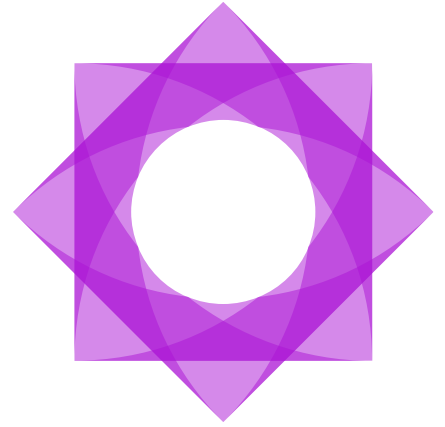




Lasernet 10.

Lasernet Scripting 10

Torben Pedersen, Haveela Anil, Sunil Panchal
Revision 10
2025-10-24

Contents.

1 Introduction.....	4
1.1 Who Should Use This Guide?	4
2 Terms of Use.	5
3 Expanding Functionality with Scripting.	6
3.1 Introduction to Scripting	6
3.2 Script Editor	7
3.3 Unit Testing.....	8
3.3.1 Grab JobInfos	10
3.3.2 Check Class	12
3.3.3 Loading and Saving Jobs	13
3.4 Scripts as Form Modifiers	13
3.4.1 Form Modifier Resolution Order	13
3.5 Manipulating Overlays	14
3.5.1 The page Object	14
3.5.2 Shortcut Functions.....	15
3.6 Manipulating Printer Profiles.....	15
3.6.1 The page Object	15
3.6.2 Shortcut Functions.....	16
3.7 Manipulating Rearranges	16
3.7.1 CurrentRearrange.....	16
3.7.2 Add or Edit scripts from a Rearrange	18
3.8 Manipulating Charts (Deprecated).....	19
4 Scripting Reference.....	21
4.1 Global Lasernet Script Functions	21
4.2 Global Lascript Script Classes	35
4.2.1 Job	35
4.2.2 Logger	43
4.2.3 Modifier	45
4.2.4 AzureStorageCommands	46
4.2.5 DatabaseCommands.....	47
4.2.6 SharepointCommands.....	48
4.2.7 XMLTransformers.....	49
4.2.8 SystemInfo.....	50
4.2.9 ByteArray	51
4.2.10 GUID.....	53
4.2.11 Hash	55
4.2.12 MagicBytes	57
4.2.13 Configuration	58
4.3 Form Engine Variables	62
4.4 Form Engine Classes	62
4.4.1 Grab.....	62
4.4.2 InputRearrange.....	67

4.4.3 OutputRearrange	72
4.4.4 Sheet	74
4.4.5 Form	76
4.5 Global Script Classes	77
4.5.1 ActiveXObject	77
4.5.2 File	79
4.5.3 Zip	95
4.6 JavaScript Global Classes	98
4.6.1 Color	98
4.6.2 Dir	99
4.6.3 FontStyle	106
4.6.4 ShapeStyle	108
4.6.5 Connection	110
4.6.6 Module	111
4.6.7 Printer	112
4.6.8 PrinterProfile	113
4.6.9 Server	114

1 Introduction.

1.1 Who Should Use This Guide?

This guide covers the scripting features and extensibility offered by Lasernet Output Management. It is written both for the Lascript system administrator and Lascript developers. It is intended primarily as a reference to the different JavaScript functions in Lascript.

2 Terms of Use.

No part of this publication may be reproduced, transmitted, transcribed, or translated into any language in any form by any means without the prior written permission of Formpipe Software. The information in this manual is subject to change without notice. Any company names or data is fictive unless otherwise stated.

Formpipe Software shall not be liable for any loss or damage whatsoever arising from the use of this manual and the information contained therein (including errors or omissions).

Trademarks of other companies mentioned in this document appear for identification purposes only and are the property of their respective companies.

© 2025 Formpipe Software.

3 Expanding Functionality with Scripting.

3.1 Introduction to Scripting

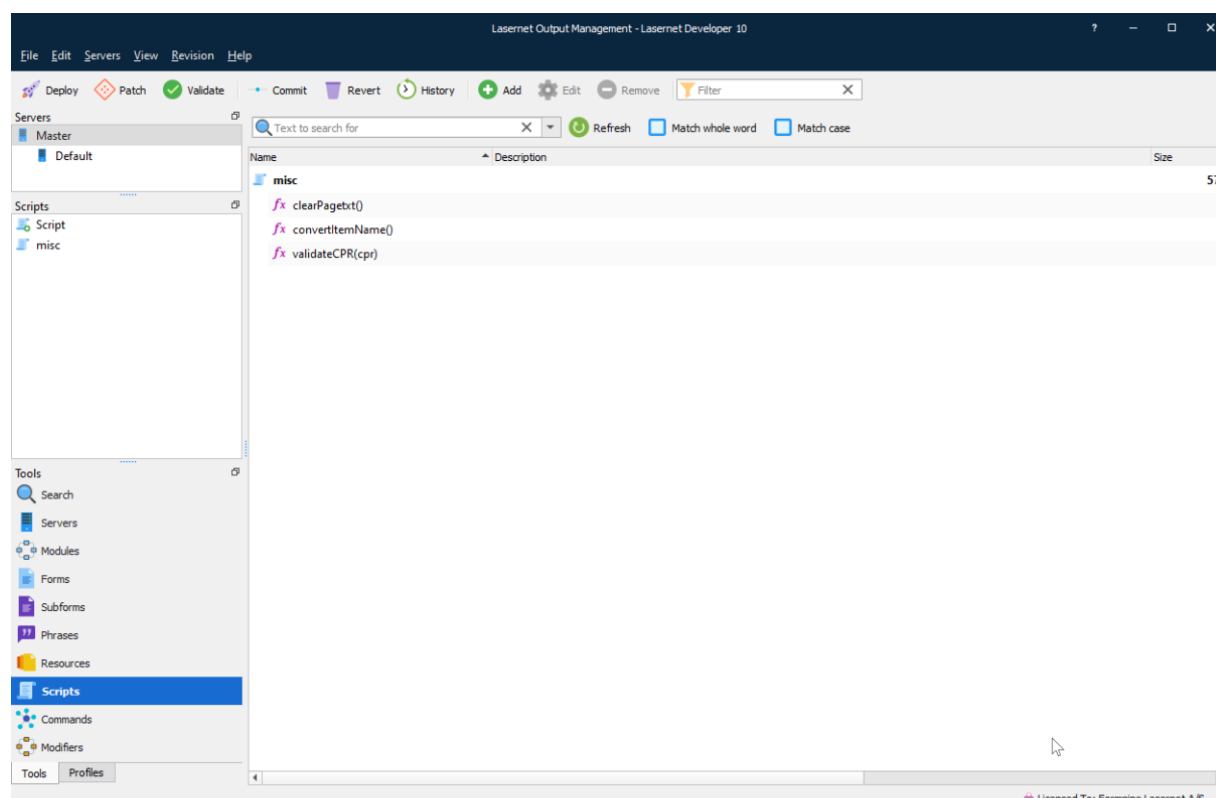
Lasernet contains a scripting framework supporting the development of advanced functionality.

The JavaScript engine in Lasernet uses QtScript, which is compliant with the ECMA-262 specification.

Scripts are typically used for checksum calculations, conditional overlays, conditional targets and conditional printer profiles, but it is possible to develop anything using scripting to extend your Lascript solution.

This hands-on introduction to scripting in Lascript will focus on some example scripts that are commonly used in many setups. Some of the terms and explanations in this section may seem difficult if you are new to the subject, but we encourage you to read it all in order to familiarize yourself with the opportunities and possibilities of scripting. A 'script' usually consists of one or more tasks, grouped together under a single name to provide specific functionality.

The Lascript Developer contains a script editor with the ability to maintain your own scripts. You can access the script editor by creating a new script or opening an existing one.



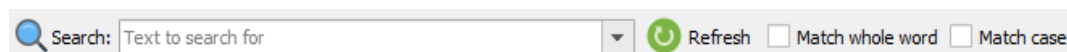
Click  **Add** to **Add** a new script object or select  **Script** in the left view and drag the icon into the list view containing script objects. When a script has been added it will be available globally for the whole configuration and you can add it to any module form as modifier or rearrange object.

Click  **Edit** to **Edit** an existing script or double-click it. A script object can contain several functions. If you double-click on the function name, you will be guided to where the selected function starts.

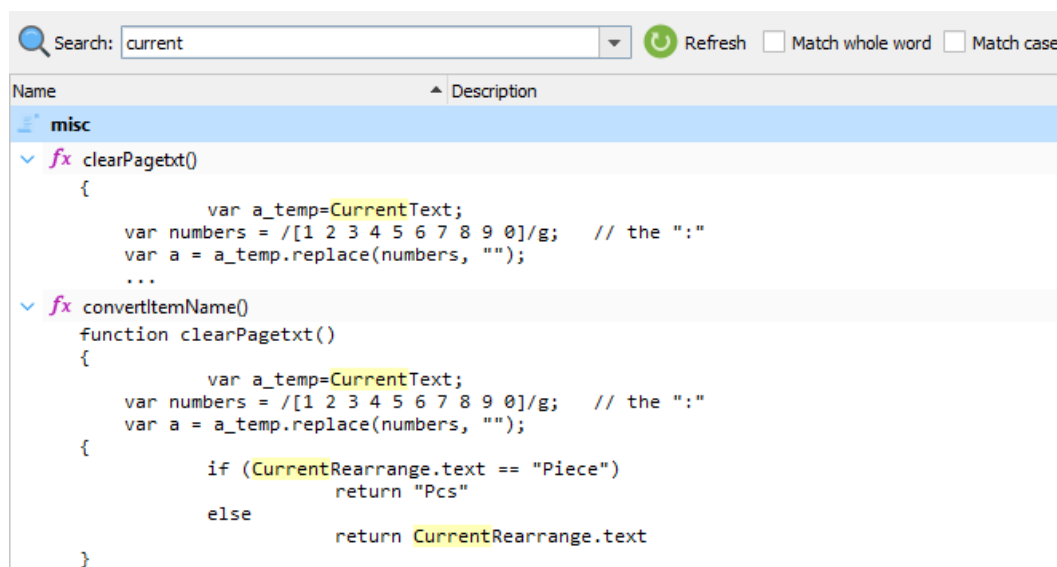
Select an object (root items are marked in bold) and click  **Remove** to **Remove** an existing object. This will remove all functions included in the main object. If you want to remove a function only, this is maintained in the Script Editor. After a function has been removed you must also remember to manually remove the object reference added in modules or forms, otherwise you will get an error when validating the configuration or when processing jobs through the modules or forms.

If you have a large library of script objects and functions, you can use the **search** functionality for finding the text string(s), you are searching for. Type in the search string and press Refresh and all the functions with a match will be listed and the text string(s) will be marked with yellow. A few script lines before and after the match will be listed as well to give a sense of context.

In your search for strings, you can decide to “Match whole word” or “Match case”.



After a search you must clear the Search field and press Refresh if you want to list all available script objects and functions again.



Name	Description
misc	
fx clearPagetxt()	
fx convertItemName()	

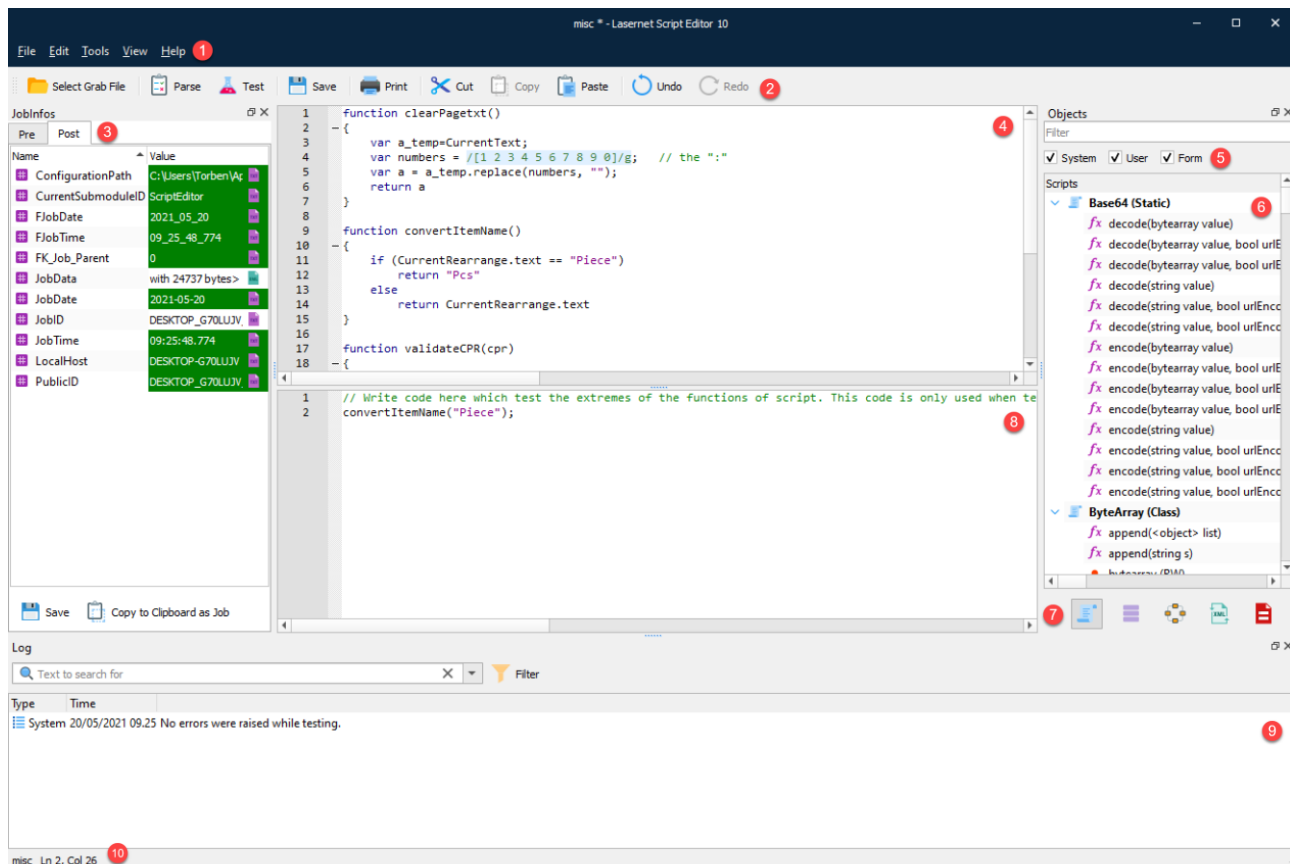
```

function clearPagetxt()
{
    var a_temp=CurrentText;
    var numbers = /[1 2 3 4 5 6 7 8 9 0]/g;    // the ":"
    var a = a_temp.replace(numbers, "");
    ...
}

function convertItemName()
function clearPagetxt()
{
    var a_temp=CurrentText;
    var numbers = /[1 2 3 4 5 6 7 8 9 0]/g;    // the ":"
    var a = a_temp.replace(numbers, "");
    {
        if (CurrentRearrange.text == "Piece")
            return "Pcs"
        else
            return CurrentRearrange.text
    }
}
  
```

3.2 Script Editor

The editor supports the development of scripts by providing you with features for efficiently writing and editing code.



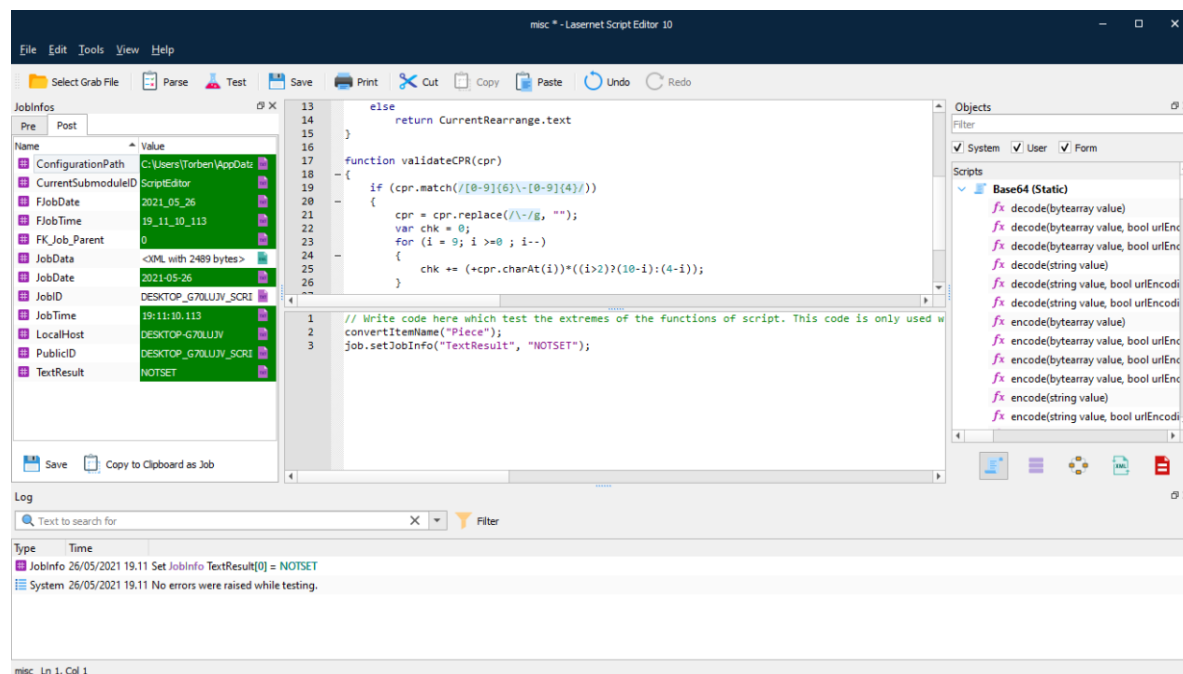
1. Menu bar with File, Edit, Tools and View functionality
2. Tool bar with Parse, Test, Save, Print, Cut/Copy/Paste and Undo/Redo functionality
3. Name and values of JobInfos in pre (before execution) and post (after execution) mode
4. Script Editor with code hints and indentation guides
5. Filtering support for Lasernet System, User and Form objects
6. Object view with drag'n'drop support to Script Editor area
7. Select Scripts, Database Commands, Modifiers or XML Transformers object view
8. Unit testing
9. Logger view for testing and debugging of scripts
10. Status for filename and cursor positioning

3.3 Unit Testing

Unit testing can help you to validate your script functions without uploading the configuration to server and parsing jobs through Lascript.

The idea of unit testing is to test the extremes for each function, to make sure that it works as expected and to catch errors if something is changed.

It is normal practice to implement the unit test before implementing the actual function which is being tested.



In **Tools** activate **Unit Test** and a window for the Unit Test editor will appear.

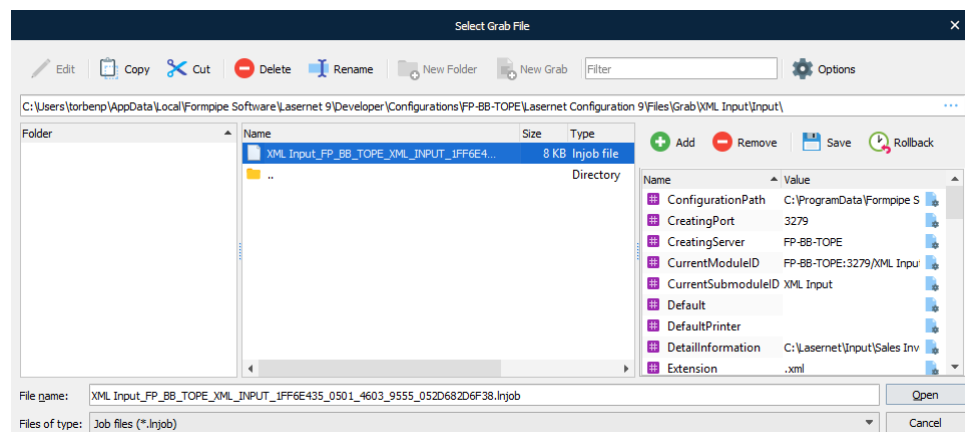
In the Unit Test mode you can type the name of the function and the conditions you would like to validate. The tool bar contains a **Parse** button to validate the syntax and **Test** button to execute the Unit Test.



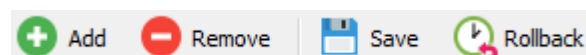
In the logger window you will receive the results of your Unit Test.

3.3.1 Grab JobInfos

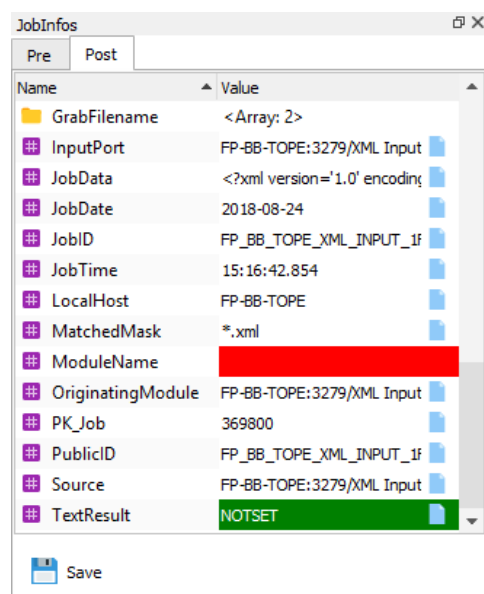
Activate **View → Grab JobInfos** if you want to list the results of JobInfos created in Unit Test mode. First you must load a grab file containing JobData. Go to **File → Select Grab File** and browse for a valid Job file (.Injob) that has been grabbed by the Lasernet Server.



When the Job file has been loaded, the JobInfos view **(2)** will list JobInfos from the Job file **(Pre)** mode and new values and new JobInfos, created via Unit Testing **(Post)** mode.



For testing purposes, you can always Add, Remove or Edit the contents of a JobInfo in the JobInfo view. The Save functionality will save any changes made in the JobInfo view into the loaded Job file. If you want to roll back JobInfos click the Rollback button. You cannot roll back changes after JobInfos have been saved into the Job file. The Rollback will affect all of the changes made to the Pre Job since it was loaded or last saved.



Log		
Type	Time	Message
 JobInfo	24-08-2018 15:40	Set JobInfo TextResult[0] = NOTSET
 System	24-08-2018 15:40	No errors were raised while testing.

The JobInfo view will be updated with the results of JobInfo manipulated commands, like `job.setJobInfo("TextResult", value)` and the logger window will contain the results of your unit test.

3.3.2 Check Class

A major part of testing functions is to look for errors raised during execution. In order to test functions which do not raise errors, but return results or manipulate objects or variables, Lasernet provides a Check class for this purpose.

The Check class contains two static functions which can be used to check results of functions and raise errors if they are not as expected.

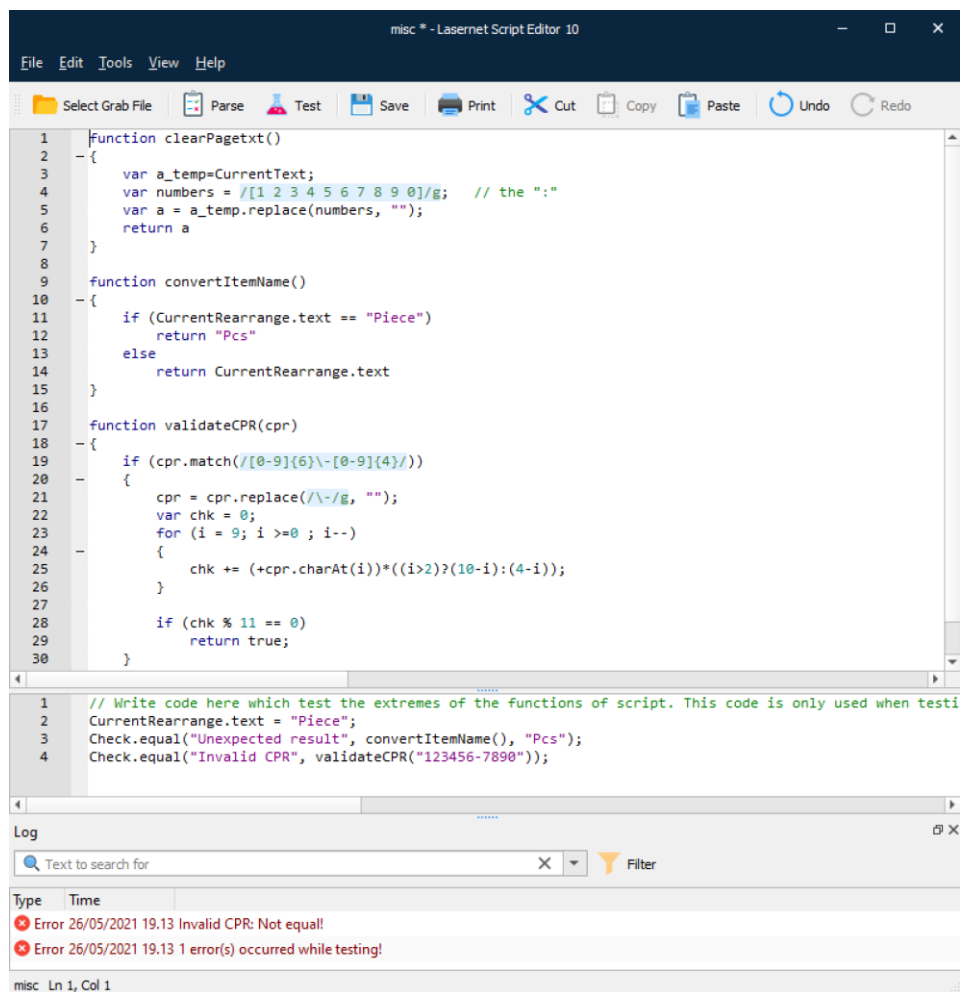
▼  **Check (Static)**
 `equal(string message, bool equal)`

The Check equal functions can be used by dragging them from the Object list, to the unit test script editor.

The first implementation checks if two objects are equal. This can be used for different object types including Color.

The second implementation raises the error message if the equal boolean is false. This way the user can compare anything in a boolean expression.

The Example below shows both equal functions being used. The first with an unexpected result from `convertItemName`, second with an invalid CPR Number (Danish Personal Security number);



```

misc * - Lascript Editor 10
File Edit Tools View Help
Select Grab File Parse Test Save Print Cut Copy Paste Undo Redo

1 function clearPageText()
2 -{
3     var a_temp=CurrentText;
4     var numbers = /[1 2 3 4 5 6 7 8 9 0]/g; // the ":"
5     var a = a_temp.replace(numbers, "");
6     return a
7 }
8
9 function convertItemName()
10 -{
11     if (CurrentRearrange.text == "Piece")
12         return "Pcs"
13     else
14         return CurrentRearrange.text
15 }
16
17 function validateCPR(cpr)
18 -{
19     if (cpr.match(/[0-9]{6}\-[0-9]{4}/))
20     {
21         cpr = cpr.replace(/\-/g, "");
22         var chk = 0;
23         for (i = 9; i >= 0 ; i--)
24         {
25             chk += (+cpr.charAt(i))*((i>2)?(10-i):(4-i));
26         }
27         if (chk % 11 == 0)
28             return true;
29     }
30 }

1 // Write code here which test the extremes of the functions of script. This code is only used when testi
2 CurrentRearrange.text = "Piece";
3 Check.equal("Unexpected result", convertItemName(), "Pcs");
4 Check.equal("Invalid CPR", validateCPR("123456-7890"));

Log
Text to search for X Filter
Type Time
Error 26/05/2021 19:13 Invalid CPR: Not equal!
Error 26/05/2021 19:13 1 error(s) occurred while testing!
misc Ln 1, Col 1

```

Two errors are raised showing the error message. In the first call, the reason for the failure is also included.

3.3.3 Loading and Saving Jobs

When a function is based on a Job it can be useful to use a grabbed Job (.Injob) to test against.

A Job can be loaded via the load function on the job object. The current job can also be saved to disk via the save function on the job object.

Multiple jobs can also be loaded in order to run a range of checks on them, something which is not possible with the loaded grab Job.

```

v  job (Static)
  fx load(string filename)
  fx load(string filename, string modulename)
  fx save(string filename)

```

Loading and saving must be done with .Injob files and can be either absolute or relative to the grab folder of the configuration. Both functions return true if load/save was successful.

3.4 Scripts as Form Modifiers

When working with a Lasernet configuration there are several places where a script can be added in place of a modifier.

In a form, scripts can be connected to the following types of modifier points:

- Form
- Sheet
- Page (only for text output forms)

In addition to this, scripts can be attached to individual rearranges.

3.4.1 Form Modifier Resolution Order

When processing a form, the modifiers or scripts are run in the following order:

Modifier point **Form Start**

Modifier point **Sheet Start**

Definition of all JobInfos contained in the Sheet

Modifier point **Sheet Before Script**

JobInfo *NumberOfPages* contains the number of pages in the document produced

JobInfo *CurrentPage* is updated

Modifier point **Page Start**

Rearrange Scripts in the Conditional Area

Modifier point **Page End**

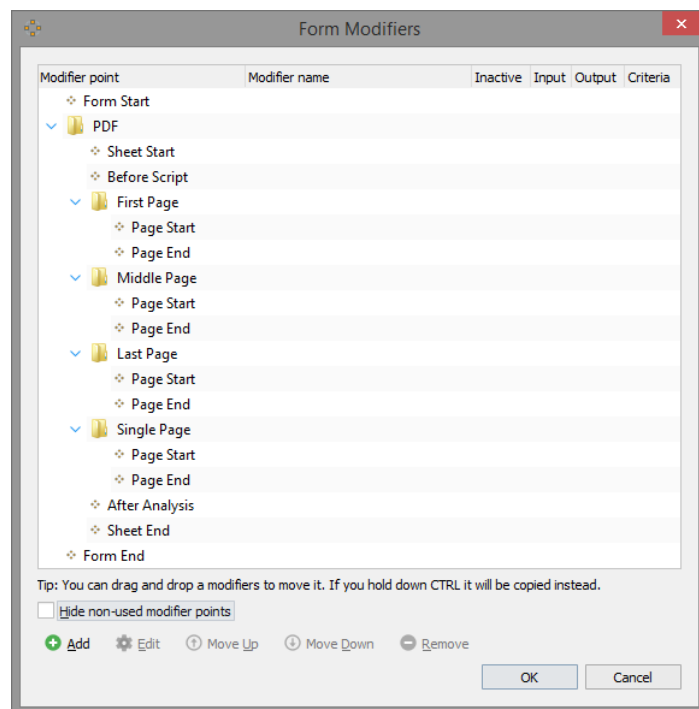
Scripts attached to rearranges

Repeated for each page

Modifier point Sheet After Analysis

Modifier point **Sheet End**

Modifier point **Form End**



3.5 Manipulating Overlays

This information applies to scripting in the Form Engine.

When producing a document it is not often that the same overlay is used for every single page. As such, Lasernet provides an easy way to use separate overlays for the first, last and middle pages. Sometimes you need even more flexibility, which where scripting comes in. To aid with script writing, Lاسernet provides the ability to access the Page object using the following functionality:

3.5.1 The page Object

When the Form Engine creates text output, it creates a number of pages. Each of these pages can be accessed from within a script and the page properties can be manipulated. It is also possible to clear and/or add overlays on each page. The pages are available in an array called pages, where the first page is named pages[1] and so forth. There is one item for each page in the array. For example, if you would like to add an overlay to the first page, you can do it in this way:

```
pages[1].addOverlay("SomeOverlay.emf", true, false);
```

'Pages' refers to the page object and allows you to access the page array. The [1] reference gives us access to the first page on which we call the method addOverlay. This method is used for instructing Lاسernet to add an overlay to that particular page. Inside the parenthesis three parameters can be specified. The first parameter is required and contains a string identifying the filename of the overlay that you wish to add to the page. The second parameter is optional and contains a boolean (true/false) value where true means that Lاسernet will remove any existing overlays on the page before adding the new one and false means that

Lasernet will keep any existing overlays. The default value of the second parameter is false and as such could be written as:

```
pages[1].addOverlay("SomeOtherOverlay.emf");
```

This will instruct Lasernet to keep the existing overlays on the page.

The third parameter is also optional and contains a Boolean (true/false) value where true means that the overlay will be visible in the Form Editor only and not processed by the Form Engine.

3.5.2 Shortcut Functions

Lasernet also provides two shortcut functions for adding overlays, either to all pages or to the current page:

```
addOverlay("NameOfTheOverlayFile.emf", true, false);  
addPageOverlay("NameOfTheOverlayFile.emf", true, false);
```

The addOverlay function is best called from an After Analysis modifier point, whereas addPageOverlay should be called from an On Page End modifier point.

3.6 Manipulating Printer Profiles

This information applies to the Form Engine. Please see the Printer Profiles section for a more thorough explanation of Printer Profiles.

To assist you in manipulating Printer Profiles, Lasernet provides you with the same page object as mentioned in chapter 3.5 Manipulating Overlays.

3.6.1 The page Object

The *page* object can be used to add and clear printer profiles for each page. The pages are available in an array called pages, where the first page is named pages[1] and so forth. There is one item for each page in the array. For example, if you would like to add a printer profile to the first page, you can do it in this way:

```
pages[1].addPrinterProfile("NameOfPrinterOutputOrPrinterService", "NameOfProfile", true);
```

'Pages' refers to the page object and allows you to access the page array. The [1] reference gives us access to the first page on which we call the method addPrinterProfile.

This tells Lasernet that we want to add a printer profile to that page. In the parentheses add the first parameter: a string which contains the name of the Printer Output module, also referred as a port, or Printer Service module on which to add the profile. Note that this is just the name of the port without any computer name prefixed. The second parameter tells Lasernet the name of the Printer Profile that we want to add for that Printer Output module or Printer Service module. This profile must have been created on the Printer Profiles page in the Lasernet Developer, when using Printer Output as printing module, or in the Printer Service application, when using Printer Service as printing module. The third parameter tells Lasernet to remove any existing profiles for the given port before adding the new profile. Changing this value to 'false' would have kept the existing profiles. Note: only the profiles for the given port are removed. This is contrary to previous versions of Lasernet where all profiles for all ports would have been removed.

Typically you would always want to remove existing profiles, since the output port will only use the first profile added. However, to allow for both backwards compatibility and future enhancements the parameter is still in use.

3.6.2 Shortcut Functions

To make it easier to add a printer profile to a page, Lasernet provides two shortcut functions: `addPrinterProfile` and `addPagePrinterProfile`. These two functions add a printer profile to all pages or to the current page.

```
addPrinterProfile("NameOfPrinterOutputOrPrinterService", "ProfileName");  
addPagePrinterProfile("NameOfPrinterOutputOrPrinterService", "ProfileName");
```

The `addPrinterProfile` function is best called from a Sheet → After Analysis point, whereas `addPagePrinterProfile` should be called from a Page End modifier point.

Note that both of the above functions will replace any existing printer profiles for the given port!

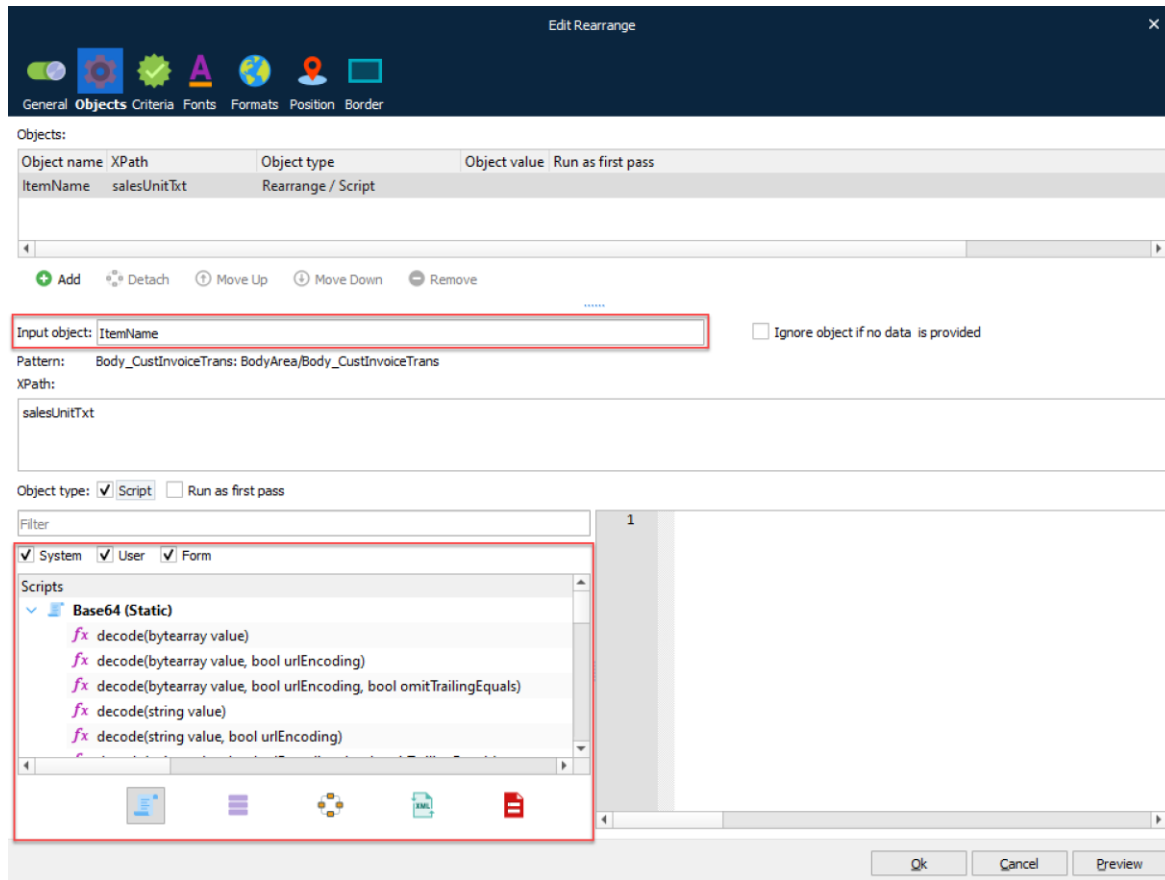
3.7 Manipulating Rearranges

This information applies to scripting in the Form Editor.

When creating rearranges in Lascript it is possible to manipulate the content and properties of the rearrange using a script. This is especially useful when calculating checksums for barcodes etc.

3.7.1 CurrentRearrange

The `CurrentRearrange` variable is available for scripts being run from a rearrange where the rearrange has been given a name. The name should be unique, otherwise Lascript may get confused and assign the wrong rearrange to `CurrentRearrange`. It is very important that the rearrange is named, which is why Lascript automatically assigns a name to it if the Active check box is selected. If the rearrange does not have a name, `CurrentRearrange` is not assigned.



When the above conditions are met, Lasernet assigns an InputRearrange to CurrentRearrange. This can then be used by a script to access data which has been picked up from the left side, manipulated and returned as the output value of the rearrange.

The script must return a value if the rearrange is to display anything. If you would like the rearrange to output nothing, simply return an empty string. The script can do the calculation directly or call a function in one of the script files.

```
'#' + CurrentRearrange.text + '#';
calcTheRearrange();
```

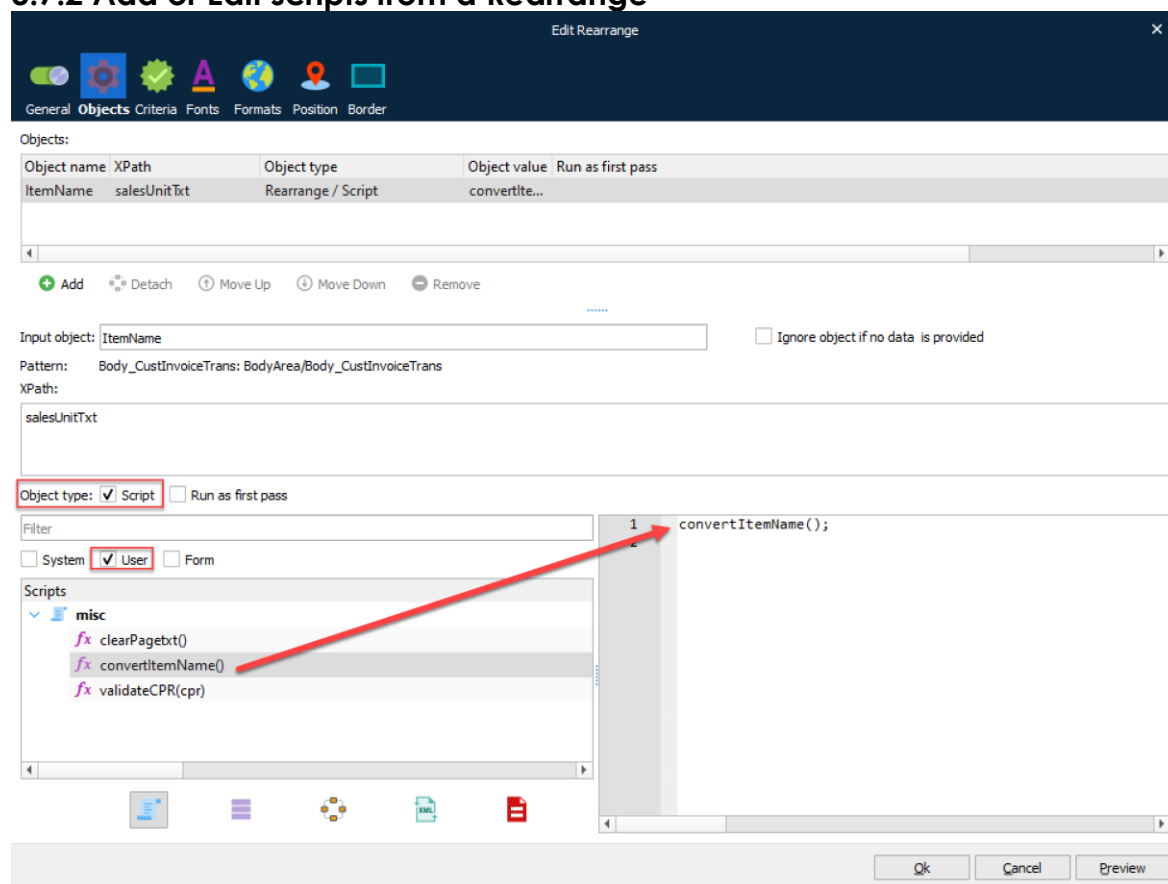
Besides manipulating the contents of the rearrange, it is also possible to change the appearance of the rearrange. An example could be to display negative numbers in red.

```
if (CurrentRearrange.number < 0)
    CurrentRearrange.color = red; // Constant defined in system.qs
```

You can even hide a rearrange by setting its color to white.

```
CurrentRearrange.color = white; // Constant defined in system.qs
```

3.7.2 Add or Edit scripts from a Rearrange



From the GUI of the Edit Rearrange dialog, you can add a new function to either a new script or an existing script, as well as Edit the chosen function.

In the list of scripts, both user defined and system defined scripts are available. You are only allowed to edit user defined scripts, since the LaseNet Form Editor prevents you from editing the system defined ones (the Edit option is not made available in the context menu).

When running scripts from a Rearrange, LaseNet will execute the script internally at an event point controlled by the system, which is important for calculating the positioning of text and number of pages.

3.7.2.1 Run as First pass

The “Run as First pass” parameter comes into effect at the event point execution time. By default, the setting is turned off, which is the recommended setting in most scenarios. However, by turning the setting on, you are able to accelerate the execution time to run as the first pass in the processing of a form. The output result of the script is used in analyzing the number of pages and calculation of text positioning on pages in the final form.

3.7.2.2 CurrentText

CurrentText is required to manipulate the content and properties of the rearrange using a script at the “Run as First Pass” event point, since the object name of the rearrange is not available at this point for the CurrentRearrange variable. For example, in order to manipulate the value of a conditional rearrange you can run the trim function to remove whitespaces surrounding the value of the current text:

```
var text = Trim(CurrentText);
```

The new value will figure in the output and become part of the calculation for the total number of pages in your output form.

3.8 Manipulating Charts (Deprecated)

The Lasernet Form Editor has a built-in chart tool for designing charts. More information about how to work with the chart tool can be found in the Form Editor manual. Inserting charts from script, as described here, is still supported but will be deprecated in future versions of Lاسرنهت.

Scripted charts are based on an external API which supports a wide variety of chart types. The API has been mapped to the scripting engine in Lاسرنهت, and as such charts can only be included via script. A “ChartHelper” script is available. The ChartHelper enables easy implementation of the most common setups for each chart type.

To use charts this way, a page-end modifier must be created on the page containing a chart to be printed. Due to limitations of the image tool in Lاسرنهت, charts currently use overlays for the layout. This means that the chart positions are static and defined by the coordinates given by the script.

Lاسرنهت includes a helper function to assist in making charts. By using this function, it is possible to create the majority of simple chart types by just changing a few parameters. The function has the following form:

```
createChart(type, posX, posY, width, height,
labelFontSize, yIntervals, margin, labels,
data, xTitle, yTitle, antiAliasing)
```

An explanation of the parameters is given in the table below.

Parameter	Type	Description
type	required	Defines the type of chart in the output. Available types include “Pie”, “Line”, “Bar”, “Area”, “Spline”, “StepLine” and “Trend”. The function also recognizes “3D” as a part of the string. Examples of this could be “Pie3D”, “Bar_3D” or “Area 3D”.
posX	required	X-axis position of the chart on the paper (in 0.1 mm).
posY	required	Y-axis position of the chart on the paper (in 0.1 mm).
width	required	Width of the chart on the paper (in 0.1 mm).
height	required	Height of the chart on the paper (in 0.1 mm).
labelFontSize	optional	Font size of the labels for the data to be displayed on the chart. <i>(default: 10)</i>
yIntervals	optional	Number of intervals on the y-axis of xy-charts. This parameter only affects xy-charts. <i>(default: 5)</i>
margin	optional	Defines the total margin of each axis. A margin of 200 makes 100 pixels margin on all sides of the chart. <i>(default: 300)</i>
labels	optional	The JobInfo name containing the labels for the data to be displayed in the chart. <i>(default: “Chart_Labels”)</i>
data	optional	The JobInfo name containing the data for display in the chart. <i>(default: “Chart_Data”)</i>
xTitle	optional	Defines the title of the x-axis of an xy-chart. <i>(default: “Chart_XTitle”)</i>
yTitle	optional	Defines the title of the y-axis of an xy-chart. <i>(default: “Chart_YTitle”)</i>

antiAliasing	optional	Defines antialiasing. The following constants are defined as valid values for this parameter: Chart.NoAntiAlias = 0 Chart.AntiAlias = 1 Chart.AutoAntiAlias = 2 (default: Chart.NoAntiAlias)
--------------	----------	--

To create complex charts, a more advanced script is required. The helper function is only intended for creating simple charts.

4 Scripting Reference.

4.1 Global Lasernet Script Functions

System scripts consist of some functions and variables that always are available. This does not necessarily mean that it is always safe to call the functions.

abortAfter

Description Aborts script execution after a certain timeout

`abortAfter (milliseconds);`

Parameters milliseconds: Integer

Returns None

Example

```
function runForever()
{
    while (true) {};
    { /* I will run forever */ }
}

abortAfter(3000, runForever)
/*
    Execute function runForever().
    Kill script if 3000 milliseconds timeout is exceeded
*/
```

addDestination

Description Adds *destination* or *alternative destination* to the list of destinations for the current job. If *replace* is *true* all current destinations are removed. If *alternative* is true the destination will be added as an alternative destination. Corresponds to:

`job.addDestination(destination, replace, alternative);`

Parameters destination : String

[replace : Boolean]

[alternative : Boolean]

Returns None

Example

```
addDestination('NULL', true);
addDestination('MyOutputPrinter', false, true);
```

addOverlay

Description Used in the Form Engine as a short cut to add the given overlay page to all pages. The *overlayName* must include the ".emf" extension, but should not include any path, since the overlay added must reside in the Overlays subdirectory of the configuration. If *replace* is *true* all current overlays are removed. If *editorOnly* is *true* the overlay will be visible in the Form Editor only.

The best place to use this function is the After Analysis modifier point.
Corresponds to:

```
pages[1-grab.numberOfPages].addOverlay(overlayName, replace,  
editorOnly);
```

Parameters

overlayName : String

[*replace* : Boolean]

[*editorOnly* : Boolean]

Returns None

Example `addOverlay('NameOfMyOverlay.emf', true, false);`

addPageOverlay

Description Used in the Form Engine as a short cut to add the given overlay page to the CurrentPage. The *overlayName* must include the ".emf" extension, but should not include any path, since the overlay added must reside in the Overlays subdirectory of the configuration. If *replace* is *true* all current overlays are removed. If *editorOnly* is *true* the overlay will be visible in the Form Editor only.

The best place to use this function is the On Page End modifier point.

```
pages[CurrentPage].addOverlay(overlayName, replace, editorOnly);
```

Parameters

overlayName : String

[*replace* : Boolean]

[*editorOnly* : Boolean]

Returns None

Example `addPageOverlay('NameOfMyOverlay.emf', true, false);`

addSubform

Description Used in the Form Engine as a shortcut to add a given subform to all pages. If *replace* is *true* all current subforms are removed. If *editorOnly* is *true*, the subform becomes visible in the Form Editor only.

The place to use this function is the Before Scripts modifier point.

Corresponds to:

```
pages[1-grab.numberOfPages].addSubform(subformName, replace, editorOnly);
```

Parameters

subformName : String

[replace : Boolean]

[editorOnly : Boolean]

Returns None

Example

```
addSubform('NameOfMySubform', true, false);
```

addPageSubform

Description Used in the Form Engine as a shortcut to add a given subform to the CurrentPage. If *replace* is *true*, all current subforms are removed. If *editorOnly* is *true*, the overlay becomes visible in the Form Editor only.

The best place to use this function is the On Page Start modifier point.

```
pages[CurrentPage].addSubform(subformName, replace, editorOnly);
```

Parameters

subformName : String

[replace : Boolean]

[editorOnly : Boolean]

Returns None

Example

```
addPageSubform('NameOfMySubform', true, false);
```

addPagePrinterProfile

Description Adds the *profileName* printer profile to *outputPort* for the current page.

outputPort can be:

- The **Name** of a Printer Output.
- The **Name** of a Printer Service Output.
- A reference to a printer that has been deployed to a particular print server (with backslashes escaped).
 - Syntax: `\\\\<print server name>\\<printer name>`

This function should only be used on page modifiers. It is a shortcut for:

```
pages[CurrentPage].addPrinterProfile(outputPort, profilename, true);
```

Parameters

outputPort : String

profileName : String

Returns None

Example

```
addPagePrinterProfile('Printer Output 1', 'Tray 2');
addPagePrinterProfile('\\\\\\UK Laser Printers\\Xerox Laser Printer', 'Tray 2');
```

addPrinterProfile

Description Adds the *profileName* printer profile to *outputPort* for all pages.

outputPort can be:

- The **Name** of a Printer Output.
- The **Name** of a Printer Service Output.
- A reference to a printer that has been deployed to a particular print server (with backslashes escaped).
 - Syntax: `\\\\\\<print server name>\\<printer name>`

This function should only be used on page modifiers.

Parameters

outputPort : String

profileName : String

[replace : Boolean]

Returns None

Example

```
addPrinterProfile('Printer Output 1', 'Tray 2', true);

addPrinterProfile('\\\\\\UK Laser Printers\\Xerox Laser
Printer', 'Tray 2', true);
```

Convert.toDate
Description

Converts a string in a specific format to a date. These expressions may be used for the format:

Expression Output

d	The day as a number without a leading zero (1 to 31)
dd	The day as a number with a leading zero (01 to 31)
ddd	The abbreviated localized day name (e.g. 'Mon' to 'Sun'). Uses the system locale to localize the name.
dddd	The long localized day name (e.g. 'Monday' to 'Sunday'). Uses the system locale to localize the name.
M	The month as a number without a leading zero (1 to 12)
MM	The month as a number with a leading zero (01 to 12)
MMM	The abbreviated localized month name (e.g. 'Jan' to 'Dec'). Uses the system locale to localize the name.
MMMM	The long localized month name (e.g. 'January' to 'December'). Uses the system locale to localize the name.
yy	The year as two digit number (00 to 99)
yyyy	The year as four digit number. If the year is negative, a minus sign is prepended in addition.

Parameters

date : String

format: String

Returns

Date

Example

```
var myDate = Convert.toDate("20141003", "yyyyMMdd");
```

Convert.toDateRP

Description Converts a string in a specific format to a date using a regional profile created in Lasernet:

Parameters

date : String
regional profile: String

Returns Date

Example

```
var myDate = Convert.toDateRP("2014-10-03", "US Format");
```

Convert.toNumber

Description Converts a string to a number. This function does not handle thousands group separators.

Parameters number : String

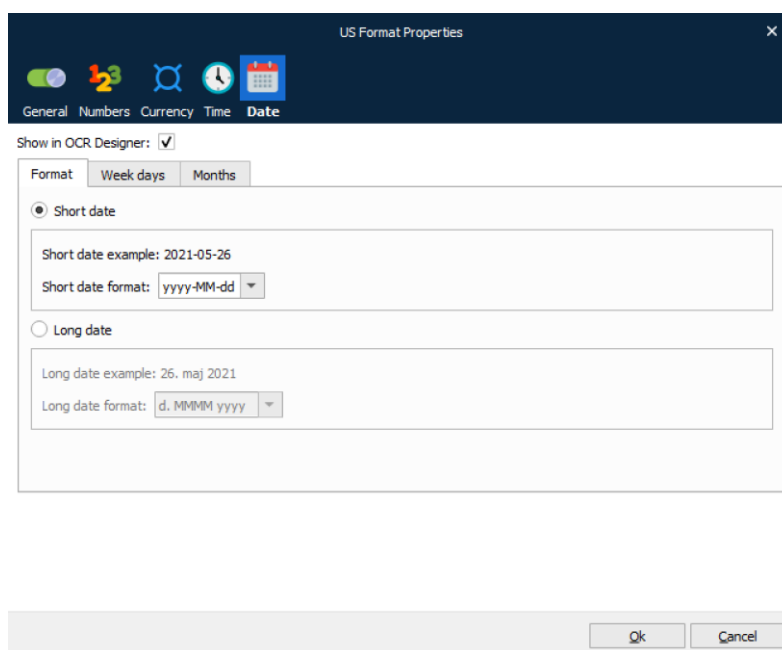
Returns Date

Example

```
var myNumber = Convert.toNumber("123456789.00");
```

Convert.toNumberRP

Description Converts a string in a specific format to a date using a regional profile created in Lasernetet:



Parameters number : String

regional profile: String

Returns Number

Example

```
var myNumber = Convert.toNumberRP("123,456,789.00", "US Format");
```

createDirectory

Description Creates the directory given in *path*. Does not work recursively.

Parameters path : String

Returns None

Example `createDirectory('c:\\MyFolder');`

createDirectories

Description Same as createDirectory but works recursively.

Parameters path : String

Returns None

Example `createDirectories('c:\\LaserNet\\MyFolder\\Output');`
Will create Lasernet, MyFolder and Output folders if they do not exist.

CurrentISOWeekNumber

Description Returns the ISO week number for the current date. See *ISOWeekNumber* for a description of what ISO Week Number is.

Corresponds to:

`ISOWeekNumber(new Date());`

Parameters None

Returns Number

Example

CurrentISOYear

Description Returns the ISO year of the current date. See *ISOYear* for a description of what ISO Year is.

Corresponds to:

```
ISOYear(new Date());
```

Parameters None

Returns Number

Example

getJobInfo

Description Returns the value of the JobInfo with the given *jobInfoName* and *index*. As in the method *getJobInfo* the *index* parameter has the default value of 0 (zero).

Parameters *jobInfoName* : String
[*index* : Number]

Returns String

Example Retrieves the value of the second value of MyJobInfo:

```
getJobInfo('MyJobInfo', 1);  
or
```

```
job.getJobInfo('MyJobInfo', 1);
```

ISOWeekNumber

Description Calculates the ISO week number of the given *date*. The ISO week number is calculated on the assumption that week 1 of a year is the first week with minimum 4 days, or in other words, the first week with a Thursday in it. This means that the ISO week number for 1st January can be either 1, 52 or 53. Similarly the ISO week number for 31st December can be either 1, 52 or 53.

Parameters *date* : Date

Returns Number

Example

ISOYear

Description Calculates the ISO Year of the given *date*. The ISO year is calculated on the assumption that week 1 of a year is the first week with minimum 4 days, or in other words, the first week with a Thursday in it. This means that the ISO year for 1st January can be either the actual year or the year before. The ISO year for 1st January 2006 is 2005, whereas the ISO year for 2nd January 2006 is 2006. Similarly the ISO year for 1st January 2005 is 2004. The ISO year 31st December 2007 is 2008.

Parameters date : Date

Returns Number

Example

loadJobInfoFromFile

Description Tries to load the content of the file given into the JobInfo *JobInfoName* if it succeeds. The function returns true if it succeeds, otherwise it returns false.

Parameters jobInfoName : String

filename : String

replace : Boolean

Returns Boolean

Example

```
if (loadJobInfoFromFile('MyJobInfo', 'c:\\myfile.txt', true)
== false)
{
    // do something if not loaded properly
}
```

saveJobInfoToFile

Description	Saves the contents of the JobInfo with the given <i>jobInfoName</i> to a file with <i>filename</i> name.
Parameters	jobInfoName : String filename : String
Returns	None
Example	<pre>saveJobInfoToFile('MyJobInfo', 'c:\\lasernet\\myfile.txt');</pre>

setJobInfo

Description	Sets the JobInfo <i>jobInfoName</i> to <i>value</i>. If <i>replace</i> is <i>true</i> all current JobInfos in the list are cleared. The default value of <i>replace</i> is <i>true</i>. Default loglevel is JOB_INFO. Corresponds to: <pre>job.setJobInfo(jobInfoName, value, replace, loglevel)</pre>
Parameters	jobInfoName : String value : String [replace : Boolean] [loglevel : Integer, where valid integers are; JobInfo: 12 Debug: 1 NoLog: 8]
Returns	None
Example	<pre>setJobInfo('MyJobInfo', 'value of my JobInfo');</pre>

Sum

Description

This function sums the values of any number of given arguments. It loops through the arguments and for each argument checks the type in the following way.

Checks whether the argument has a type property and the value of this is *Text*. In this case the argument is assumed to be a *InputRearrange* or *OutputRearrange* and the value to add is taken from the arguments *value* property.

Using the *typeof* operator checks whether the argument is of type object and if this is the case assumes that the argument is an array of *InputRearrange* or *OutputRearrange* and then sums the values of this array.

Using the *typeof* operator checks whether the argument is of type *Number* and if this is the case adds the number to the sum.

If none of the above, it assumes that the argument has a number property and adds the value of this.

Parameters

Any number of arguments

Returns

Number

Example

To sum all values of a conditional rearrange simply call *Sum* with the rearrange as its only parameter.

```
Sum(ItemPrice);
```


SumPage

Description Works much in the same way as Sum with the exceptions that if the first parameter is a number (typeof returns number) this number is taken to be the page on which to sum the arguments. If the first parameter is not a number CurrentPage is assumed instead. The rest of the parameters are evaluated in the same way as in the Sum function.

Parameters Any number of arguments

Returns Number

Example Sum of all CurrentRearrange values on page 3:

```
total = SumPage(3, CurrentRearrange);
```

Sum of all CurrentRearrange values on the current page:

```
total = SumPage(CurrentPage, CurrentRearrange);
```

or

```
total = SumPage(CurrentRearrange);
```

SumSubtotal

Description Sums on all pages up to and including the given first parameter or CurrentPage. If the first parameter is a number that number is assumed to be the page until which to sum. Otherwise CurrentPage is assumed. The rest of the parameters are evaluated in the same way as in the Sum function.

Parameters Any number of arguments

Returns Number

Example Sum of all CurrentRearrange values until the page the one being processed:

```
total = SumSubTotal(PreviousPage, CurrentRearrange);
```

Sum of all CurrentRearrange values until the current page:

```
total = SumSubTotal(CurrentPage, CurrentRearrange);
```

or

```
total = SumSubTotal(CurrentRearrange);
```

Sleep

Description Suspends the execution of the current thread until the time-out interval elapses. Useful in a busy loop, as it gives the CPU a chance to work on other threads.

Parameters milliseconds : Integer

Returns None

Example `Sleep(1000);`

Trim

Description Removes spaces from the end and beginning of the given text and returns the result. Very useful when retrieving values from rearranges, since their trimming settings is not used when returning values to script.

Parameters text : String

Returns String

Example `var text = Trim(CurrentRearrange.text);`

zipDirectory

Description Zips all files recursively in the given *directoryPath* and returns a ByteArray with the zip file.

Parameters directoryPath : String

Returns ByteArray

Example `File.writeBinary('c:\\test.zip',
zipDirectory('c:\\FilesToZip\\'));`

zipFilesInDirectory

Description Zips files recursively in the given *directoryPath* and returns a ByteArray with the zip file. Uses the *includeMatch* and *excludeMatch* regular expressions to determine what files to put in the zip file. *excludeMatch* has precedence over *includeMatch*.

Parameters

directoryPath : String

includeMatch : RegExp

excludeMatch : RegExp

Returns ByteArray

Example This example zips all files that do not contain .exe in it:

```
job.setJobData(zipFilesInDirectory("c:\\windows\\system32",
/./, /\.exe/i);
```

To not exclude any files write:

```
job.setJobData(zipFilesInDirectory("c:\\windows\\system32",
/./, ""));
```

4.2 Global Lasernet Script Classes

4.2.1 Job

Description

The Job class is used throughout Lascript to access the JobData being processed. An instantiation of the Job class: job almost always exists. The only situation in which the job object is not available is when creating module JobInfos. It is not possible to instantiate a Job from script.

Properties

type

Description Always returns the value 'Job'

Access Read

Returns String

Example

Methods

getJobData

Description Provides access to the primary JobData of the job being processed

Access Read

Returns ByteArray

Example Saves JobData to a file:

```
File.write("c:\\myfile.txt", job.getJobData());
```

hasJobInfo

Description Check if Job has a specific JobInfo

Access Read

Parameters jobInfoName : String

Returns Boolean

Example Saves JobData to a file:

```
if (job.hasJobInfo('FileName') == false)
{
    // job does not have the FileName jobinfo
};
```

jobInfoIndexOf

Description Returns index of a particular value within a JobInfo array

Access Read

Parameters jobInfoName : String

Value: String

[CaseSensitive: Boolean = true]

Returns Boolean

Example Returns the Content-Type HTTP header value from a request:

```
job.getJobInfo('RequestHeaderValues',
    job.jobInfoIndexOf('RequestHeaderNames', 'Content-Type')
);
```

addDestination

Description Adds a destination to where the Job should be sent, when the current module has finished its processing. If the replace parameter is true all existing destinations for the job are cleared. If replace is false a destination is added.

Parameters destination :String
[replace : Boolean = true]

Returns None

Example

```
job.addDestination('Printer');  
job.addDestination('File out, false');
```

deleteJobInfo

Description Removes all JobInfos with the given *jobInfoName* from the job entirely

Parameters jobInfoName :String

Returns none

Example `job.deleteJobInfo('MyJobInfo');`

getAllJobInfos

Description Returns a stringlist for JobInfo names assigned to a job.

Parameters none

Returns StringList

Example

```
var jobinfos = job.getAllJobInfos();
for (idx = 0; idx < jobinfos.length; idx++)
{
    var key = jobinfos[idx];
    var value = job.getJobInfo(key);
    Logger.logEvent(Debug, key + " = " + value);
}
```

getJobInfoCount

Description Returns the number of items in the JobInfo array with the given jobInfoName. If the JobInfo does not exist the method returns 0 (zero).

Please note that calling `job.getJobInfoCount()` is a relatively expensive operation, therefore it is better to catch the value in a local variable.

Parameters jobInfoName : String

Returns Number

Example

```
var count = job.getJobInfoCount('FileNames');
var index;
for (index = 0; index < count; index++)
{
    var filename = new String(job.getJobInfo('FileNames',
index));
    // Do something fascinating with that file
}
```

getJobInfo

Description Returns the value of the jobinfo with the given jobInfoName. If more than one jobinfo with that name exists a specific one can be chosen using the given index parameter. If no jobinfo with the given name exists an empty string is returned.

Parameters jobInfoName : String

Returns String

Parameters jobInfoName : String
[arrayIndex : Number = 0]

Returns String

Example `job.getJobInfo('MyJobInfo');`

getJobInfoBinary

Description Returns the value of the jobinfo with the given jobInfoName. If more than one jobinfo with that name exists a specific one can be chosen using the given index parameter. If no jobinfo with the given name exists an empty ByteArray is returned.

Parameters jobInfoName : String
[arrayIndex : Number = 0]

Returns ByteArray

Example `job.getJobInfoBinary('MyJobInfo');`

getSize

Description Returns the size of the JobData part of the Job.

Parameters None

Returns Number

Example `var size = job.getSize();
var size2 = job.size;`

load

Description A Job can be loaded via the load function on the job object.

It might be useful to load a range of jobs and run a range of checks on them.

Loading must be done with .lnjob files and can be either absolute or relative to the grab folder of the configuration. Function will return true if load was successful.

Parameters filename : string

[modulename : string]

Returns Boolean

Example

```
job.load('c:\\myfile1.lnjob');
job.load('myfile2.lnjob');
```

save

Description A Job can be saved via the save function on the job object.

It might be useful to save a range of jobs and run a range of checks on them.

Save must be done with .lnjob files and can be either absolute or relative to the grab folder of the configuration. Function will return true if save was successful.

Parameters filename : String

Returns Boolean

Example

```
job.save('c:\\myfile1.lnjob');
job.save('myfile2.lnjob');
```

setJobData

Description Provides access to the primary JobData of the job being processed

Parameters ByteArray or String

Returns None

Example Loads the content of myfile.txt into JobData jobInfo:

```
job.setJobData(File.read("c:\\myfile.txt"));
```


setJobInfo

Description Sets the value of the JobInfo with the given jobInfoName to value. If replace is true any existing JobInfo with the given name is deleted. Otherwise, a new value is appended. It is currently not possible to set a value for a specific index.

Parameters

- jobInfoName : String
- value : String
- [replace : Boolean = true]
- [loglevel : Integer = JobInfo, where valid integers are;
- JobInfo: 12
- Debug: 1
- NoLog: 8
-]

Returns none

Example `job.setJobInfo('MyJobInfo', 'A nice value');`

setJobInfoBinary

Description Sets the value of the JobInfo with the given jobInfoName to value. If replace is true any existing JobInfo with the given name is deleted. Otherwise, a new value is appended. It is currently not possible to set a value for a specific index.

The ByteArray value can be obtained from the readAll method of the Fileclass (for example).

Parameters

```

jobInfoName : String

value : ByteArray

[replace : Boolean = true]

[loglevel : Integer = JobInfo, where valid integers are;

JobInfo: 12

Debug: 1

NoLog: 8

]
```

Returns none

Example

```

var f = new File('c:\\test.txt');
if (f.open(IO_ReadOnly))
{
    job.setJobInfoBinary('ContentsOfTheFile', f.readAll());
    f.close();
}
```

substituteJobInfos

Description Returns a string built upon data from the Job and the specified expression. Expression can use hash marks to return data from Job in the tag format “#<jobinfoName>#”.

Parameters expression : String

Returns String

Example

```

var filename = job.substituteJobInfos('file_#JobID#');
```

4.2.2 Logger

Description

The Logger class is used throughout Lasernet to make custom logger messages. An instantiation of the Logger class: logger almost always exists. The only situation in which the logger object is not available is when creating module JobInfos. It is not possible to instantiate the Logger from script.

Methods

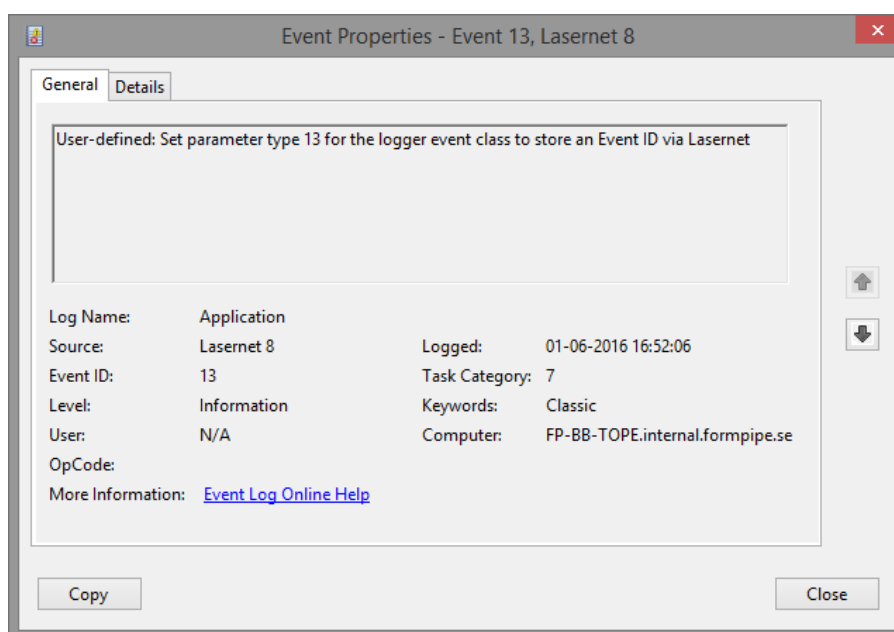
logEvent

Description

Creates a logger message. The message parameter contains the actual message, which is shown in the Message field in the monitor. The type parameter determines what kind of message is logged according to the following list.

EventIDs type 13 or “EventLog” are used to store messages in Windows Event Log and not in the monitor. Event log messages must be activated in the Lascript server settings before Windows Event Logs can be stored.

EventIDs type 14 or “SysLog” are used to store messages on Syslog server and not in the monitor. Syslog messages must be activated in the Lascript server settings before Syslogs can be stored.



(Screen shot taken from Lascript 8, but is similar in newer versions of Lascript)

Parameters

type : Number, where type can be one of the following;

Windows Event Log: 13

JobInfo: 12

Debug: 1

NoLog: 8

Values below 100 are reserved for future use, but values from 100 and above may be used for any custom message you want.

message :String

Returns

None

Example

```
logger.logEvent(Debug, 'The value is now: ' +  
job.getJobInfo('value'));  
logger.logEvent(999, 'My custom message');  
Logger.logEvent(EventLog, 'Store message in Windows Event Log');  
Logger.logEvent(13, 'Store message in Windows Event Log');  
Logger.logEvent(SysLog, 'Store message on Syslog server');  
Logger.logEvent(14, 'Store message on Syslog server');
```

4.2.3 Modifier

Description

The Modifier class is used throughout Lasernet to access the modifiers defined in Lascript Developer. These modifiers are made available through an associative array: modifiers which have one entry for each modifier defined. Use the name of the modifier to access it.

Properties

type	
Description	Always returns the value Modifier.
Access	Read
Returns	String
Example	

Methods

run	
Description	Executes the modifier using the given job. If the modifier has support for it, it can be run on the JobInfo with the given jobInfoName (parameter one), that after processing contains the new value. If the modifier has support for it, it can be run on the JobInfo with the given jobInfoName (parameter one) and (parameter two), where parameter two after processing contains the new value and leave the value for the first parameter as is. If no jobInfoName is given the standard JobData JobInfo is used.
Parameters	job : Job [jobInfoName : String] [jobInfoName : String]
Returns	none
Example	<pre>modifiers['Base64 Encoder'].run(job, 'MyJobInfo', 'MyNewJobInfo'); modifiers['Base64 Encoder'].run(job, 'MyJobInfo'); modifiers['Base64 Decoder'].run(job);</pre>

4.2.4 AzureStorageCommands

Description

The AzureStorageCommands class is used throughout Lasernet to access the Azure Storage commands defined in Lascript Developer. These commands are made available through an associative array: azurestoragecommands which have one entry for each command defined. Use the name of the azure storage command to access it.

Properties

type	
Description	Always returns the value Modifier.
Access	Read
Returns	String
Example	

Methods

run	
Description	Executes the azure storage command using the given job. The name is case-sensitive.
Parameters	job : Job [jobInfoName : String]
Returns	None
Example	<pre>azurestoragecommands['DownloadBlob'].run(job);</pre>

4.2.5 DatabaseCommands

Description

The DatabaseCommands class is used throughout Lasernet to access the database commands defined in Lascript Developer. These commands are made available through an associative array: databasecommands which have one entry for each command defined. Use the name of the database command to access it.

Properties

type	
Description	Always returns the value Modifier.
Access	Read
Returns	String
Example	

Methods

run	
Description	Executes the database command using the given job. The name is case-sensitive.
Parameters	job : Job [jobInfoName : String]
Returns	None
Example	<code>databasecommands['DatabaseLookup'].run(job);</code>

4.2.6 SharepointCommands

Description

The SharepointCommands class is used throughout Lasernet to access the Microsoft Sharepoint commands defined in Lasernet Developer. These commands are made available through an associative array: sharepointcommands which have one entry for each command defined. Use the name of the sharepoint command to access it.

Properties

type	
Description	Always returns the value Modifier.
Access	Read
Returns	String
Example	

Methods

run	
Description	Executes the sharepoint command using the given job. The name is case-sensitive.
Parameters	job : Job [jobInfoName : String]
Returns	None
Example	<pre>sharepointcommands['SharepointLookup'].run(job);</pre>

4.2.7 XMLTransformers

Description

The XMLTransformers class is used throughout Lasernet to access the XML Transformers defined in Lascript Developer. These commands are made available through an associative array: xmltransformers which have one entry for each command defined. Use the name of the xml transformer to access it. This class accesses XML Transformers defined in the XML Transformer Engine only. XML Transformers defined as modifiers are accessed via the modifier class.

Properties

type	
Description	Always returns the value Modifier.
Access	Read
Returns	String
Example	

Methods

run	
Description	Executes the XML Transformer command using the given job.
Parameters	job : Job [jobInfoName : String]
Returns	None
Example	<code>xmltransformers['XMLActions'].run(job);</code>

4.2.8 SystemInfo

Description

The SystemInfo class is used throughout Lasernetet to access various system information. It has only static properties and methods and is therefore never instantiated.

Static Properties

computerName

Description	Returns the name of the computer on which Lascript is currently running.
--------------------	--

Access	Read
---------------	------

Returns	String
----------------	--------

Example	<pre>var MyComputer = new String (SystemInfo.computerName);</pre>
----------------	---

Static Methods

getTempPath

Description	Returns the temporary path directory used by windows.
--------------------	---

Parameters	None
-------------------	------

Returns	String
----------------	--------

Example	<pre>SystemInfo.getTempPath();</pre>
----------------	--------------------------------------

getTempFilename

Description	Returns a temporary unique filename in the directory given. This does not need to be the directory returned by <i>SystemInfo.getTempPath</i> .
--------------------	--

Parameters	path : String
-------------------	---------------

Returns	String
----------------	--------

Example	<pre>SystemInfo.getTempFilename('c:\\lasernet');</pre>
----------------	--

4.2.9 ByteArray

Description

The ByteArray class is used for searching a bytearray or manipulating a bytearray.

Properties

type	
Description	Returns the value 'ByteArray'.
Access	Read
Returns	String
bytearray	
Description	Returns array of bytes.
Access	Read/Write
Returns	bytearray

Methods

append	
Description	Appends a string to the array.
Parameters	value: String
Returns	Nothing
Example	<pre>var bytes = new ByteArray(); bytes.append("anything");</pre>
Output	
count	
Description	Returns number of bytes in array.
Parameters	None
Returns	Integer
Example	<pre>Logger.logEvent(Debug, "Replaced " + bytes.count() + " bytes");</pre>
Output	
indexOf	

Description	Searches for position of another byte array within byte array.
Parameters	bytearray: ba Integer: from
Returns	Integer. First position of bytearray after from index position.
Example	<pre>var bytes = new ByteArray(); bytes.bytearray = job.jobdata; var search = new ByteArray(); search.append(["a", "b", "c"]); var position = bytes.indexOf(search.bytearray); if (position > -1) ...</pre>

Output

replace

Description	Replaces a range of bytes within byte array with another byte array.
Parameters	Integer: Start index Integer: length bytearray: after
Returns	Integer. First position of bytearray after from index position.
Example	<pre>var replace = new ByteArray(); replace.append(["a", "b", "c"]); bytes.replace(position, replace.count(), replace.bytearray); job.jobdata = bytes.bytearray;</pre>

Output

resize

Description	Resizes array to the specified size.
Parameters	Integer: New size
Returns	Nothing
Example	<pre>bytes.resize(10);</pre>

Output

4.2.10 GUID

Description

The GUID class represents a unique number.

Properties

uuid	
Description	Universally Unique Identifier.
Access	Read/Write
Returns	UUID
Example	<pre>var guid1 = new Guid(); var guid2 = new Guid(); if (guid1.uuid == guid2.uuid) Logger.logEvent(Debug, "Same"); else Logger.logEvent(Debug, "Different"); guid2.uuid = guid1.uuid; // assign the uuid of guid1 to guid2 if (guid1.uuid == guid2.uuid) Logger.logEvent(Debug, "Same"); else Logger.logEvent(Debug, "Different");</pre>
Output	Different Same

Methods

fromRfc4122	
Description	The RFC4122 representation of a GUID in a byte array.
Parameters	GUID: Byte array
Returns	
Example	<pre>var guid = new Guid(); var data = guid.toRfc4122(); var shortguid = Base64.encode(data); shortguid = shortguid.replace("/", "_"); shortguid = shortguid.replace("+", "-"); shortguid = shortguid.mid(0, 22); Logger.logEvent(Debug, "ShortGuid = '" + shortguid + "' from guid = '" + guid + "'");</pre>
Output	ShortGuid = '_H6-tFHyQQ650vX/F8L+0w' from guid = '{fc7eb4-51f2-40ee-b93a-f5ff17c2fed3}'

fromString

Description

Parameters GUID : String

Returns String

Example

```
var guid = new Guid();
guid.fromString('{fc7eb4-51f2-40ee-b93a-f5ff17c2fed3}');
Logger.logEvent(Debug, guid);
```

Output

```
{fc7eb4-51f2-40ee-b93a-f5ff17c2fed3}
```

toRfc4122

Description A byte array representation of a GUID in the RFC4122 format.

Parameters

Returns Byte array

Example

```
var shortguid = '_H6-tFHyQ0650vX/F8L+0w';
shortguid = shortguid.replace("_", "/");
shortguid = shortguid.replace("-", "+");
var data = Base64.decode(shortguid + "===");
var guid = new Guid();
guid.fromRfc4122(data);
Logger.logEvent(Debug, "Guid = '" + guid + "'");
```

Output

```
Guid = '{fc7eb4-51f2-40ee-b93a-f5ff17c2fed3}'
```

toString

Description Converts the GUID to a string representation.

Parameters

Returns String

Example

```
var guid = new Guid();
Logger.logEvent(Debug, guid.toString());
```

Output

```
{1e4408bb-5211-4179-8d8f-7230702730fe}
```

4.2.11 Hash

Description

The Hash class returns a hash value.

Methods

md5

Description	The md5 representation of a Hash in a byte array.
Parameters	Hash: Byte array
Returns	
Example	<pre>var data = new ByteArray; data.append("Return this string as a md5 hash"); var md5 = Hash.md5(data.bytearray);</pre>
Output	1105DD801EECB3F886C9BF87F851686

sha1

Description	The sha1 representation of a Hash in a byte array.
Parameters	Hash: Byte array
Returns	String
Example	<pre>var data = new ByteArray; data.append("Return this string as a sha1 hash"); var sha1 = Hash.sha1(data.bytearray);</pre>
Output	E66B4CE83BB5302FD6825480EE6CD6D68C73647F

sha256

Description	The sha256 representation of a Hash in a byte array.
Parameters	Hash: Byte array
Returns	String
Example	<pre>var data = new ByteArray; data.append("Return this string as a sha256 hash"); var sha1 = Hash.sha256(data.bytearray);</pre>
Output	E66B4CE83BB5302FD6825480EE6CD6D68C73647F

crc32

Description The crc32 representation of a Hash in a byte array.

Parameters

Returns Byte array

Example

```
var data = new ByteArray;  
data.append("Return this string as a crc32 hash");  
var crc32 = Hash.crc32(data.bytearray);
```

Output 1434952195

toByteArray

Description Converts a hex hash string, for example obtained via Hex::md5, Hex::sha1 or Hex::sha256, into a ByteArray.

Parameters Hash: String

Returns ByteArray

Example

```
var ba =  
Hash.toByteArray("E66B4CE83BB5302FD6825480EE6CD6D68C73647F");
```


4.2.12 MagicBytes

Description

The MagicBytes class returns a Boolean to identify a file format signature.

Methods

isDOC, isDOCX, isEMF, isImage, isODT, isPDF, isPNG, isRTF, isXLS, isXLSX, isZIP, isJSON, isEDIFACT, isJPEG, isTIFF

Description	Identifies data in a byte array.
Parameters	data: Byte array
Returns	Boolean
Example	<pre>var f = File('C:\\testfile'); if (f.open(IO_ReadOnly)) { if (MagicBytes.isDOCX(f.readAll())) Logger.logEvent(Debug, 'Is Word!'); if (MagicBytes.isImage(f.readAll())) Logger.logEvent(Debug, 'Is image!'); if (MagicBytes.isPDF(f.readAll())) Logger.logEvent(Debug, 'Is PDF!'); }</pre>

isDOC, isDOCX, isEMF, isImage, isODT, isPDF, isPNG, isRTF, isXLS, isXLSX, isZIP, isTIFF

Description	Identifies data using the given job or a JobInfo with a given jobInfoName. If no jobInfoName is given the standard JobData JobInfo is used.
Parameters	job: Job [JobInfoName: string]
Returns	Boolean
Example	<pre>if ((getJobInfo('Extension').lower() == ".docx") && (MagicBytes.isDOC(job))) Logger.logEvent(Debug, 'Is Word!'); if (MagicBytes.isPDF(job, 'MyJobInfo')) Logger.logEvent(Debug, 'Is PDF!');</pre>

4.2.13 Configuration

Description

The Configuration class returns information about the configuration.

Properties

configFilePath	
Description	Returns the path to the resource folder.
Parameters	
Returns	String
Example	<pre>Logger.logEvent(Debug, configuration.configFilePath);</pre>

revision	
Description	Returns the revision number of the Configuration. This only works when script is run server side - not in the Developer. In case of Server running with patch(es), it will show 'Patched' instead of a number.
Parameters	
Returns	String
Example	<pre>Logger.logEvent(Debug, configuration.revision);</pre>

Methods

getConnections	
Description	List of Connections of any type.
Parameters	
Returns	An array of Connections (4.6.5).
Example	<pre>var connections = configuration.getConnections(); for (idx = 0, count = connections.length; idx < count; idx++) { var connection = connections[idx]; Logger.logEvent(Debug, 'Connection=' + c.name + ' (Guid=' + c.guid + ', Description='+ c.description + ', type='+ c.type +')'); }</pre>
Output	A list of Connections with names, descriptions and types

getImages

Description List of images in the resource folder.

Parameters [includePath: boolean = true]

Returns StringList

Example

```
var images = configuration.getImages(false);
for (idx = 0, count = images.length; idx < count; idx++)
{
    var i = images[idx];
    Logger.logEvent(Debug, 'Image=' + i);
}
```

Output A list of images without path

getModules

Description List of Modules of any type.

Parameters

Returns An array of Modules (4.6.6).

Example

```
var modules = configuration.getModules();
for (idx = 0, count = modules.length; idx < count; idx++)
{
    var m = modules[idx];
    Logger.logEvent(Debug, 'Module=' + m.name + '
(Guid=' + m.guid + ', Type='+ m.type +')');
    if (m.type == 'Printer Service')
        Logger.logEvent(Debug, '(Printer=' +
m.printer + ', PrintServer=' + m.printserver +')');
}
```

Output Module=Printer Output (Guid=bba7f010-7d64-478b-ba2e-bd24934b8c0e, Type=Printer Output)
Module=Printer Service 1 (Guid=338d106e-b957-4cd7-8005-e7511576db8c, Type=Printer Service)
(Printer=HP Color LaserJet CP522X PCL6 Class Driver, PrintServer=Ballerup)

getOverlays

Description List of Overlays available.

Parameters

Returns StringList

Example	<pre>var overlays = configuration.getOverlays(); for (idx = 0, count = overlays.length; idx < count; idx++) { var o = overlays[idx]; Logger.logEvent(Debug, 'Overlay=' + o + ' (base64=' + Base64.encode(configuration.getOverlayThumbnail(o, 160, 100)).substring(0, 666) + ')'); }</pre>
Output	Overlay=ElectricityFirstMiddle.pub.lnemf (base64=iVB0...)

getOverlayThumbnail

Description	Returns a PNG thumbnail of a given Overlay with the specified dimension.
Parameters	overlay: String width: Integer height: Integer
Returns	Byte array
Example	See getOverlays for example.

getPrinterProfiles

Description	List of Printer Profiles in Configuration.
Parameters	
Returns	An array of Printers and their Printer Profiles (4.6.7).
Example	<pre>var printerprofiles = configuration.getPrinterProfiles(); for (idx = 0, count = printerprofiles.length; idx < count; idx++) { var printer = printerprofiles[idx]; Logger.logEvent(Debug, 'Printer=' + printer.name + ', PrinterName=' + printer.printername); for (idy = 0, county = printer.profiles.length; idy < county; idy++) { var p = printer.profiles[idy]; Logger.logEvent(Debug, 'PrinterProfile=' + p.name); } }</pre>

	}
Output	Printer=Printer Output, PrinterName=HP PrinterProfile=Portrait PrinterProfile=Landscape
getServers	
Description	List of Servers in Configuration.
Parameters	
Returns	An array of Servers (4.6.9).
Example	<pre>var servers = configuration.getServers(); for (idx = 0, count = servers.length; idx < count; idx++) { var s = servers[idx]; Logger.logEvent(Debug, 'Server=' + s.name + ' (Hostname=' + s.hostname + ', Port='+ s.port +')'); }</pre>
Output	Server=Master (Hostname=Master, Port=3279) Server=Default (Hostname=localhost, Port=3279)

4.3 Form Engine Variables

Variable Name	Description
CurrentRearrange	The CurrentRearrange variable is generally available in all scripts called from named rearranges. It contains a reference to the rearrange from which the script is being called. It is of the type InputRearrange.
Line	The Line variable is generally available in all scripts called from named rearranges. It contains the line number on which the CurrentRearrange is placed, for the pattern on which the rearrange is placed. It is used to refer to other named rearranges on the same pattern. The Line variable starts with line number set to 0 (zero) <pre>function CalcPrice() : Number { return Price[Line].number * Quantity[Line].number; }</pre>
GlobalLine	The GlobalLine variable is generally available in all scripts called from named rearranges. It contains the global line number on which the CurrentRearrange is placed. This line number includes all patterns. The GlobalLine variable starts with line number set to 1 (one)
masterJob	Available from scripts run on Sheet Modifiers and rearranges. <i>masterJob</i> is of type <i>Job</i>. It is from this that the <i>job</i> is copied when each sheet starts processing. This means that you can use <i>masterJob</i> to copy data to the next sheet. <pre>masterJob.setJobInfo("JobInfoName", "Value");</pre>

4.4 Form Engine Classes

4.4.1 Grab

Description

The Grab class is used in the Form Engine to provide basic access to the input and output grab areas. The Grab class is automatically instantiated by the grab object. It is not possible to instantiate the Grab from script.

Properties

currentPage	
Description	Returns the number of the output page currently being handled. Pages are numbered from 1. This property returns the same value as the variable CurrentPage.
Access	Read
Returns	Number
Example	<pre>var PageCurrent = grab.currentPage;</pre>

currentSheet

Description	Returns the number of the current sheet being handled. Sheets are numbered from 1.
Access	Read
Returns	Number
Example	<pre>var SheetCurrent = grab.currentSheet;</pre>

lineCount

Description	Returns the number of lines in the input grab area. Can be used in connection with the getText method to loop through the lines in the grab area.
Access	Read
Returns	Number
Example	

maxWidth

Description	Returns the width of the widest line in the input grab area. The width of a line is defined as the rightmost column with a non-space character.
Access	Read
Returns	Number
Example	

numberOfPages

Description	Returns the total number of pages generated for text output.
Access	Read
Returns	Number
Example	

type

Description Always returns the value *Grab*.

Access Read

Returns String

Example

conditionalHeight

Description

Access Read

Returns Number

Example

conditionalStart

Description

Access Read

Returns Number

Example

conditionalEnd

Description

Access Read

Returns Number

Example

conditionalFooter**Description**

Access	Read
---------------	------

Returns	Number
----------------	--------

Example**Methods****getText**

Description	Retrieves text directly from the input grab area. If height is bigger than 1 then multiple lines will be returned separated by CR (hex 0D). The last line will not have an ending CR. All lines will be padded with spaces up to the given width.
--------------------	---

Parameters	line : Number column : Number width : Number [height : Number = 1]
-------------------	---

Returns	String
----------------	--------

Example

setChartData

Description Sets the ChartData *ChartDataName* to *value*. If *replace* is *true* all current ChartData in the list is cleared. The default value of *replace* is *true*.

Parameters

ChartDataName : String

value : String

[replace : Boolean]

Returns None

Example

```
grab.setChartData("Chart_Data", 100.0, true);
grab.setChartData("Chart_Data", 200.0, false);
grab.setChartData("Chart_Data", 300.0, false);

grab.setChartData("Chart_Label", "2013", true);
grab.setChartData("Chart_Label", "2014", false);
grab.setChartData("Chart_Label", "2015", false);
```

getChartData

Description Returns the value of the ChartData with the given ChartDataName. If more than one ChartData with that name exists a specific one can be chosen using the given index parameter. If no ChartData with the given name exists an empty string is returned.

Parameters

ChartDataName : String

[index : Number]

Returns String

Example Retrieves the value of the second array of Chart_Data:

```
grab.getChartData('Chart_Data', 1);
```

getChartDataCount

Description Returns the number of items in the ChartData array with the given ChartDataName. If the ChartData does not exist the method returns 0 (zero).

Parameter ChartDataName : String

Returns Number

Example

```
var count = grab.getChartDataCount("Chart_Data");
var i;
for (i = 0; i < count; i++)
{
    grab.setChartData("Chart_Data_2",grab.getChartD
ata("Chart_Data", i), false);
}
```

deleteChartData
Description

Removes all ChartData with the given *ChartDataName*.

Parameters

ChartDatatName : String

Returns

None

Example

```
grab.deleteChartData("MyChartData");
```

4.4.2 InputRearrange

Description

For each input rearrange and insert text with a name, Lasernet creates an array of InputRearrange objects. The array can be accessed using the object name set in the input rearrange dialog. Rearranges without a name are not accessible from script. This array principle makes it possible to make short scripts like Sum(MyRearrange). Objects of type InputRearrange are only available in the Form Engine.

Properties
bold
Description

Gets and sets the bold property of the font used for rendering the rearrange.

Access

Read, Write

Returns

Boolean

Example

```
myrearrange.bold = true;
```

color
Description

Gets and sets the color property of the font used for rendering the rearrange.

Access

Read, Write

Returns

Color

Example

```
var mycolor = new Color();
mycolor.setRgb(0x00ff00); // one way to set color to green
myrearrange.color = mycolor;
```

family

Description Gets and sets the font name used for rendering the rearrange.

Access Read, Write

Returns String

Example

```
myrearrange.family = "Courier New";
```

italic

Description Gets and sets the italic property of the font used for rendering the rearrange.

Access Read, Write

Returns Boolean

Example

```
myrearrange.italic = true;
```

name

Description Returns the object name for the rearrange as set in Lasernet Form Editor.

Access Read

Returns String

Example

```
logger.logEvent(0, 'Name of Current Rearrange = '+CurrentRearrange.name);
```

number

Description Returns the value of the rearrange as a number. The number is parsed from the original text using the settings for input number formats. If a number cannot be parsed 0 (zero) is returned.

Access Read

Returns Number

Example

```
If (myRearrange.number == 0) return 'Empty';
```

page

Description Always returns 0 (zero).

Access Read

Returns String

Example

pointSize

Description Gets and sets the pointSize property of the font used for rendering the rearrange.

Access Read, Write

Returns Number

Example `myrearrange.pointSize = 12;`

strikeOut

Description Gets and sets the strike out property of the font used for rendering the rearrange.

Access Read, Write

Returns Boolean

Example `myrearrange.strikeOut = false;`

text

Description Returns the original text for the rearrange – that is, before it has been changed by any scripts or other means. The text is not trimmed even if this has been selected in the Designer. The text is exactly as it appears in the left side grab.

Access Read

Returns String

Example `myrearrange.family = "Courier New";`

type

Description Returns the type of the rearrange. Currently always returns Text.

Access Read

Returns String

Example `myrearrange.family = "Courier New";`

underline

Description Gets and sets the underline property of the font used for rendering the rearrange.

Access Read, Write

Returns Boolean

Example `myrearrange.underline = true;`

value

Description Returns the value of the rearrange, for example, after a script has manipulated the value. Be aware that calling the `.value` property on a rearrange with a script on it may result in an infinite loop if that script directly or indirectly refers to itself!

Access Read

Returns String

Example `myrearrange.family = "Courier New";`

weight

Description

Access Read, Write

Returns Number

Example

orientation**Description**

Access	Read, Write
---------------	-------------

Returns	Number
----------------	--------

Example

4.4.3 OutputRearrange

Description

For each output rearrange and insert text with a name, Lasernet creates an array of OutputRearrange objects. The array can be accessed using the object name set in the output rearrange dialog. Rearranges without an object name are not accessible from script. This array principle makes it possible to set properties for an output rearrange. Objects of type OutputRearrange are only available in the Form Engine.

Properties

backgroundColor

Description	Gets and sets the background color property of the font used for rendering the rearrange.
--------------------	---

Access	Read, Write
---------------	-------------

Returns	Boolean
----------------	---------

Example	<pre>myoutputrearrange.backgroundColor = 0x00ff00; // one way to set color to green;</pre>
----------------	--

borderColor

Description	Gets and sets the border color property of the font used for rendering the rearrange.
--------------------	---

Access	Read, Write
---------------	-------------

Returns	Color
----------------	-------

Example	<pre>myoutputrearrange.borderColor = 0xff0000; // one way to set color to red;</pre>
----------------	--

borderWidth

Description	Gets and sets the border width in points used for rendering the rearrange.
--------------------	--

Access	Read, Write
---------------	-------------

Returns	String
----------------	--------

Example	<pre>myoutputrearrange.borderWidth = 10;</pre>
----------------	--

marginSize

Description	Gets and sets the margin size in millimeters used for rendering the rearrange.
--------------------	--

Access	Read, Write
---------------	-------------

Returns	Boolean
----------------	---------

Example	<pre>myoutputrearrange.marginSize = 100;</pre>
----------------	--

4.4.4 Sheet

The Sheet class is used in the Form Engine to access the sheet currently being processed. The Sheet class is automatically instantiated in the sheetobject. The Sheet class cannot be instantiated from script.

Properties

evaluate

Description

On a Sheet Start Modifier this property can be set to false to stop evaluating/analyzing the sheet. No processing will be done for the sheet and nothing will be printed for the sheet. Only Sheet Start modifiers will be run for that sheet.

The usual criteria based on JobInfos can be used to determine whether to run the modifier and thereby hinder the evaluation of the sheet.

Access

Read, Write

Returns

Boolean

Example

```
sheet.evaluate = false;
```

name

Description

Returns the name of the sheet as set in the Form Designer.

Access

Read

Returns

String

Example

```
logger.logEvent(0, 'Name of the sheet = ' + sheet.name);
```

type

Description

Always returns the value 'Sheet'.

Access

Read

Returns

String

regionalProfile

Description	Returns and sets the name of the associated regional profile, which is used for rendering output. If there is no regional profile with the name chosen, the output rendering will fail with an error and no output will be shown for the affected rearranges. Note: Value can only be read if it is set/written by the script. Value cannot be read if set from the interface after introducing the list of values with criteria.
Access	Read, Write
Returns	String
Example	<pre>sheet.regionalProfile = "Default";</pre>

Functions

fontStyle

Description	Returns a new FontStyle object, which allows you to manipulate with the parameters of the selected style name (for more information see Chapter 4.6.3 FontStyle).
Parameters	StyleName :String
Access	Read
Returns	FontStyle
Example	<pre>var basicFont = sheet.fontStyle("Basic");</pre>

shapeStyle

Description	Returns a new ShapeStyle object, which allows you to manipulate with the parameters of the selected style name (for more information see Chapter 4.6.4 ShapeStyle).
Parameters	StyleName :String
Access	Read
Returns	ShapeStyle
Example	<pre>var basicShape = sheet.shapeStyle("Basic");</pre>

4.4.5 Form

The Form class is used in the Form Engine to access the form currently being processed. The Form class is automatically instantiated in the form object. The form class cannot be instantiated from script.

Properties

name	
Description	Returns the name of the form as set in the Form Designer.
Access	Read
Returns	String
Example	<pre>logger.logEvent(0, 'Name of the sheet = ' + form.name);</pre>

type	
Description	Always returns the value 'Form'.
Access	Read
Returns	String
Example	

regionalProfile	
Description	Returns and sets the name of the associated regional profile, which is used for parsing input. If there is no regional profile with the name chosen, the input parsing will fail with an error for the affected rearranges. Note: Value can only be read if it is set/written by the script. Value cannot be read if set from the interface after introducing the list of values with criteria.
Access	Read, Write
Returns	String
Example	<pre>form.regionalProfile = "Default";</pre>

Functions

fontStyle

Description Returns a new FontStyle object, which allows you to manipulate with the parameters of the selected style name (for more information see Chapter 4.6.3 FontStyle).

Parameters StyleName :String

Access Read

Returns FontStyle

Example

```
var basicFont = form.fontStyle("Basic");
```

shapeStyle

Description Returns a new ShapeStyle object, which allows you to manipulate with the parameters of the selected style name (for more information see Chapter 4.6.4 ShapeStyle).

Parameters StyleName :String

Access Read

Returns ShapeStyle

Example

```
var basicShape = form.shapeStyle("Basic");
```

4.5 Global Script Classes

4.5.1 ActiveXObject

Description

The ActiveXObject class is used to instantiate simple ActiveX objects in script. It is not possible to describe properties and methods since these vary from one object to another.

It is not possible to use properties and methods that return another ActiveXObject. Only the more simple types are supported.

Constructors

ActiveXObject

Description	Returns a new ActiveXObject based on the classID. If <i>servername</i> is specified the ActiveX object will be instantiated on that server. This presumes that Lasernet runs under an account that has the required privileges.
--------------------	---

Parameters	classID : String [servername : String]
-------------------	---

Returns	ActiveXObject
----------------	---------------

Example	<pre>var ax = new ActiveXObject('Microsoft.Word');</pre>
----------------	--

4.5.2 File

Description

The File class is used to access files in the file system. It is possible to both read and write files.

```
function FileWriteTest()
{
    var f = new File('c:\\file.txt');

    if (f.open(IO_WriteOnly, false))
    {
        f.write('This is a test \n');
        f.write('This is a test \n');
        f.write('This is a test \n');
        f.write('This is a test \n');
        f.close();
    }
}
```

As can be seen in the above example two backslashes ('\\') are used as a separator in the filename. The reason for this is that strings in JavaScript can have special symbols encoded using backslash – therefore two backslashes in a row actually evaluates to one backslash thereby giving us the right path.

Properties

type	
Description	Returns the value 'File'.
Access	Read
Returns	String
Example	

atEnd**Description**

Returns true if the file position is at the end of the file.

This is primarily used when reading files. Using this property it is possible to continue reading the contents of a file until no more data is available. The return value of this property is undefined before the file is opened.

Access

Read

Returns

Boolean

Example

```
function fileReadTest()
{
    var f = new File('c:\\file.txt');
    if (f.open(IO_ReadOnly, false))
    {
        // Continue reading until we reach end of file
        while (!f.atEnd())
        {
            var line = f.readLine(1000);
            job.setJobInfo('Lines', line, false);
            job.setJobInfo('LineLength',
f.bytesRead, false);
        }
        f.close();
    }
}
```


bytesRead

Description Returns the number of bytes that the last call to readLine read from the file after a call to readLine(). If readLine() has not been called the property returns 0 (zero).

Access Read

Returns Number

Example

```
function fileReadTest()
{
    var f = new File('c:\\file.txt');
    if (f.open(IO_ReadOnly, false))
    {
        // Continue reading until we reach end of file
        while (!f.atEnd())
        {
            var line = f.readLine(1000);
            job.setJobInfo('Lines', line, false);
            job.setJobInfo('LineLength',
f.bytesRead, false);
        }
        f.close();
    }
}
```

created

Description Gets the created date time for the file.

Access Read

Returns Date

Example

```
var f = new File('c:\\file.txt');
var t = f.created;
```

isDir

Description Returns *true* if the file referred to is in fact a directory, *false* otherwise.

Access Read

Returns Boolean

Example

```
var f = new File('c:\\windows');
if (f.isDir)
{
    // Do some magic
}
```

isFile

Description Returns *true* if the file referred to is in fact a file, *false* otherwise.

Access Read

Returns Boolean

Example

```
var f = new File('c:\\file.txt');
if (f.isFile)
{
    // Do some magic
}
```

isReadable

Description Returns *true* if the file is readable, *false* otherwise.

Access Read

Returns Boolean

Example

```
var f = new File('c:\\file.txt');
if (f.isReadable)
{
    // Do some magic
}
```

isWritable

Description Returns *true* if the file is writable, *false* otherwise.

Access Read

Returns Boolean

Example

```
var f = new File('c:\\file.txt');
if (f.isWritable)
{
    // Do some magic
}
```

lastModified

Description Gets the last modified date time for the file.

Access Read

Returns Date

Example

```
var f = new File('c:\\file.txt');
var lastModifiedDate = f.lastModified;
```

lastRead

Description Gets the last read date time for the file.

Access Read

Returns Date

Example

```
var f = new File('c:\\file.txt');
var lastReadDate = f.lastRead;
```

name

Description

Sets and gets the filename of the file being handled. Do not set the filename after the file has been opened – the behaviour is undefined.

In general you should always use the full path of the file. Not using a full path will most likely succeed, but the path will then be dependent on which directory is the current directory for the entire Lasernet application. This directory may change at any time.

Access

Read, Write

Returns

String

Example

size

Description

Returns the size in bytes of the file. The file does not have to be open to get the size.

Access

Read

Returns

Number

Example

```
var f = new File('c:\\file.txt');
var fileSizeInBytes = f.size;
```

Static methods

copy

Description

Copies a source file to a new location and/or filename. Function returns true on success and false on error.

Parameters

sourceFilename : String

destinationFilename : String

[allowOverwrite : Boolean = true]

Returns

Boolean

Example

```
File.copy('c:\\file1.txt', 'c:\\file2.txt', true);
```

exists

Description Returns true if the file with filename, which should be the full path to the file, exists.

Parameters filename : String

Returns Boolean

Examples

```
if (File.exists("c:\\myfile.txt"))
{
    // Do your magic with that file.
}
```

isFile

Description Returns true if the file with filename, which should be the full path to the file, is a file.

Parameters filename : String

Returns Boolean

Examples

```
if (File.isFile("c:\\myfile.txt"))
{
    // Do your magic now you know it's a file.
}
```

isDir

Description Returns true if the file with filename, which should be the full path to the file, is a directory.

Parameters filename : String

Returns Boolean

Examples

```
if (File.isDir("c:\\myfolder"))
{
    // Do your magic now you know it's a folder.
}
```

move

Description Moves a source file to a new location and/or filename. Can also be used to rename a file. Function returns true on success and false on error.

Parameters

- sourceFilename : String
- destinationFilename : String
- [allowOverwrite : Boolean = true]

Returns Boolean

Example `File.move('c:\\file1.txt', 'c:\\file2.txt', true);`

read

Description Reads the entire contents of the file with the given filename into a String and returns that. Throws a File.read() exception if it for some reason fails to read the file.

Parameters filename : String

Returns String

Example `var content = File.read("c:\\myfile.txt");`
Note that the file has to be in a normal ANSI code page. If you need more a more advanced code page like UTF8, you must open the file and read the contents manually.

```
var f = new File('c:\\test.txt');
f.setCodec('UTF8');
if (f.open(IO_ReadOnly))
{
    var text = new String (f.readAllText());
    f.close();
}
```

readBinary

Description Reads the entire contents of the file with the given filename into a ByteArray and returns that. Throws a File.readBinary() exception if it for some reason fails to read the file.

Parameters filename : String

Returns ByteArray

Example

```
var content = File.readBinary("c:\\myfile.txt");
```

remove

Description Returns true if the file with filename, which should be the full path to the file, was removed.

Parameters filename : String

Returns Boolean

Examples

```
if (File.remove("c:\\myfile.txt"))
{
    // Do your magic now you know the file was
    removed.
}
```

size

Description Reads the size of the file with the filename given and returns the value. Throws a File.size() exception and -1 as value if it for some reason fails to read the file.

Parameters filename : String

Returns Number

Example

```
var long = File.size("c:\\myfile.txt");
```

write

Description Writes a string to a file. Returns true if content was successfully written to file.

Parameters

filename : String

content : String

Returns Boolean

Examples

```
if (File.write("c:\\myfile.txt", "Lasernet Scripting"))
{
    // Do your magic now you know content was
    written.
}
```

writeBinary

Description Writes an array of bytes to a file. Returns true if data was successfully written to file.

Parameters

filename : String

data : ByteArray

Returns Boolean

Examples

```
if (File.writeBinary("c:\\myfile.txt", job.jobdata))
{
    // Do your magic now you know jobdata was
    written.
}
```

Methods

close

Description Closes the file so no more data can be read from or written to it until it has been opened again by a call to open().

Parameters None

Returns None

Example

```
var f = new File('c:\\test.txt');
if (f.open(IO_ReadOnly))
{
    var text = new String (f.readAllText());
    f.close();
}
```


exists

Description Returns true if the file with filename exists. If no filename is given the filename set by the constructor or the name property is used. Useful for making sure that a file actually exists before opening it.

Usually you would choose to use the static method `File.exists()` instead.

Parameters [Filename : String = ""]

Returns Boolean

Example

```
var f = new File('c:\\test.txt');
if (f.exists())
{
    // Handle the file
}
```

flush

Description Flushes any buffered data to the file. Returns true if successful; otherwise returns false.

Parameters None

Returns Boolean

Example

open

Description Opens the file with the given mode and exclusive settings. *mode* can be one of the following:

IO_RAW // raw access (not buffered)

IO_Async // asynchronous mode

IO_ReadOnly // readable device

IO_WriteOnly // writable device

IO_ReadWrite // read+write device

IO_Append // write+append device

IO_Truncate // truncate device

IO_Translate // translate CR+LF

Set exclusive to true to make sure that you have exclusive access to the file. This only works on existing files. If the file does not already exist the exclusive setting does not have any effect. The exclusiveTimeout value is used to set how long to wait for exclusive access to the file. The value is in milliseconds. If exclusive access cannot be obtained before this timeout open returns false.

Parameters

mode : Number

[exclusive : Boolean = true]

[exclusiveTimeout : Number = 10000]

Returns Boolean

Example

```
var f = new File('c:\\test.txt');
if (f.open(IO_ReadOnly))
{
    // read the file

    f.close();
}
```

readAll

Description Reads the entire file into a ByteArray and returns the ByteArray. This can be used to load data from a file and put it into a JobInfo.

Parameters None

Returns	ByteArray
----------------	-----------

Example	<pre>var f = new File('c:\\test.txt'); if (f.open(IO_ReadOnly)) { job.setJobInfoBinary('ContentsOfTheFile', f.readAll()); f.close(); }</pre>
----------------	--

readAllText

Description	Reads the entire file into a String and returns the string. If a codec has been set with setCodec() this codec is used to read the file. If not it is assumed that the file is a standard ANSI file. The property bytesRead reflects how many characters have been read when a codec has been set. Otherwise it returns how many bytes have been read, which will correspond to the number of characters.
--------------------	---

Parameters	None
-------------------	------

Returns	String
----------------	--------

Example	<pre>var f = new File('c:\\test.txt'); if (f.open(IO_ReadOnly)) { job.setJobInfo('ContentsOfTheFile', f.readAllText()); f.close(); }</pre>
----------------	--

readLine

Description	Reads a line from the file. The returned String will include the CR/LF characters if any are read and a codec has not been set with setCodec. If an end of line is not found before maxLengthcharacters has been read, only a maxlength character is read. The property bytesRead will contain the number of bytes actually read. To make sure that you always read to the end of a line – or end of line, if no end of line exists you can use the size property.
--------------------	--

Parameters	maxLength : String
-------------------	--------------------

Returns	String
----------------	--------

Example

```
var f = new File('c:\\test.txt');
if (f.open(IO_ReadOnly))
{
    var line = new String (f.readLine(f.size));
    f.close();
}
```

remove
Description

Tries to remove (delete) the file with filename and returns true if it succeeds. If no filename is given the filename set by the constructor or the name property is used. The file should not be open when trying to remove it. The filename must contain the full path to the file.

Parameters

[filename : String = ""]

Returns

Boolean

Example

```
var f = new File('c:\\test.txt');
if (f.remove())
    logger.logEvent(0, 'File deleted');
```

reset
Description

Seeks to the start of the file. Returns true on success; otherwise returns false (for example, if the file is not open).

Parameters

None

Returns

Boolean

Example
seek
Description

This function sets the current position to *pos*, returning true on success, or false if an error occurred.

Parameters

pos : Integer

Returns

Boolean

Example

setCodec

Description Sets the codec to use in the readLine(), write() and readAllText() methods.

Parameters codecname : String

Returns Boolean

Example

```
var f = new File('c:\\test.txt');
f.setCodec('UTF8');
if (f.open(IO_ReadOnly))
{
    var text = new String (f.readAllText());
    f.close();
}
```

write

Description Writes the given text to the file. The file must be open to be able to write to it.

Parameters text : String

Returns None

Example

```
var f = new File('c:\\test.txt');
if (f.open(IO_WriteOnly))
{
    f.write('Text to write in my file');
    f.close();
}
```

Example

```
/* (f.open(IO_Append)) was required to append text to a file,
in older versions of Lasernet. f.open(IO_WriteOnly+IO_Append))
is required to append text to a file in Lasernet 10 and newer
*/
var f = new File('c:\\test.txt');
if (f.open(IO_WriteOnly+IO_Append))
{
    f.write('Text to append in end of my file');
    f.close();
}
```

writeBinary

Description	Writes the given data to the file. The file must be open to be able to write to it. This function is usually used to write the contents of a job to a file.
Parameters	data : ByteArray
Returns	None
Example	<pre>var f = new File('c:\\test.txt'); if (f.open(IO_WriteOnly)) { f.writeBinary(job.getJobInfoBinary("JobData")); f.close(); }</pre> This set of scripts can be easily replaced by a global script function; <code>saveJobInfoToFile(jobInfoName, fileName);</code> Or a static function call; <pre>File.writeBinary('c:\\test.txt', job.getJobInfoBinary('JobData'));</pre>

4.5.3 Zip

Description

The Zip class is used to create and manipulate zipfiles from script and is meant as an alternative or supplement to the Zip Modifier, which is somewhat limited in its functionality. The zip file is created and manipulated in memory, which sets some limitations on file size.

Properties

count

Description	The number of files in the zip file.
--------------------	--------------------------------------

Access	Read
---------------	------

Returns	Number
----------------	--------

Example

type

Description	Always returns the value 'Zip'.
--------------------	---------------------------------

Access	Read
---------------	------

Returns	String
----------------	--------

Example

Methods

addFile

Description	Adds the given data to the zip file with the given <i>filename</i> .
--------------------	--

Parameters	Data : ByteArray
-------------------	------------------

	Filename : String
--	-------------------

Returns	none
----------------	------

Example

```
var zipper = new Zip();
zipper.newZipFile();
zipper.addFile(job.jobdata, "preview.pdf");
```

addFileFromDisk

Description	Adds a file with the given <i>filename</i> from disk into the zip file. It is given the <i>filenameInZip</i> in the zipfile.
Parameters	filename : String filenameInZip: String
Returns	None
Example	<pre>var zipper = new Zip(); zipper.newZipFile(); zipper.addFileInZip("c:\\preview.pdf", "preview.pdf");</pre>

extractArchiveToDisk

Description	Extracts all the files in the archive to disk. The given <i>filepath</i> is used as root for the files, which all are extracted with full path.
Parameters	filepath : String
Returns	Boolean
Example	<pre>var zipper = new Zip(); if (zipper.loadZipFileFromDisk("d:\MyZipFile.zip")) { if (zipper.extractArchiveToDisk("c:\\extractfolder")) // extract OK }</pre>

extractFileToDisk

Description	Extracts a file from the archive onto disk. All the files are numbered from 0 to <i>count</i> -1.
Parameters	Index : Number filepath : String
Returns	Boolean
Example	<pre>... if (zipper.extractFileToDisk(1, "c:\\extractfolder")) // extract OK</pre>

getZipFile

Description	Returns the entire zip file as a ByteArray. You can use this to put the zip file into a JobInfo.
Parameters	None
Returns	ByteArray
Example	<pre>var zipper = new Zip(); ... job.setJobData(zipper.getZipFile());</pre>

loadZipFileFromDisk

Description	Loads the zip file from disk with the given <i>filename</i> . Returns true if the zip file was successfully loaded.
Parameters	Filename : String
Returns	Boolean
Example	<pre>var zipper = new Zip(); zipper.loadZipFileFromDisk("d:\MyZipFile.zip");</pre>

newZipFile

Description	Prepares a new zip file. This function should always be called before adding files to a new archive.
Parameters	None
Returns	None
Example	<pre>var zipper = new Zip(); zipper.newZipFile();</pre>

saveZipFile

Description	Saves the zip file to disk. Uses the given <i>filename</i> for the zip file.
Parameters	Filename : String
Returns	None
Example	<pre>var zipper = new Zip(); zipper.newZipFile(); zipper.addFile(job.getJobInfoBinary("JobData"), "preview.pdf"); zipper.saveZipFile("d:\MyZipFile.zip");</pre>

4.6 JavaScript Global Classes

4.6.1 Color

Description

The Color class is used for representing colors.

Methods

setRgb

Description

Sets the color code of the color.

The value is a bitmask on the form 0xRRGGBB, where RR=red, GG=green, and BB=blue, all as hexadecimal digits.

Parameters

Colorcode : Number

Returns

None

Example

```
var mycolor = new Color();  
mycolor.setRgb(0x00ff00); // one way to set color to green
```

setRgb

Description

Sets the red, green and blue color values of the color to *red*, *green* and *blue*, respectively.

Parameters

red : Number

green : Number

blue : Number

Returns

None

Example

```
var mycolor = new Color();  
mycolor.setRgb(0, 255, 0); // another way to set color to  
green
```

4.6.2 Dir

Description

The Dir class is used for searching and manipulating directories.

Constructors

Dir	
Description	Creates a new Dir instance which points to the given <i>filePath</i> .
Parameters	filePath : String
Returns	Dir
Example	<pre>var d = new Dir('c:\\myfolder');</pre>

Enums

FilterSpec	
Dirs	List directories only
Files	List files only
Drives	List drives only
NoSymLinks	Do not list symbolic links
All	Lists everything
TypeMask	Dirs Files Drives NoSymLinks
Readable	List files that the application has read access to
Writable	List files that the application has write access to
Executable	List files for which the application has execute access
RWEMask	Readable Writable Executable
Modified	List files that have been modified
System	Lists system files
AccessMask	Readable Writable Executable Modified Hidden System

These filtering options are used in the *entryList* method.

SortSpec	
Name	Sort by name
Time	Sort by modification time
Size	Sort by file size
Unsorted	Do not sort
SortByMask	Name Time Size
DirsFirst	List directories before files
Reversed	Reverse the sort order
IgnoreCase	Sort case-insensitively

These sorting options are used in the *entryList* method.

Static properties

current

Description Returns the current directory for the application.

Access Read

Returns String

Example

```
var myDir = Dir.current; // C:/Program Files/Formpipe  
Software/Lasernet 10
```

home

Description Returns the home directory for the user under which the application runs.

Access Read

Returns String

Example

```
var myDir = Dir.home; // C:/Users/LasernetUser
```

root

Description Returns the root directory.

Access Read

Returns String

Example

```
var myDir = Dir.root; // C:/
```

drives

Description Returns an array of the drives on the system.

Access Read

Returns String[]

Example

```
var myDir = Dir.drives; // C:,D:
```

Static methods

cleanDirPath

Description Removes duplicate directory separators / and removes any relative references .. and . found in *filePath*.

Parameters filePath : String

Returns String

Example

```
var myDir = Dir.cleanDirPath("c:\\program  
files\\..\\lasernet");  
// c:/lasernet
```

convertSeparators

Description Returns a converted version of filePath where all directory separators are converted to backslash.

Parameters filePath : String

Returns String

Example

```
var myDir =  
Dir.convertSeparators(Dir.cleanDirPath("c:\\files\\..\\lasernet"));  
// c:\lasernet
```

Properties

name

Description Returns the name of the directory. This is not the full path, but simply the last part.

Access Read

Returns String

Example

```
var myDir = new Dir('d:\\lasernet');  
var lastFolderName = myDir.name; // lasernet
```

path

Description The full path of the directory.

Access Read, Write

Returns String

Example

```
var myDir = new Dir('d:\\lasernet');  
var lastFolderName = myDir.path; // d:/lasernet
```

absPath

Description Returns the full absolute path – usually starting with a drive specification.

Access Read

Returns String

Example

```
var myDir = new Dir('d:\\file\\..\\lasernet\\');  
var MyPath = myDir.absPath; // d:/lasernet
```

canonicalPath

Description Returns the full path, but without any symbolic links or . or .. relative references.

Access Read

Returns String

Example

```
var myDir = new Dir('d:\\file\\..\\lasernet\\');  
var MyPath = myDir.canonicalPath; // D:/lasernet
```

readable

Description Returns *true* if the directory is readable otherwise *false* is returned.

Access Read

Returns Boolean

Example

```
var myDir = new Dir('d:\\file\\..\\lasernet\\');  
var MyPath = myDir.readable; // false
```

exists

Description Returns *true* if the directory exists otherwise *false* is returned.

Access Read

Returns Boolean

Example

```
var myDir = new Dir('d:\\file\\..\\lasernet\\');
var MyPath = myDir.exists; // false
```

Methods

filePath

Description Returns the filepath of the *filename* file in the directory.

Parameter Filename: String

Returns String

Example

```
var myDir = new Dir('d:\\lasernet\\');
var MyPath = myDir.filePath('test.spl'); //
d:/lasernet/test.spl
```

absFilePath

Description Returns the absolute filepath of the *filename* file in the directory.

Parameter Filename: String

Returns String

Example

```
var myDir = new Dir('d:\\lasernet\\..\\file');
var MyPath = myDir.filePath('test.spl'); // d:/file/test.spl
```

cd

Description Changes directory to the given *dirname* if possible. Otherwise an exception is thrown.

Parameter Dirname : String

Returns Boolean

Example

```
var myDir = new Dir('d:\\lasernet\\..\\file');
myDir.cd('c:\\lasernet');
```

cdUp

Description If possible, changes the directory to the parent directory of the current one. If not possible, an exception is thrown.

Parameter None

Returns Boolean

Example

```
var myDir = new Dir('d:\\lasernet\\..\\file');
myDir.cdUp(); // changes to d:/
```

entryList

Description Returns an array with the list of all files and directories in the directory that matches the given *filterSpec* and *sortSpec*.

Parameter

filter : String

[filterSpec : Number = All]

[sortSpec : Number = Name]

Returns String[]

Example

```
var dir = new Dir('c:\\temp\\');
var filelist = new Array(dir.entryList('*', Dir.All));
```

fileExists

Description Checks if the file *filename* exists in the directory. Returns *true* if it does, otherwise *false* is returned.

Parameter Filename : String

Returns Boolean

Example

```
var dir = new Dir('c:\\temp\\lasernet');
if (dir.fileExists('file.temp'))
{
    // do something now we know the file exists in the
    directory
}
```

mkdir

Description Creates the directory *dirname* in the directory. If for some reason it cannot create the directory (e.g. invalid name) it throws an exception.

Parameter	<code>dirname : String</code>
Returns	<code>Boolean</code>
Example	<pre>var dir = new Dir('c:\\temp\\'); dir.mkdir('test');</pre>

mkdirs

Description	Creates the directory tree <i>dirname</i> if possible. Otherwise an exception is thrown.
Parameter	<code>Dirname : String</code>
Returns	<code>Boolean</code>
Example	<pre>var dir = new Dir('c:\\temp\\'); dir.mkdirs('c:\\temp\\lasernet'); // creates the <i>Lasernet</i> folder in c:\\temp</pre>

rmdir

Description	Deletes the directory <i>dirname</i> if possible (directory must not contain any files). Returns false if the directory cannot be deleted and true on success.
Parameter	<code>Dirname : String</code>
Returns	<code>None</code>
Example	<pre>var dir = new Dir('c:\\temp\\'); dir.rmdir('lasernet'); // removes the <i>Lasernet</i> folder in c:\\temp</pre>

rmdirs

Description	Deletes the specified <i>dirname</i> directory and any parent directories provided they are empty. Returns false if the directory cannot be deleted and true on success.
Parameter	<code>Dirname : String</code>
Returns	<code>None</code>
Example	<pre>var dir = new Dir('c:\\temp\\'); dir.rmdirs('lasernet'); // removes the <i>Lasernet</i> folder and parent folder in c:\\temp\\lasernet</pre>

rmRecursive

Description	Removes the directory, including all its contents. Returns true if successful, otherwise false. If a file or directory cannot be removed, rmRecursive() keeps going and attempts to delete as many files and sub-directories as possible, then returns false. If the directory was already removed, the method returns true (expected result already reached). Note: this function is meant for removing a small application-internal directory (such as a temporary directory), but not user-visible directories. For user-visible operations, it is recommended to report errors more precisely to the user, to offer solutions in case of errors, to show progress during the deletion since it could take several minutes, etc.
Parameter	None
Returns	Boolean
Example	<pre>var dir = new Dir('C:\\Temp\\FolderToRemove'); var result = dir.rmRecursive(); logger.logEvent(Debug, result ? 'Removal Succeeded' : 'Removal Failed');</pre>

4.6.3 FontStyle

Description

The FontStyle class is used for manipulating font style properties.

Properties

bold

Description Gets and sets the bold property of the font used for style.

Access Read, Write

Returns Boolean

Example

```
var globalFont = sheet.fontStyle("Global");
globalFont.bold = true;
```

color

Description Gets and sets the color property of the font style.

Access Read, Write

Returns Color

Example

```
var globalFont = form.fontStyle("Global");
globalFont.color = green;
```

family

Description Gets and sets the font name of the font style.

Access Read, Write

Returns String

Example

```
var globalFont = form.fontStyle("Global");
globalFont.family = "Courier New";
```

italic

Description Gets and sets the italic property of the font style.

Access Read, Write

Returns Boolean

Example

```
var globalFont = form.fontStyle("Global");
globalFont.italic = true;
```

name

Description Returns the style name.

Access Read

Returns String

Example

```
var globalFont = form.fontStyle("Global");
logger.logEvent(0, 'Name of Current Style = ' +
globalFont.name);
```

pointSize

Description Gets and sets the pointSize property of the font style

Access Read, Write

Returns Number

Example

```
var globalFont = form.fontStyle("Global");
globalFont.pointSize = 12;
```

strikeOut

Description Gets and sets the strike out property of the font style.

Access Read, Write

Returns Boolean

Example

```
var globalFont = form.fontStyle("Global");
globalFont.strikeOut = false;
```

underline

Description Gets and sets the underline property of the font style.

Access Read, Write

Returns Boolean

Example

```
var globalFont = form.fontStyle("Global");
globalFont.underline = true;
```

4.6.4 ShapeStyle

Description

The ShapeStyle class is used for manipulating shape style properties.

Properties

fillColor

Description Gets and sets the fill color property of the shape style.

Access Read, Write

Returns Color

Example

```
var globalShape = form.shapeStyle("Global");
globalShape.fillColor = blue;
```

lineColor

Description Gets and sets the line color property of the shape style.

Access Read, Write

Returns Color

Example

```
var globalShape = form.shapeStyle("Global");
globalShape.lineColor = green;
```

lineDashes

Description Gets and sets the line dashes type property of the shape style.

Access Read, Write

Returns Number

Example

```
var globalShape = form.shapeStyle("Global");
globalShape.lineDashes = 2;

var localShape = sheet.shapeStyle("Local");
localShape.lineDashes = localShape.DotLine;
```

Values

- 1; SolidLine
- 2; DashLine
- 3; DotLine
- 4; DashDotLine
- 5; DashDotDotLine

lineWeightPts

Description Gets and sets the line weight property of the shape style.

Access Read, Write

Returns Number

Example

```
var globalShape = form.shapeStyle("Global");
globalShape.lineWeightPts = 10;
```

name	
Description	Returns the style name.
Access	Read
Returns	String
Example	<pre>var globalShape = form.shapeStyle("Global"); logger.logEvent(0,'Name of Current Style = '+ globalShape.name);</pre>

4.6.5 Connection

Description

Holds the properties for a Connection.

Properties

name	
Description	Name of the Connection in the Configuration.
Access	Read
Returns	String
Example	<pre>Logger.logEvent(Debug, connection.name);</pre>

description	
Description	Description of the Connection in the Configuration.
Access	Read
Returns	String
Example	<pre>Logger.logEvent(Debug, connection.description);</pre>

type	
Description	Type of the Connection in the Configuration. Database, SharePoint, SharePoint365, AzureStorage, AzureStorageSAS.
Access	Read
Returns	String

Example	<code>Logger.logEvent(Debug, connection.type);</code>
----------------	---

guid

Description	Unique id of the Connection in the Configuration.
--------------------	---

Access	Read
---------------	------

Returns	String
----------------	--------

Example	<code>Logger.logEvent(Debug, connection.guid);</code>
----------------	---

4.6.6 Module

Description

Holds the properties for a Module.

Properties

name

Description	Name of the Module in the Configuration.
--------------------	--

Access	Read
---------------	------

Returns	String
----------------	--------

Example	<code>Logger.logEvent(Debug, module.name);</code>
----------------	---

description

Description	Description of the Module in the Configuration.
--------------------	---

Access	Read
---------------	------

Returns	String
----------------	--------

Example	<code>Logger.logEvent(Debug, module.description);</code>
----------------	--

type

Description	Type of the Connection in the Configuration. For example; File Input, Printer Service.
--------------------	--

Access	Read
---------------	------

Returns	String
Example	<code>Logger.logEvent(Debug, module.type);</code>

guid

Description	Unique id of the module in the Configuration.
Access	Read
Returns	String
Example	<code>Logger.logEvent(Debug, module.guid);</code>

printer

Description	For Modules of type 'Printer Service', the printer can be read via this property.
Access	Read
Returns	String
Example	<code>Logger.logEvent(Debug, module.printer);</code>

printserver

Description	For Modules of type 'Printer Service', the Print Server can be read via this property.
Access	Read
Returns	String
Example	<code>Logger.logEvent(Debug, module.printserver);</code>

4.6.7 Printer

Description

Holds the properties for a Printer.

Properties

name

Description	Name of the Printer module in the Configuration.
Access	Read
Returns	String
Example	<pre>Logger.logEvent(Debug, printer.name);</pre>

description

Description	Description of the Printer module in the Configuration.
Access	Read
Returns	String
Example	<pre>Logger.logEvent(Debug, printer.description);</pre>

printername

Description	Name of the printer in the Printer module in the Configuration.
Access	Read
Returns	String
Example	<pre>Logger.logEvent(Debug, printer.printername);</pre>

profiles

Description	A list of the printer profiles for a printer.
Access	Read
Returns	Array of PrinterProfiles (4.6.8).
Example	See getPrinterProfiles for example.

4.6.8 PrinterProfile

Description

Holds the properties for a PrinterProfile.

Properties

name	
Description	Name of the PrinterProfile of a Printer in the Configuration.
Access	Read
Returns	String
Example	See getPrinterProfiles for example.

4.6.9 Server**Description**

Holds the properties for a Server.

Properties

name	
Description	Name of the Server in the Configuration.
Access	Read
Returns	String
Example	<pre>Logger.logEvent(Debug, server.name);</pre>

description	
Description	Description of the Server in the Configuration.
Access	Read
Returns	String
Example	<pre>Logger.logEvent(Debug, server.description);</pre>

hostname	
Description	Hostname of the Server in the Configuration.
Access	Read
Returns	String
Example	<pre>Logger.logEvent(Debug, server.hostname);</pre>

port

Description	Port of the Server in the Configuration.
--------------------	--

Access	Read
---------------	------

Returns	String
----------------	--------

Example	<code>Logger.logEvent(Debug, server.port);</code>
----------------	---