All open queries have been closed successfully.
-1	No open query exists.

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

`1`	The server has been disconnected from the data source.
`-1`	Error. The server could not be disconnected from the data source.

Example

```
OdbcDisconnect("my_identifier");
```

OdbcExecute

Syntax

`OdbcExecute(`*`str_dbidentifier, str_sql_statement`*`);`

`str_dbidentifier`	The identifier specified in `OdbcConnect`.
`str_sql_statement`	The SQL statement to execute.

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.

Se also *OdbcExecuteEx*.

Returns

`1`	The statement was executed successfully.
`-1`	Error. An error occurred while trying to execute the statement.

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);
```

str_dbidentifier	The identifier specified in `OdbcConnect`.
str_sql_statement	The SQL statement to execute.
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]);
```

str_dbidentifier	The identifier specified in `OdbcConnect`.
str_sql_statement	The SQL statement to execute.
var_n	The variables that will store the data.

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

`1`	The statement was executed successfully.
`0`	No data found. The statement did not return any result set. The variables are cleared.
`-1`	Error. An error occurred while trying to execute the statement.

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`*`]);`

`str_dbidentifier`	The identifier specified in `OdbcConnect`.
`str_sql_statement`	The SQL statement to execute.
`var_n`	The variables that will store the data.

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

`1`	The statement was executed successfully.
`0`	No data found. The statement did not return a result set. The variables are cleared.
`-1`	Error. An error occurred while trying to execute the statement.

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`*`]);`

`str_dbidentifier`	The identifier specified in `OdbcConnect`.
`var_n`	The variables that will store the data.

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.

Returns

`1`	The statement was executed successfully.
`0`	No data found. The statement did not return any result set. The variables are cleared.
`-1`	Error. An error occurred while trying to execute the statement.

Example

```
$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

`ODBCRollbackTrans(`*`str_dbidentifier`*`);`

`str_dbidentifier`	The identifier specified in `OdbcConnect`.

Description

Rolls back and ends a transaction that you have opened with `ODBCBeginTrans`.

Returns

`1`	The transaction was successfully rolled back.
`-1`	Error. Problems occurred while trying to roll back the transaction.

Example

```
ODBCRollbackTrans("my_identifier");
```

OdbcSetCodepage

Syntax

```
OdbcSetCodepage(str_dbidentifier, str_codepage);
```

str_dbidentifier	A name you specify which will be used in all the following ODBC functions to identify which data source they will access.
str_codepage	The code page corresponding to the retrieved data.

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

```
OdbcSetCodepage("my_identifier", "ISO 8859-2");
```

OdbcSetReorderText

`OdbcSetReorderText` is obsolete.

OutputFile

Syntax

```
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

`1`	The contents were successfully sent to the connector.
`0`	An error occurred. This may result if the file cannot be opened or read.

Example

Assume that you have an XMLOUT Process that produces the following output data:

```
<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:
<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:

```
OutputFile ("fruits.txt");
```

The output sent to the connector will be:

```
<html>
<head>Tender</head>
<body>
<h1>Vitamins Inc. kindly offers the following products</h1>
<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

```
OutputLXFJobResource(num_jobID, str_name, num_index);
```

num_jobID	The ID of the job in which the LXF job resource was created.
str_name	The name of the job resource that contains the LXF overlay you want to add to the PageOUT Process.
num_index	The job resource index.

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

num>0	The LXF overlay has been successfully added to the Process.
`0`	Failure.

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

```
OutputLXFJobResourcePrnOffs(num_jobID, str_name, num_index);
```

num_jobID	The ID of the job in which the LXF job resource was created.
str_name	The name of the job resource that contains the LXF overlay you want to add to the PageOUT Process.
num_index	The job resource index.

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

num>0	The LXF overlay has been successfully added to the Process.
0	Failure.

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

`1`	String was written successfully.
`0`	String was not written successfully.

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:

```
outputstring($fieldvariable);
```

The `$fieldvariable` variable is assigned the value of a field which is displayed as text in the XML prologue.

OverlayGetNumPages

Syntax

```
OverlayGetNumPages(filename);
```

`filename`	Path and name of the overlay file.

Description

Returns the total number of pages in a multi-page overlay.

Returns

`num`	The number of pages.
`0`	Error. The number of pages could not be retrieved.

Example

```
$totnum=OverlayGetNumPages("MultiPageOverlay.tiff");
```

P

Page

Syntax

```
Page();
```

Description

Returns the number of the current page.

Returns

num	The number of the current page.

Example

```
$pg=page();
if(num($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

```
$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

num	The number of pages in the Process.

Example

```
$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

num	The number of Documents sent to the output connector.
-1	Error

Example

```
$doc_count = PPDocCount();
```

PreProc

Syntax

```
PreProc();
```

Description

Determines whether or not StreamServer is running in pre-process phase.

Returns

1	True
0	False

Example

```
//!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);
```

num_loglevel	Log level that specifies when to append a string to the log file. • `0` corresponds to log level `Severe error messages` in the Platform. • `1` corresponds to log level `All error messages` in the Platform. • `2-3` corresponds to log level `All error and warning messages` in the Platform. • `4-8` corresponds to log level `All error, warning and information messages` in the Platform. • `9` corresponds to log level `All error, warning and extended information messages` in the Platform. 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

`0`	OK
`-1`	Failed

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*.

R

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

0	Not re-read
1	Re-read
-1	Error (the path could not be found)

Example

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

```
//!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.

```
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`*`);`

red	A number specifying the level of red color (0-255).
green	A number specifying the level of green color (0-255).
blue	A number specifying the level of blue color (0-255).

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`*`);`

num	A number to be rounded.
decimals	A number of decimals.

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
```

S

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

`0`	Metadata is saved
`num``<>0`	Error

Example

```
$ret = SaveCurrMetadata("zip");
```

SavePPMetadata

Syntax

```
SavePPMetadata(doc_handle, metadata_name);
```

doc_handle	Identifies the document.
metadata_name	A string specifying the name of the metadata associated with the document specified in the *doc_handle* parameter.

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

`0`	The metadata is saved
`num<>0`	Error

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);
```

doc_handle	Identifies the document.
metadata_name	A string specifying the name of the metadata associated with the specified document.

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

`0`	The metadata is saved.
`num<>0`	Error

Example

```
$doc_id = GetFirstSegDoc();
while(num($doc_id) > 0)
{
   $ret = SetSegMetadata($doc_id, "city", "Prague");
   $ret = SaveSegMetadata($doc_id, "city");
   $doc_id = GetNextSegDoc($doc_id);
}
```

SessionEnd

Syntax

`SessionEnd(`*`str_session_name`*`);`

`str_session_name`	A string specifying the session name.

Description

Ends a session in the StreamServer, and removes all associated variables.

Returns

`1`	OK
`0`	Failed

Example

Ends a session called `My_Session`:

```
sessionend("My_Session");
```

SessionGetVariable

Syntax

`SessionGetVariable(`*`str_session_name`*`, `*`str_var_name`*`);`

`str_session_name`	A string specifying the session name.

str_var_name	A string specifying the variable name.

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`*`);`

str_session_name	A string specifying the name of the session.
str_var_name	A string specifying the variable name.
str_var_value	A string specifying the variable value.

Description

Specifies a value for a given variable in a given session. If the variable does not exist, it will be created.

Returns

`1`	OK
`0`	Failed

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);
```

str_session_name	A string specifying the name of the session.
num_timeout	A positive integer specifying the timeout length (minutes).

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

`1`	OK
`0`	Failed

Example

Starts a session called `My_Session`, and specifies the timeout to 25 minutes:

```
sessionstart("My_Session", 25);
```

SetCopies

Syntax

```
SetCopies(num);
```

num	A number specifying the number of copies.

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

`0`	OK
`-1`	Failed

Example

```
setcopies(5);
```

SetCurrMetadata

Syntax

```
SetCurrMetadata(metadataName, value);
```

metadataName	A string specifying the metadata name for the current document.
str	A string specifying the value of the metadata to set for the current document.

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

`0`	The metadata value is set.
num<>`0`	Error.

Example

```
$ret = SetCurrMetadata("zip", "Prague");
```

SetCurrMetadataNum

Syntax

```
SetCurrMetadataNum(metadataName, value);
```

metadataName	A string specifying the metadata name for the current document.
num	A number specifying the value of the metadata to set for the current document.

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

`0`	The metadata value is set.
num<>`0`	Error.

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]]);
```

path	One or more paths, separated by semicolon, that replace a % sign in a file specification.
str	One or more strings that replace %1 etc in the file specification string resulting after the % path substitution above. **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 q1_invoices
setextjobid ($q1_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

```
$font=SetFontProperties($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

```
SetJobDescr(str_descr);
```

str_descr	The description of the current job.

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

```
setjobdescr("inv_job");
```

SetJobFailed

Syntax

```
SetJobFailed();
```

Description

Specifies the current job status as failed.

Returns

N/A

Example

```
setjobfailed();
```

SetJobOwner

Syntax

```
SetJobOwner(owner);
```

owner	A string specifying the job owner.

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);
```

lang	A string specifying a language.

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`*`);`

key	The name of the metadata attribute.
value	The value to which to set the metadata attribute for this job.

Description

Creates a new metadata attribute for the input job.

Returns

1	The metadata attribute was successfully added.
-1	An error occurred.

Example

```
$result=SetMetaDataJob("type", "print");
```

SetPPMetadata

Syntax

`SetPPMetadata(`*`doc_handle, metadata_name, value`*`);`

doc_handle	Identifies a document
metadata_name	A string specifying the metadata name associated with the current document.
value	A string specifying the value of the metadata associated with the current document.

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

`0`	The metadata value is set.
num<>`0`	Error.

Example

```
$doc_id = GetFirstPPDoc();
while(num($doc_id) > 0)
{
   $ret = SetPPMetaData ($doc_id, "city", "Prague");
   $doc_id = GetNextPPDoc($doc_id);
}
```

SetPPMetadataNum

Syntax

`SetPPMetadataNum(`*doc_handle, metadataName, value*`);`

doc_handle	Identifies a document
metadata_name	A string specifying the metadata name associated with the current document.
value	A number specifying the value of the metadata associated with the current document.

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

`0`	The metadata value is set.
num<>`0`	Error

Example

```
$doc_id = GetFirstPPDoc();
while(num($doc_id) > 0)
{
   $ret = SetPPMetaDataNum ($doc_id, "invoice_no", 112233);
   $doc_id = GetNextPPDoc($doc_id);
}
```

SetPrnXoffs

Syntax

`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

```
if($doctype="invoice")
{
    setprnxoffs(4);
}
```

SetPrnYoffs

Syntax

`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");
```

```
   $doc_id = GetNextSegDoc($doc_id);
}
```

SetSegMetadataNum

Syntax

`SetSegMetadataNum(`*`doc_handle`*`, `*`metadataName`*`, `*`value`*`);`

doc_handle	Identifies a document
metadataName	A string specifying the metadata name associated with the document.
value	A number specifying the value of the metadata for the specified document.

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

`0`	Metadata is set.
num<>`0`	Error

Example

```
$doc_id = GetFirstSegDoc();
while(num($doc_id) > 0)
{
   $ret = SetSegMetadataNum($doc_id, "invoice_no", 1234);
   $doc_id = GetNextSegDoc($doc_id);
}
```

SetSerNo

Syntax

`SetSerNo(`*`filename`*`, `*`str_serial_id`*`, `*`num_value`*`);`

filename	A string specifying the path and filename of the file containing the number sequence.

str_serial_id	A string specifying the serial ID.
num_value	Number you want to set the serial number to.

Description

Sets the value of a serial number in a number sequence. See `GetSerNo` for additional information.

Returns

`0`	OK
`-1`	Failed

Example

```
setserno("serno.txt","ID1",100);
```

SetSubst

Syntax

`SetSubst(`*tbl_file*`, `*str_key*`, `*num_col*`, `*str_value*`);`

tbl_file	A string specifying the path and filename of the substitution table.
str_key	The key of the entry to search for.
num_col	A number specifying the value column. The first value column number is `0`.
str_value	A string representing a value to assign.

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.

```
//!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`:

```
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:

```
//!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*`);`

num	A number specifying an adjustment in millimeters.

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);`

num	A number specifying an adjustment in millimeters.

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

`0`	OK
`-1`	Failed

Example

```
if(num($count)>15)
{
    skip();
}
```

Sort

Syntax

```
Sort(sortkey_id, value_1[, value_2,..value_n]);
```

sortkey_id	A string containing the key identifier.
value_1 . . *value_n*	Strings specifying the key values.

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

`0`	OK
`-1`	Failed

Example

```
sort("inv", $invno);
```

SortSegDoc

Syntax

`SortSegDoc(`*`value_1`*`[, `*`value_2`*`,..`*`value_n`*`]);`

`value_1,.. value_N`	Strings specifying the metadata keys used for sorting documents.

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

`0`	Documents are sorted.
num<>`0`	Error

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

num	The X coordinate in millimeters.
0	The function was called in a bad context.

Example

```
$xcoord = StGetParagraphPageXMmm();
```

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

num	The X coordinate in points.
0	The function was called in a bad context.

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

num	The Y coordinate in millimeters.
0	The function was called in a bad context.

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

num	The Y coordinate in points.
0	The function was called in a bad context.

Example

```
$ycoord = StGetParagraphPageYPt();
```

StGetParagraphXMm

Syntax

```
StGetParagraphXMm();
```

Description

Returns the X-coordinate in millimeters for the upper left corner of an object's paragraph, relative to the text area origin.

Returns

num	The X coordinate in millimeters.
0	The function was called in a bad context.

Example

```
$xcoord = StGetParagraphXMm();
```

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

num	The X coordinate in points.
0	The function was called in a bad context.

Example

```
$xcoord = StGetParagraphXPt();
```

StGetParagraphYMm

Syntax

```
StGetParagraphYMm();
```

Description

Returns the Y-coordinate in millimeters for the upper left corner of an object's paragraph, relative to the text area origin.

Returns

num	The Y coordinate in millimeters.
0	The function was called in a bad context.

Example

```
$ycoord = StGetParagraphYMm();
```

StGetParagraphYPt

Syntax

```
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

num	The Y coordinate in points.
0	The function was called in a bad context.

Example

```
$ycoord = StGetParagraphYPt();
```

StGetProcessingProperty

Syntax

```
StGetProcessingProperty(property);
```

property	The StoryTeller processing property to retrieve the value for. For available properties, see the table below: **Note:** Property keys are case sensitive!

Property	Description
`language`	The current language used during runtime.

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

```
$locale = StGetProcessingProperty("language");
```

StGetProperty

Syntax

```
StGetProperty(property);
```

property	The non-dimensional StoryTeller property to retrieve the value for. For available properties, see the table below: **Note:** Property keys are case sensitive! **Note:** To retrieve dimension properties, use *StGetPropertyMm* on page 324 or *StGetPropertyPt* on page 326.

Property	Description
`Name`	The object name
`Description`	The object description
`Visibility`	The visibility of the object `0` – Normal `1` – Hidden (Object is suppressed) `2` – Invisible
`Rotation`	Clockwise rotation in degrees.
`SizeMode`	For images: `0` – Resize `1` – Original

`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.

`LineColor`	The line color. See StoryTeller documentation for supported formats.
`FillType`	This property represents the Fill > Type property in Properties panel. 0 – Solid 1 – Linear gradient horizontal 2 – Linear gradient vertical 3 – Radial gradient
`FillColor`	The fill color. See StoryTeller documentation for supported formats.
`FillSecondColor`	The second fill color for gradient colors.
`ScaleMode`	`0` – None `1` – Scale `2` – Proportional
`ImageURI`	The URI to the image.
`ImageFormat`	The image format. Read-only.
`ImageBitDepth`	The image bit depth. Read-only.
`ImagePixelWidth`	The image pixel width. Read-only.
`ImagePixelHeight`	The image pixel height. Read-only.
`ImageResolutionX`	The horizontal image resolution.
`ImageResolutionY`	The vertical image resolution.
`ImagePage`	The number of the page for a multi-page TIFF image.
`OverlayURI`	The URI to the overlay
`OverlayPage`	The number of the page for a multi-page overlay.
`BarcodeName`	The barcode type.
`BarcodeData`	The barcode data.
`SubstitutionURI`	The URI to the substitution.
`Cells[n]/FillColor`	The fill color of the *n*th cell on the row, where `n=0` is the left cell.
`Cells[]/LineColor`	The border color of the *n*th cell on the row, where `n=0` is the left cell.
`Cells[]/TopLineColor`	The border color of the top border of the *n*th cell on the row, where `n=0` is the left cell.

`Cells[]/ LeftLineColor`	The border color of the left border of the *n*th cell on the row, where `n=0` is the left cell.
`Cells[]/ RightLineColor`	The border color of the right border of the *n*th cell on the row, where `n=0` is the left cell.
`Cells[]/ BottomLineColor`	The border color of the bottom border of the *n*th cell on the row, where `n=0` is the left cell.
`ColumnsCount`	The number of columns in a table. Use in script before the table.
`CellsCount`	The number of cells in a row. Use in script before the row.

Description

Retrieves the value of the specified non-dimensional property in StoryTeller.

Returns

`string`	The value of the specified property. For invalid properties an empty string is returned.

Example

```
$visibility = StGetProperty("Visibility");
```

StGetPropertyMm

Syntax

`StGetPropertyMm(`*`property`*`);`

`property`	The dimensional property to retrieve the value for. For available properties, see the table below. **Note:** Property keys are case sensitive! **Note:** To retrieve a non-dimensional property, use *StGetProperty* on page 321.

Property	Description
`X`	The horizontal position of the object's anchor point relative to the origin (upper left corner) of the parent object, normally the page.

`Y`	The vertical position of the object's anchor point relative to the origin (upper left corner) of the parent object, normally the page.
`Width`	The width of the object's bounding box.
`Height`	The height of the object's bounding box, or the height of a table row.
`TextHeightMin`	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.
`TextHeightMax`	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.
`ContentVShrink`	The maximum vertical shrink of the content within a shape.
`ContentVGrowth`	The maximum vertical growth of the content within a shape.
`ContentHShrink`	The maximum horizontal shrink of the content within a shape.
`ContentHGrowth`	The maximum horizontal growth of the content within a shape.
`ShapeVShrink`	The maximum vertical shrink of the shape.
`ShapeVGrowth`	The maximum vertical growth of the shape.
`ShapeHShrink`	The maximum horizontal shrink of the shape.
`ShapeHGrowth`	The maximum horizontal growth of the shape.
`LineThickness`	The line thickness.
`Columns[]/Width`	The column width of the *n*th column of the table, where `n=0` is the left column.
`RowHeight`	The row height.
`Cells[]/LineThickness`	The line thickness of the *n*th cell on the row, where `n=0` is the left cell.
`Cells[]/TopLineThickness`	The line thickness of the top border of the *n*th cell on the row, where `n=0` is the left cell.
`Cells[]/LeftLineThickness`	The line thickness of the left border of the *n*th cell on the row, where `n=0` is the left cell.

`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

num	The value of the specified property in millimeters. For invalid properties `0` is returned.

Example

```
$Height = StGetPropertyMm("Height");
```

StGetPropertyPt

Syntax

`StGetPropertyPt(`*property*`);`

property	The dimension property to retrieve the value for, in points. See *StGetPropertyMm* on page 324 for available properties. **Note:** Property keys are case sensitive! **Note:** To retrieve a non-dimensional property, use *StGetProperty* on page 321.

Description

Retrieves the value of the specified dimension property in StoryTeller. The value is returned in points.

Returns

num	The value of the property in points. For invalid properties `0` is returned.

Example

```
$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:

Object	...is empty if...
Composition Center Section	No fragment is returned for the section.
Substitution, Text object, Story frame	It contains no text, commands, or layout objects, and all content objects are empty.
Table row	No cell in the row contains text, commands, or layout objects, and all content objects are empty.
Table	Table has no body rows, or alla body rows are empty.
Page	No Story frames on the page contains text, commands, or layout objects, and all content objects are empty.

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);
```

language	The language to translate into
key	The key to identify the entry in the language file.

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

```
$value = stLanguageLookup(stGetProcessingProperty("language"),
&data_field);
if($value="")
{
   $value=&data_field;
}
```

Note: This function call generates a log message when value is not found for the key.

StMsgAddNode

Syntax

StMsgAddNode(strXPath, name);

strsXPath	The position in the tree where the node is appended
name	The name of the node to append.

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

```
StMsgAddNode("/", "/New_node");
```

StMsgAttachXML

Syntax

StMsgAttachXML(strXPath, xml, src_XPath);

strXPath	XPath to a Message node where a DOM node is appended.
xml	An XML string from which an XML DOM node is loaded.
src_XPath	DOM node that is appended to a Message node

Description

StoryTeller tool function.

Appends an XML DOM node – loaded from an XML string – to a Message node.

Returns

N/A

Example

```
StMsgAttachXML("/","<MyString>" , "/Resources" );
```

StMsgAttachXMLFile

Syntax

```
StMsgAttachXMLFile(strXPath, xml_file, src_XPath);
```

strsXPath	XPath to a Message node where a DOM node is appended.
xml_file	An XML file from which an XML DOM node is loaded.
src_XPath	DOM node that is appended to a Message node

Description

StoryTeller tool function.

Appends an XML DOM node – loaded from an XML file – to a Message node.

Returns

N/A

Example

```
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);
```

int id	The ID of the timer you want to stop.
int logIt	0 – Do not log elapsed time. 1 – Log elapsed time. 2 – Log elapsed time and write to file.

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);
```

property	The non-dimensional property to set the value for. See *StGetProperty* on page 321 for available properties. **Note:** Property keys are case sensitive! **Note:** To set dimension properties, use *StSetPropertyMm* on page 331 or *StSetPropertyPt* on page 332.
value	The value to set on the specified property.

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);
```

property	The dimension property to set the value for. See *StGetPropertyMm* on page 324 for available properties. **Note:** Property keys are case sensitive! **Note:** To set non-dimension properties, use *StSetProperty* on page 331.
value	The value in millimeters to set on the specified property.

Description

Sets the specified value in millimeters on 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);
```

property	The dimension property to set the value for. See *StGetPropertyMm* on page 324 for available properties. **Note:** Property keys are case sensitive! **Note:** To set non-dimension properties, use *StopTimer(0, 1);* on page 330.
value	The value in points unit to set on the specified property.

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.

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);
```

str	A string.
num_strlen	Required length of string.

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

```
$a=strlen("hello");
```

Result:

```
$a=5
```

StrQTok

Syntax

```
StrQTok(str, char_sep, quote_char, esc_char, result_array);
```

str	The string to be tokenized.
char_sep	A character separator.
quote_char	A string quote character.
esc_char	A quote escape character.
result_array	The array to which the tokens will be assigned.

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

```
$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"
$arr1[2]="A, B and 'C'"
```

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

```
$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

```
StURIExist(str URL);
```

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

```
$validbool = StURIExist("dcresource:e16150ac-e5bf-434e");
```

Subst

Syntax

```
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`:

```
$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);`

tbl_file	A string specifying the path and filename of the substitution table.
str_key	A string specifying a key in the table.
array	A variable that receives the array of columns from the substitution entry.

Description

Looks up a key in a substitution table, and creates an array containing the entry values. Returns the number of entry values.

Returns

num	If the key was found, returns the number of items returned in `array`.
`-1`	The key was not found.

Example

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

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

The following script creates an array for the entry with key `agatha`:

```
$nCol = SubstArr("../data/tables/author.tbl", "agatha", $addrArr);
$surName = $addrArr[0];
$birthDay =  $addrArr[1];
```

Result:

```
$nCol = 2
$surName = christie
$birthDay = 1890
```

Notes

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

SubStr

Syntax

`SubStr(`*`str`*`, `*`num_pos`*`, `*`num_len`*`);`

str	An input string.
num_pos	A number specifying the first position of sub-string in the string.
num_len	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.

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);
```

str	An input string.
search_str	A search string to be replaced.
repl_str	A replacement sub-string.

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"
```

T

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:

```
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

```
$var1="StreamServe";
$var2=ToUpper($var1);
```

Result:

```
$var2="STREAMSERVE"
```

U

UpdateDocStatus

Syntax

```
UpdateDocStatus(status[, error_code]);
```

status	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`.
error_code	The error code to update the repository with when the status is `error`.

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

`0`	Status updated.
num<>`0`	Error

Example

```
$ret = UpdateDocStatus("processed");
```

UpdateExternalCompletionStatus

`UpdateExternalCompletionStatus` is an old function and is removed.

UpdateJobStatus

Syntax

```
UpdateJobStatus(status[, error_code]);
```

status	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`.
error_code	The error code to update the repository with when the status is `error`.

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

`0`	Status updated.
num<>`0`	Error

Example

```
$ret = UpdateJobStatus("processed");
```

UpdatePPDocStatus

Syntax

```
UpdatePPDocStatus(doc_handle, status[, error_code]);
```

doc_handle	Identifies the document.
status	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`.
error_code	The error code to update the repository with when the status is `error`.

Description

Updates repository status of the document specified in the *doc_handle* parameter returned by either the `GetFirstPPDoc()` or the `GetNextPPDoc()` function.

Returns

`0`	Status updated
num<>`0`	Error

Example

```
$doc_id = GetFirstPPDoc();
while(num($doc_id) > 0)
{
   $ret = UpdatePPDocStatus(num($doc_id), "processed");
   $doc_id = GetNextPPDoc($doc_id);
}
```

UpdatePPJobStatus

Syntax

```
UpdatePPJobStatus(status[, error_code]);
```

status	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`.
error_code	The error code to update the repository with when the status is `error`.

Description

Updates status of the current post-processor job in the repository.

Returns

`0`	Status updated.
num<>`0`	Error

Example

```
$doc_id = GetFirstPPDoc();
while(num($doc_id) > 0)
{
   $ret = UpdatePPJobStatus(num($doc_id), "processed");
   $doc_id = GetNextPPDoc($doc_id);
}
```

UpdateSegDocStatus

Syntax

`UpdateSegDocStatus(`*`doc_handle`*`, `*`status`*`[, `*`error_code`*`]);`

`doc_handle`	An identifier specifying a document.
`status`	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`.
`error_code`	The error code to update the repository with when the status is `error`.

Description

Updates repository status of the document specified in the *`doc_handle`* parameter returned by either the `GetFirstSegDoc()`, `GetCurrSegDoc()`, or the `GetNextSegDoc()` function.

Returns

`0`	Status updated.
num<>`0`	Error

Example

```
$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`*`]);`

doc_handle	Identifies the document.
status	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 • custom • custom5 • custom6 • custom7 • custom8 • custom9
error_code	The error code to update the repository with when the status is error.

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

0	Status updated.
num<>0	Error

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.

0	The file was successfully closed.
1	Failure.

Example

```
//Close "myconnection" and store the returned value in $closeok

$closeok = URLclose("myconnection");
```

URLExist

Syntax

```
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

1	The URL exists.
0	The URL does not exist.

Example

```
$validbool = URLExist("http://myusername:mypassword@myserver/
file.txt");
```

URLOpen

Syntax

```
URLOpen(connectionID, strURL, "r");
```

connectionID	A name you specify which will be used in all the following URL functions to identify which URL to access.
strURL	A string specifying the URL of the file.
"r"	Opens the connection in read mode.

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.

`0`	The file was successfully opened.
`1`	Error

Example

```
//Open myfile.txt for reading

$readok = URLopen("my_conn_ID", "ftp://myserver/myfile.txt", "r");
```

URLReadLn

Syntax

`URLReadLn(`*`connectionID`*`, `*`var`*`);`

`connectionID`	The connection ID you specified when opening 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 `URLOpen` before calling `URLReadLn`. Use the `URLClose` function to close the file after reading.

Returns

A number indicating whether the line was successfully read.

`0`	The line was successfully read.
`1`	Failure.

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]yymmdd);
```

[*yy*]*yymmdd*	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);`

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

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

`1`	File was written successfully.
`0`	File was not written.

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

```
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.

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*).

Document Search Criteria	
Field name	The field in the Lotus Notes database that the code block should search for a match.
Criteria	The search criteria, i.e. the variable you defined in the Event.

- The variable in the Process in which to store the retrieved data.

Field to variable mappings	
Field name	The field in the Lotus Notes Database that contains the data that the code block should retrieve.
Variable name	The variable in the Process in which the retrieved data should be stored.

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.
3. Select **Insert** from the Code Wizard menu. The Insert new block dialog box opens.
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.