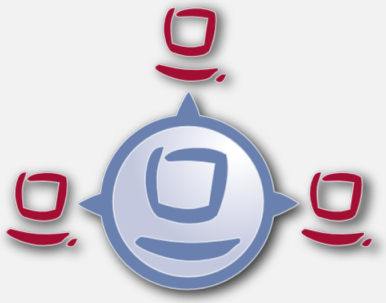

opsi-winst (4.12.0): Was ist neu ?

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

Stand: 24.07.2017

Inhaltsverzeichnis

1 Copyright **1**

2 Boolsche Funktionen [W/L] **2**

3 Selbst definierte lokale Funktionen [W/L] **3**

3.1 Konzept 3

3.2 Syntax 4

3.3 Beispiele 4

4 Import von Funktionen aus Libraries [W/L] **8**

5 Neue Funktionen zur Steuerung des Loggings [W/L] **9**

6 Geändertes Loggings [W/L] **11**

Kapitel 1

Copyright

Das Copyright an diesem Handbuch liegt bei der uib gmbh in Mainz.

Dieses Handuch ist veröffentlicht unter der creative commons Lizenz
Namensnennung - Weitergabe unter gleichen Bedingungen (by-sa).

Eine Beschreibung der Lizenz finden Sie hier:
http://creativecommons.org/licenses/by-sa/3.0/de/

Der rechtsverbindliche Text der Lizenz ist hier:
http://creativecommons.org/licenses/by-sa/3.0/de/legalcode

Die Software von opsi ist in weiten Teilen Open Source.
Nicht Open Source sind die Teile des Quellcodes, welche neue Erweiterungen enthalten die noch unter Kofinanzierung stehen, also noch nicht bezahlt sind.
siehe auch: http://uib.de/de/opsi-erweiterungen/erweiterungen/

Der restliche Quellcode ist veröffentlicht unter der AGPLv3:

Der rechtsverbindliche Text der AGPLv3 Lizenz ist hier:
http://www.gnu.org/licenses/agpl-3.0-standalone.html

Deutsche Infos zur AGPL: http://www.gnu.org/licenses/agpl-3.0.de.html

Für Lizenzen zur Nutzung von opsi im Zusammenhang mit Closed Source Software kontaktieren Sie bitte die uib gmbh.

Die Namen *opsi*, *opsi.org*, *open pc server integration* und das opsi-logo sind eingetragene Marken der uib gmbh.

Kapitel 2

Boolsche Funktionen [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]

Kapitel 3

Selbst definierte lokale Funktionen [W/L]

Seit Version 4.12 kennt opsi-script auch lokale Funktionen.

Ein Beispiel:

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

Konzept

Auch bisher gab es schon Möglichkeiten opsi-script Code zu strukturieren:

- `sub` Sektionen
- `sub` Sektionen in gesonderten Dateien
- *include* Anweisungen

Diese Möglichkeiten ließen es aber nicht zu, ausgelagerten Code beliebig und sicher zwischen verschiedenen Scripten oder Anwendern auszutauschen. Dies liegt daran, daß all diese Konzepte nicht gekapselt sind und mit globalen Variablen arbeiten.
Mit den hier vorgestellten definierbaren lokalen Funktionen soll das Schreiben von Funktionen möglich werden, welche in externen Bibliotheken (Libraries) gepflegt werden können.
Entsprechend soll es dann auch eine zentrale durch die opsi-community und uib gepflegte Library geben.

Um dieses Ziel zu erreichen werden in dieser Erweiterung folgende Konzepte umgesetzt:

- Funktionen mit Rückgabewert:
  Die Funktionen haben einen Rückgabewert welche vom Typ `string` oder `stringlist` ist. Der Aufruf eine solchen Funktion kann überall da Erfolgen, wo ein Stringausdruck bzw.eine Stringliste erwartet wird.
- Frei definierbare Funktions Aufrufparameter:
  Einer Funktion können Parameter übergeben werden. Diese Parameter werden bei der Deklaration der Funktion definiert. Die Aufrufparameter können vom Typ `string` oder `stringlist` sein.
  Die Aufrufparameter können als *CallByValue* oder per *callByReference* übergeben werden. *CallByValue* ist der Default. Das bedeutet: wird keine Aufrufmethode explizit angegeben, so wird *CallByValue* verwendet. Soll *CallByValue* explizit angegen werden, so erfolgt dies über das Schlüsselwort `val`. *CallByValue* bedeutet, das beim Aufruf der Inhalt einer beim Aufruf verwendeten Variable auf die Aufrufvariable kopiert wird.
  CallByReference muß über das Schlüsselwort `ref` explizit angegeben werden. *callByReference* bedeutet, dass beim Aufruf eine Verbindung zwischen der aufrufenden Varibalen und des lokalen Aufrufparameters erstellt wird. Eine

Änderung der lokalen Variable des Aufrufparameters, wirkt sich direkt auf die beim Aufruf verwendete Variable aus.
Die Übergabeparameter stehen innerhalb der Funktion als lokale Variablen zur Verfügung.

- Lokale Variablen:
  Eine Funktion enthält lokale Variablen: Implizit gibt es die Aufrufparameter als lokale Variablen und die Variable `$result$` welche vom Typ des Rückgabewertes ist. Darüberhinaus können weitere Variablen innerhalb der Funktion definiert werden.
  All dies Variablen sind lokal, d.h. sie sind nur innerhalb dieser Funktion sichtbar. Eine lokale Variable mit dem selben Namen einer globalen Variable verdeckt innerhalb der Funktion die entsprechende globale Variable.
- Geschachtelte Funktionen:
  Eine lokale Funktion kann wiederum eine oder mehrere Definitionen von lokalen Funktionen enthalten. Diese Funktionen sind nur innerhalb der Funktion sichtbar in der sie definiert sind.
- Rekursive Aufrufe:
  Eine Funktion kann sich selbst rekursiv aufrufen.
- Primäre und sekundäre Sektionen innerhalb von Funktionen:
  Der Funktionskörper kann eigene Sektionen enthalten. Diese sind lokal zu dieser Funktion, also nur innerhalb der Funktion sichtbar.

Syntax

Definition

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

Dabei ist:

- `DefFunc` das Schlüsselwort zur Beginn der Definition einer lokalen Funktion.
- *<func name>* der frei gewählte Name der Funktion.
- *calltype* ist der Aufruftyp [`val` | `ref`]. Wird kein Aufruftyp angegeben, + so gilt `val` als gesetzt.
- *parameter* ist der freigewählte Name des Aufrufparameters, welcher unter diesem Namen innerhalb der Funktion als lokale Variable zur Verfügung steht.
- *type* ist der Datentyp des Parameters bzw. der Funktion und entweder `string` oder `stringlist`;
- *<function body>*: ist der Körper der Funktion, welcher dem opsi-script syntax genügen muß.
- `endfunc` zeigt als Schlüsselwort das Ende einer Funktionsdefinition an.

Beispiele

Einfache Funktion welche zwei Strings miteinander verbindet:

```
[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",$str1$)
```

Erwartete Ergebnisse:

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

Funktion vom Type `stringlist` welcher ein `string` und eine `stringlist` übergeben werden:

```
[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",$list2$)
```

Erwartete Ergebnisse:

- $list1$ = [hi,opsi]

Funktion vom Type `string` welcher ein boolscher `string` übergeben wird:

```
[actions]

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

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

Erwartete Ergebnisse:

- *true*

Funktion vom Type `string` welcher ein `string` übergeben wird mit lokaler Variable:

```
[actions]
DefVar $mystr$

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

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

Erwartete Ergenisse:

- *123he*

Funktion vom Type `string` welcher ein `string` übergeben wird mit lokaler Variable und geschachtelter Funktion:

```
[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")
```

Erwartete Ergebnisse:

- *outerinner123*

Einfache Funktion vom Type `string` welcher ein `string` by reference übergeben wird mit lokaler 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 $mystr$ = $str1$ + $str2$
```

Erwartete Ergebnisse:

- *123setinlocal*
- *hasetinlocal*

Funktion vom Type `stringlist` welcher eine Variable vom Type `stringlist` mit *call by reference* übergeben wird mit lokalen `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,$list2$)
```

Erwartete Ergenisse:

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

Funktion vom Type `stringlist` welcher ein `string` übergeben wird mit lokaler Variable und lokaler sekundärer Sektion:

```
[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')
```

Erwartete Ergebnisse:

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

Kapitel 4

Import von Funktionen aus Libraries [W/L]

`importLib` <string expr> ; import library // since 4.12.0
<string expr> : <file name>[.<file extension>][::<function name>]
Wenn keine *.<file extension>* (Dateierweiterung) übergeben wird, so wird `.opsiscript` als Default verwendet.
Wenn kein *::<function name>* übergeben wird, so werden alle Funktionen der angegebenen Datei importiert.

<file name> ist:

- Ein kompletter Pfad zu einer Datei. [W/L]
- Eine Datei in `%ScriptPath%` [W/L]
- Eine Datei in `%opsiScriptHelperPath%\lib` [W]
  Entspricht: *%ProgramFiles32Dir%\opsi.org\opsiScriptHelper\lib*
- Eine Datei in `%ScriptPath%/../lib` //since 4.11.5.2 [W/L]
- Eine Datei in `%WinstDir%\lib` [W]

Die Prüfung erfolgt in dieser Reihenfolge. Die erste Datei die gefunden wird, wird genommen.

Kapitel 5

Neue Funktionen zur Steuerung des Loggings [W/L]

Über opsi Configs (`Host-Parameter`) kann das Logging beeinflusst werden:

- `opsi-script.global.debug_prog` : boolean
  Wenn false werden Logmeldungen welche zum Debuggen des opsi-script selber dienen nicht ausgegeben, soweit es sich nicht um Warnungen oder Fehler handelt.
  Default: false
  Damit werden die Logdateien entlastet und nur noch Meldungen die Script relevant sind, stehen in den Logdateien. Die Umstellung der entsprechenden Logmeldungen im Quellcode des opsi-script, ist noch nicht abgschlossen und wird bei ca. 1700 Log aufrufen auch noch etwas dauern.

- `opsi-script.global.debug_lib` : boolean
  Wenn false, so werden Logmeldungen aus lokalen Funktionen welche aus Libraries importiert wurden, nur ausgegeben soweit es sich um Warnungen oder Fehler handelt.
  Default : false
  Die Umsetzung ist noch buggy, wird aber bis zum stable Release repariert sein.

- `opsi-script.global.default_loglevel` : intstr
  Setzt (überschreibt) den Standard defaultloglevel von opsi-script.
  Dieser Config hat keinen Einfluss auf Scripte bei denen der Loglevel per `setLogLevel` explizit gesetzt worden ist.
  Default : *6*

- `opsi-script.global.force_min_loglevel` : intstr
  Erzwingt einen minimalen Loglevel.
  Dies dient dazu bei der Entwicklung und/oder Fehlersuche gezielt und temporär für einzelne Clients den Loglevel zu erhöhen ohne hierzu Anpassungen am Script vornehmen zu müssen.
  Default: *0*

- `opsi-script.global.ScriptErrorMessages` : boolean
  Wenn false werden Syntax-Fehlermeldungen nicht interaktiv ausgegeben sondern nur in die Logdatei geschrieben. Im Produktivbetrieb ist es sinnvoll das dieser Parameter false ist. Daher ist Default für diesen Config=false. Der Default von opsi-script für diesen Parameter ist (aus historischen Gründen) true. Im Servicekontext überschreibt der Config den Default von opsi-script. Ausserhalb des Servicekontext gilt der Default von opsi-script. Diese Default Werte können innerhalb eines scriptes mit der Anweisung `ScriptErrorMessages` überschrieben werden.
  Default: false

- `opsi-script.global.AutoActivityDisplay` : boolean
  Wenn true wird während des Laufs von externen Prozessen (winbatch,dosbatch,execwith Sektionen) ein (marquee) Fortschrittsbalken (der Endlos durch läuft) angezeigt.

Default: true

Kapitel 6

Geändertes Loggings [W/L]

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