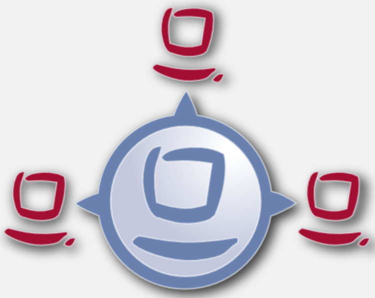


opsi-winst (4.12.0): What is new ?

uib gmbh
Bonifaziusplatz 1b
55118 Mainz
Tel.:+49 6131 275610
www.uib.de
info@uib.de

Stand: 24.07.2017

Contents

1 Copyright **1**

2 Boolean Functions [W/L] **2**

3 Local functions [W/L] **3**

3.1 Concept . . . 3

3.2 Syntax . . . 4

3.3 Examples . . . 4

4 Import von Library Funktionen [W/L] **8**

5 New functions to control the logging [W/L] **9**

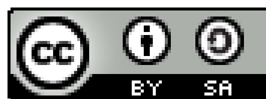
6 Geändertes Loggings [W/L] **10**

Chapter 1

Copyright

The Copyright of this manual is held by uib gmbh in Mainz, Germany.

This manual is published under the creative commons license
Attribution - ShareAlike (by-sa).



A German description can be found here:
http://creativecommons.org/licenses/by-sa/3.0/de/

The legally binding German license can be found here:
http://creativecommons.org/licenses/by-sa/3.0/de/legalcode

The English description can be found here: http://creativecommons.org/licenses/by-sa/3.0/

The English license can be found here: http://creativecommons.org/licenses/by-sa/3.0/legalcode

Most parts of the opsi software are open source.
The parts of opsi that are not open source are still under a co-funded development. Information about these parts can be found here: opsi cofunding projects

All the open source code is published under the AGPLv3.

The legally binding AGPLv3 license can be found here: http://www.gnu.org/licenses/agpl-3.0-standalone.html

Some information around the AGPL: http://www.gnu.org/licenses/agpl-3.0.en.html

For licenses to use opsi in the context of closed software please contact the uib gmbh.

The names *opsi*, *opsi.org*, *open pc server integration* and the opsi logo are registered trade marks of uib gmbh.

Chapter 2

Boolean Functions [W/L]

`boolToString(`<boolean expression>`)` : bool string (true/false) // since 4.12.0.0 [W/L]

`stringToBool(`<string expression: true/false>`)` : boolean // since 4.12.0.0 [W/L]

Chapter 3

Local functions [W/L]

Since version 4.12, the opsi-script has also local functions.

An example:

```
DefFunc myFunc(val $str1$ : string, $str2$ : string) : string
        set $result$ = $str1$ + $str2$
endfunc
```

3.1 Concept

There are a lot possibilities to structure opsi-script code:

- `sub` Sections
- `sub` Sections in external files
- *include* Statements

But all these possibilities are not functional to create reusable external code that can be exchanged between scripts or opsi administrators without problems. The reason is, that this code is not encapsulated and use global variables.
The defined local functions presented here now solves this problem. With this concept it is possible to write functions that can be collected and maintained in external libraries.
In consequence we will start to build up a central opsi-script library which is maintained by uib and the opsi community.

In order to reach this target we have implemented the following concepts:

- Functions with return value:
  The functions have a return value which is of the type `string` or `stringlist`. Executing such function can be performed wherever a string expression or a stringlist is expected.
- Freely definable function call parameters:
  Parameters can be passed to a function. These parameters are defined when the function is actually declared. The call parameters can be of type `string` or `stringlist`.
  The call parameters are available as local variables within the function.
  The call parameters can be passed as *CallByValue* or *callByReference*. *CallByValue* is the default. That means, if no call method is specified explicitly, then *CallByValue* will be applied. In the case that *CallByValue* needs to be explicitly specified, then the keyword `val` should be used. *CallByValue* means, that the value of a variable used during the call is copied to the call variable.
  CallByReference must be specified explicitly using the keyword `ref`. *callByReference* means that a connection is created between the variable used as parameter when calling the function and the local variable that represents the call parameter inside the function. Changing the local variable of the call parameter has a direct effect on the variable used during such call.

- Local Variables:
  A function contains local variables: Implicitly, the call parameters are available as local variables and the variable `$result$` which is from the type of the returned value. Further variables can be defined within the function.
  All these variables are local, which means that they are only visible within this function. A local variable with the same name of a global variable masks the corresponding global variable within the function.
- Nested functions:
  A local function can in turn have one or even more definitions of local functions. These functions are only visible within the function in which they are defined.
- Recursive calls:
  A function can call itself recursively.
- Primary and secondary sections within functions:
  The function body can contain its own sections of it. These are local to this function, that means that these sections are only visible within the function.

3.2 Syntax

Definition

```
DefFunc <func name>([calltype parameter type][,[calltype parameter type]]) : type
<function body>
endfunc
```

Where:

- `DefFunc` is the keyword used to start defining a local function..
- *<func name>* is the freely chosen name of the function.
- *calltype* is the call type of the parameter [`val` | `ref`]. `val`=*Call by Value*, `ref`=*Call by Reference*. Default: `val`
- *parameter* is the free selected name of the call parameter which is available as a local variable within the function under the aforementioned name.
- *type* is the type of data of the parameter concretely the function whether `string` or `stringlist`;
- *<function body>*: is the body of the function which opsi-script syntax must suffice.
- `endfunc` is the keyword used to end defining a local function..

3.3 Examples

Simple function that connects two strings:

```
[actions]
DefVar $mystr$
DefVar $str1$
set $str1$ = 'ha'

DefFunc myFunc(val $str1$ : string, $str2$ : string) : string
        set $result$ = $str1$ + $str2$
endfunc

set $mystr$ = myFunc("he","ho")
set $mystr$ = myFunc("he",timeStampAsFloatStr)
set $mystr$ = myFunc("he",$str1$)
```

Expected results:

- *heho*
- *he42921.809*
- *heha*

Function of the type `stringlist` which will deliver a `string` and a `stringlist`:

```
[actions]
DefVar $mystr$
DefVar $str1$
DefStringlist $list1$
DefStringlist $list2$

set $str1$ = 'ha'

DefFunc myFunc1(val $str1$ : string, $list1$ : stringlist) : stringlist
        set $result$ = createStringlist($str1$ , takeString(2,$list1$))
endfunc

set $list2$ = splitstring("/etc/opsi/huhu","/")
set $list1$ = myFunc1("hi",$list2$)
```

Expected results:

- $list1$ = [hi,opsi]

Function of type `string` to which a boolean `string` will be deliver:

```
[actions]

DefFunc myFunc2($str1$ : string) : string
        set $result$ = booltostring($str1$)
endfunc

if stringtobool(myfunc2('1 > 0'))
        comment "true"
else
        comment "false"
endif
```

Expected results:

- *true*

Function of the type string to which a string is passed with local variable:

```
[actions]
DefVar $mystr$

DefFunc myFunc3($str1$ : string) : string
        DefVar $locstr1$
        set $locstr1$ = '123'
        set $result$ = $locstr1$ + $str1$
endfunc

set $mystr$ = myFunc3("he")
```

Expected results:

- *123he*

Function of the type `string` to which a string is passed with local variable and nested function:

```
[actions]
DefVar $mystr$

DefFunc myFunc4($str1$ : string) : string
        DefVar $locstr1$

        DefFunc myFunc5($str1$ : string) : string
                set $result$ = 'inner' + $str1$
        endfunc

        set $locstr1$ = '123'
        set $result$ = $str1$ + myFunc5($locstr1$)
endfunc

set $mystr$ = myFunc4("outer")
```

Expected results:

- *outerinner123*

Simple function of type `string` which pass a` string` by reference with a local variable:

```
[actions]
DefVar $mystr$
DefVar $str1$
DefVar $str2$

set $str1$ = 'ha'
set $str2$ = 'hi'

DefFunc myFunc6(ref $str1$ : string) : string
        DefVar $locstr1$
        set $locstr1$ = '123'
        set $str1$ = 'setinlocal'
        set $result$ = $locstr1$ + $str1$
endfunc

set $mystr$ = myFunc6($str2$)
set $mystr$ = $str1$ + $str2$
```

Expected results:

- *123setinlocal*
- *hasetinlocal*

Function of type `stringlist` which will pass a variable of type `stringlist` with a *call by reference* also with a local `stringlist` variable:

```
[actions]
DefVar $mystr$
```

```
DefStringlist $list1$
DefStringlist $list2$

et $list2$ = splitstring("/etc/opsi/huhu","/")

DefFunc myFunc7(ref $list1$ : stringlist) : stringlist
        DefStringlist $loclist1$
        set $loclist1$ = splitstring("/a/b/c","/")
        set $list1$ = createStringList('setinlocal')
        set $loclist1$ = addListToList($loclist1$,$list1$)
        set $result$ = $loclist1$
endfunc

set $list1$ = myFunc7($list2$)
comment "$list2$ index 0: " + takestring(0,$list2$)
```

Expected results:

- $list1$ = [,a,b,c,setinlocal]
- *setinlocal*

Function of type `stringlist` which pass a `string` with a local variable and a local secondary section:

```
[actions]
DefStringlist $list1$

DefFunc myFunc8($str1$ : string) : stringlist
        DefStringlist $loclist1$
        set $loclist1$ = getoutstreamfromsection("shellInAnIcon_test")
        set $result$ = $loclist1$

        [shellinanicon_test]
        set -x
        $str1$
endfunc

set $list1$ = myFunc8('pwd')
```

Expected results:

- $list1$ = [+ pwd, /home/uib/gitwork/lazarus/opsi-script]

Chapter 4

Import von Library Funktionen [W/L]

`importLib` <string expr> ; import library // since 4.12.0.0
<string expr> : <file name>[.<file extension>][`::`<function name>]
If no *.<file extension>* is given `.opsiscript` is used as default.
If no *::<function name>* is given, all function from the given file will be imported.

<file name> is:

- A complete path to an existing file. [W/L]
- A existing file in `%ScriptPath%` [W/L]
- A file in `%opsiScriptHelperPath%\lib` [W]
  Is equivalent to: *%ProgramFiles32Dir%\opsi.org\opsiScriptHelper\lib*
- A existing file in `%ScriptPath%/../lib` [W/L]
- A existing file in `%WinstDir%\lib` [W]

The tests for the location of the <file name> are done in the order above. *opsi-script* uses the first file it finds that has a matching name.

Chapter 5

New functions to control the logging [W/L]

Using opsi Configs (`Host-Parameter`) you may now change the logging:

- `opsi-script.global.debug_prog` : boolean
  If false log messages that are only relevant for debugging the opsi-script program it self are not written excepting Warnings and Errors.
  Default: false
  This will keep the log files smaller because you will find only messages that are relevant for understanding what your script is doing.
  The adjustment of all log messages to this new way is in progress and will be take a while since all (about 1700) log calls inside the code are reviewed.
- `opsi-script.global.debug_lib` : boolean
  If false log messages from defined functions that are imported from external library files will be suppressed excepting Warnings and Errors.
  Default : false
  This feature is still buggy right now, but will be fixed until the stable release.
- `opsi-script.global.default_loglevel` : intstr
  Sets (overrides) the default log level that is implemented inside the opsi-script code. This config has no effect on scripts where the log level is explicit set by a `setLogLevel` statement.
  Default : *6*
- `opsi-script.global.force_min_loglevel` : intstr
  Forces a minimal log level.
  This can be used while debugging or development to set temporary and for selected clients a higher log level without changing the script. Default: *0*
- `opsi-script.global.ScriptErrorMessages` : boolean
  This config overwrites the opsi-script internal default value for `ScriptErrorMessages` if opsi-script is running in the context of the opsi web service. If the value is true, syntactical errors trigger a pop up window with some informations on the error. This is in productive environments no good idea. Therefore the default value for this config is *false*.
  Inside a script the statement `ScriptErrorMessages` may be used to set this different from the defaults.
  Default: false
- `opsi-script.global.AutoActivityDisplay` : boolean
  If true shows a marquee (endless) progressbar while external processes (winbatch/dosbatch sections) are running.
  Default: true

Chapter 6

Geändertes Loggings [W/L]

In this version we changed the logging structure and the default log level.

Attention
The new default loglevel is 7 (was 6).

The logging at the different log levels is now:

- Log level 5:
  comments,messages, Execution of sections
- Log level 6:
  Statements, New values for stringvars , results of complete boolean expression
- Log level 7:
  new values for stringlist vars, output from external processes (shellInAnIcon) if the out put is not assinged to a stringlist variable, results of parts of a boolean expression
- Log level 8:
  other stringlist output eg. string lists from stringlist functions and output from external processes (shellInAnIcon) that is assinged to a stringlist variable.

In dieser Version ist das Logging weiter umstrukturiert und damit eine Verhaltensänderung gegeben:

Da (im Normalfall) alle Debugmeldungen welche nur wir als Programmierer brauchen, aus den Logs entfernt wurden ist das ganze etwas schlanker und wir haben das Logging etwas strukturierter über die Loglevel verteilt.

Achtung::Der Default Loglevel ist dabei von 6 auf 7 angehoben worden.

- Auf Loglevel 6 wird geloggt:
  Alle Programmanweisungen, alle Wertzuweisungen zu Stringvariablen, die Ergebnisse kompletter boolscher Ausdrücke (hinter if)
- Auf Loglevel 7 wird geloggt:
  Alle Zuweisungen zu Stringlisten Variablen, die Ausgaben von externen Prozessen, soweit diese nicht einer Stringliste zugewiesen werden, die Ergebnisse der Teilauswertung boolscher Ausdrücke (hinter if)
- Auf Loglevel 8 wird geloggt:
  Stringlisten welche von Funktionen erzeugt werden, die Ausgaben von externen Prozessen, wenn diese einer Stringliste zugewiesen werden