Befehlsübersicht für Knotenscripte

Hier werden die internen Befehle zur Script-Konfiguration von Knoten erläutert. Diese können entweder direkt eingegeben werden oder sind Bestandteil von Scripten, welche der NodeScript Wizard verwendet.

Allgemeine Regeln

Scripte unterteilen sich in drei Ebenen: Basisbefehle, Templatebefehle und Eingabeaufforderungen.

Basisbefehle sind Anweisungen an den Knoten, bestimmte Einstellungen durchzuführen (z.B. bei einem Lichtport die Helligkeit einzustellen). Basisbefehle bestehen aus einem Anweisungswort und zugehörigen Parametern. Befehle und Parameter müssen gültige Anweisungen ergeben, sonst wird die Anweisung nicht ausgeführt.

Templatebefehle sind vorgelagerte Anweisungen zur Textersetzung. Es wird die Templatesyntax velocity verwendet. Templatebefehle werden nur im Wizard ab Version 1.9 unterstützt.

Menübefehle sind nochmal vorgelagert und werden verwendet, um ein Menü anzuzeigen, in welchem Parameter vom Benutzer abgefragt werden - wie z.B. welche Ports verwendet werden sollen. Diese Parameter werden internen Variablen zugewiesen und können dann mit Hilfe der Templates ausgewertet werden.

Ein Script kann nur aus Basisbefehlen bestehen. Template- und Menübefehle sind optional und ermöglichen eine einfache Wiederverwendbarkeit und Benutzerführung.

Basisbefehle

Kommentar

Kommentare werden durch '##'-Zeichen eingeleitet. In der Zeile wird alles hinter dem Kommentarzeichen ignoriert.

Wartezeit (Delay)

Verzögerungen bei der Ausführung des Scriptes werden durch einen *wait*-Befehl erreicht. Die Zeitangabe erfolgt im ms.

```
wait 5000   ## wait 5 sec
```

Label zuweisen

Label sind Namen von Ports, Macros, Accessories und Aspekten. Die Zuweisung der Namen erfolgt mit

dem *set*-Befehl.

```
## change the label
set macro 0 name="Weiche1Gerade"
set macro 1 name="Weiche1Abzweig"
set servo 0 name="weiche1"
set switch 10 name="herz1A"
set accessory 0 name="Weiche1"

## ab Wizard-Build 3032
set flag 3 name="Flag 3"

## ab Wizard-Build 3540
set aspect 0 accessory=${selectedAccessory} name="${macroName}_gerade_AC"

## ab Wizard-Build 4153
set feedback 10 name="Station Test"
```

CV Werte zuweisen

Um CV-Werte anzupassen, wird auch der *set*-Befehl verwendet.

```
## set the CVs
set CV 437 2    ## GPIO 0
set CV 440 2    ## GPIO 1
```

Knoten-Reset auslösen

Um einen Knoten-Reset auszulösen, werden folgende Befehle verwendet:

```
## restart the node
reset
wait 5000   ## wait 5 sec
reselect    ## select the node again (internal wait for 5000ms to get data
from node)
```

Ab Build 3489:
Um einen Knoten-Reset nur dann auszulösen wenn der Knoten den *Pending Restart* Fehler gesetzt hat, muss folgender Befehl verwendet werden:

```
restart pendingonly=[true|false] delay=3000
```

Wenn `pendingonly=true` ist, wird geprüft ob der *Pending Restart* Fehler gesetzt ist. Falls Ja wird ein Restart des Knoten ausgeführt, dann (im obigen Fall) `3000ms` gewartet und anschliessend der Knoten neu selektiert. Falls der *Pending Restart* Fehler nicht gesetzt ist, werden diese Anweisungen übersprungen.

Port konfigurieren

```
config port ptype=light ValueOff=6 ValueOn=230 DimmOff=20 DimmOn=6 number=17
```

Die Parameter können auch weggelassen werden.
Die nicht angegebenen Parameter werden dann auf 0 gesetzt.

ptype	Parameter
light	valueoff / valueon / dimmoff / dimmon / dimmon88 / dimmoff88 / mapping
servo	lowerlimit / upperlimit / turntime
switch	iotype / ticks / loadtype
switchpair	ticks / loadtype
backlight	dimmoff / dimmon / dimmon88 / dimmoff88 / mapping

LoadType

Mögliche Werte:
0: allgemein (nur Schalten, alle Fehlerbilder ignorieren)
1: ohmsche Last
2: Magnetspule ohne Endabschaltung
3: Magnetspule mit Endabschaltung
Wird für keine Last eine Lageüberwachung unterstützt, entfällt der Parameter.

Port-Typ ändern

Um den Port auf einen anderen Typ zu mappen muss folgendes Kommando verwendet werden:

```
assert port number=<Port-Num> ptype=<Port-Typ>
```

Der ptype ist der Ziel-Porttyp.

ptype
analog
backlight
input
light
motor
servo
sound
switch
switchpair

Makro konfigurieren

Allgemeines Vorgehen:

Zuerst muß ein Makro ausgewählt werden, alle folgenden Befehl wirken dann auf dieses Makro.

```
select macro 7    ## Makro 7 wird ausgewählt
```

Nun kann für dieses Makro der Name und das allgemeine Verhalten festgelegt sowie die einzelnen Schritte hinzugefügt werden.
Mit **config macro** wird die Wiederholrate und die Verzögerung des Makros eingestellt.
Mit **config macrotime** wird die Startzeit des Makros eingestellt.
Mit **set macro 8 name=„das_ist_8“** wird ein Name definiert.
Mit **add step (parameter)** werden einzelnen Schritte hinzugefügt.

 Das Makro wird auf den Knoten übertragen und permanent gespeichert!

Beispiel Makroverhalten:

```
## Macro 1 wird ausgewählt
select macro 1
## Hier werden 3 Wiederholungen und die Verzögerung auf 1 gesetzt
config macro repeat=3 slowdown=1
## Macro 1 läuft nur Mittwochs (Tag 2), jede Stunde zur 15’ten Min.
config macrotime day=2 hour=everyfull minute=15
```

config	Parameter	Wert
macro	repeat	Variable o. Zahl (0-250)
	slowdown	Variable o. Zahl (1-250)
macrotime	day	Variable o. Zahl (0-7, everyday(7))
	hour	Variable o. Zahl (0-25, everyfull(24), everyfullatday(25))
	minute	Variable o. Zahl (0-62, everyminute(60), everyhalfhour(61), everyquater(62))

Beispiel Makroschritte hinzufügen:

```
### hier werden die benötigten Variablen gesetzt (Zeilen mit Velocity-Syntax
beginnen mit einem #)
#set($macro0 = 0)
#set($macro1 = 1)
#set($Macro_Weiche1Gerade = "Weiche1Gerade")
#set($Macro_Weiche1Abzweig = "Weiche1Abzweig")

#set($switchport_herz1A = 0)
#set($swicthport_herz1B = 1)
#set($Port_herz1A = "Herz1A")
#set($Port_herz1B = "Herz1B")

#set($servoport_weiche1 = 0)
#set($Servo_weiche1 = "ServoWeiche1")
```

```
### Hier beginnt der Abschnitt mit der BiDiB-Macro-Sysntax
### den Ports und Macros werden Namen gegeben
set switch ${switchport_herz1A} Name="${Port_herz1A}"
set switch ${switchport_herz1B} Name="${Port_herz1B}"

set servo ${servoport_weiche1} Name="${Servo_weiche1}"

set macro ${macro0} name="${Macro_Weiche1Gerade}"
set macro ${macro1} name="${Macro_Weiche1Abzweig}"

select macro ${macro0}
add step ptype=macro action=stop name="${Macro_Weiche1Abzweig}"
###add step ptype=macro action=stop number=${macro1}
###add step ptype=switch action=off name="${herz1A}"
add step ptype=switch action=off number=10
add step ptype=switch action=off name="${herz1B}"
add step delay=200 ptype=servo action=start name="${Servo_weiche1}"
target=30
add step ptype=switch action=on name="${herz1B}"

### alternative mit macro number
### add step ptype=macro action=stop number=1
### alternative mit port number
### add step ptype=switch action=off number=10

## Servobewegung abwarten mit Portnummer
add step ptype=moveServoQuery number=10
```

Folgende Actions werden unterstützt:

ptype	**action**
macro	start / stop / end
servo	start
switch	on / off
switchpair	on / off
light	on / off / up / down / neon/ blinka / blinkb / flasha / flashb / doubleflash
input	query0 / query1
accessoryOkay	no_feedback / query0 / query1
backlight	start
critical	begin / end
flag	clear / query0 / query1 / set
delay	-
randomDelay	-
moveServoQuery	

Für die Typen `delay` und `randomDelay` wird keine `action` angegeben. Eine Verzögerung wird wie folgt definiert:

```
add step ptype=delay delay=10
add step ptype=randomDelay delay=100
```

Accessory erzeugen

Um ein Accessory zu erstellen werden die nachfolgenden Befehle verwendet:

```
## prepare the accessory
select accessory 0
add aspect 0 macroname=%weiche1Gerade%
add aspect 1 macronumber=1
```

Das Startup-Verhalten eines Accessorys kann mit folgendem Befehl definiert werden:

```
config accessory startup aspect=1
```

Als Startup-Verhalten ist die Zuweisung eines Aspekts (`aspect=1`), das Wiederherstellen des letzten Begriffes (`config accessory startup restore`) oder keine Aktion (`config accessory startup none`) möglich.

 Das Accessory wird auf den Knoten übertragen und permanent gespeichert!

Templatebefehle

Templatebefehle erlauben das Anlegen von Variablen, die Manipulation von Variablen und das Einfügen dieser so erzeugen Werte in die Basisbefehle. Damit können Vorlagen erstellt werden, in denen nur noch die benutzten Ports und Accessorys festgelegt werden. Als Syntax für die Templatebefehle wird velocity verwendet. (Befehlsreferenz) Velocity wurde ursprünglich für html-Seiten verwendet, kann aber ebenso für allgemeine Textersetzungen verwendet werden.

template-Befehle beginnen immer mit einem #-Zeichen.

Wertzuweisungen

Werte werden mit dem *#set*-Befehl zugewiesen:

```
#set($led_gn = 6)   ## led_gn ist an Port 6 angeschlossen
```

Der *#set*-Befehl kann auch benutzt werden, um Werte zu verändern:

```
#set($led_gn = $led_gn + 3)  ## led_gn ist jetzt an Port 9 angeschlossen
```

Strings werden mit Quotation (Anführungszeichen) zugewiesen.

Werte einsetzen

Innerhalb von Basisbefehlen kann man Werte einsetzen, in dem man ein *${* davor und ein *}* dahinter

setzt.

```
#set($my_accessory = 2)   ## wir benutzen Accessory 2
#set($my_macro = 6)   ## wir benutzen Makro 6
set macro ${my_macro} name="Signal_${my_accessory}_hp0"  ## Basisbefehl:
Makro benennen
#set($my_macro1 = ${my_macro} + 1)
set macro ${my_macro1} name="Signal_${my_accessory}_hp1"  ## Basisbefehl:
Makro benennen
```

Das erzeugt folgende Ausgabe:

```
set macro 6 name="Signal_2_hp0"   ## Basisbefehl: Makro benennen
set macro 7 name="Signal_2_hp1"   ## Basisbefehl: Makro benennen
```

Mit Hilfe der Zuweisungen an my_macro und my_accessory und der Texteinsetzungen in den Folgezeilen wird erreicht, dass diese Folgezeilen immer gleich bleiben können; es wurde Dateninhalt und Datenbenutzung getrennt. Durch Einfügen der Accessory-Nummer in den Makro-Namen erreicht man eine Eindeutigkeit, diese ist hilfreich, wenn man dieses Makro später benutzen will.

Schleifen, Wiederholungen

Mit dem Befehlen ***#foreach*** und ***#end*** können Abschnitte mehrfach ausgegeben werden. ***#break*** kann so eine Schleife unterbrechen (Link zur Velocity-Dokumentation).
#foreach arbeitet alle Elemente eines Array ab.
Hier im Beispiel wird ein Array mit den Zahlen 1 bis 5 definiert. **$my_number** enthält eine Zahl nach der anderen. Bis **$my_number** größer 4 ist und ***#foreach*** mit ***#break*** abgebrochen wird.

```
#set ($my_array = [1,2,3,4,5])

#foreach ($my_number in $my_array)  ## wir wiederholen
  #if ( $my_number > 4 )
   #break
  #end
Das wird wiederholt
#end
```

Ausgabe:

```
Das wird wiederholt
Das wird wiederholt
Das wird wiederholt
Das wird wiederholt
```

Weitere Beispiele:

```
#foreach ( $portnum in [0..30] )
add step ptype=light action=off number=$portnum
#end
```

Menübefehle

Menübefehle werden zu Beginn eines Scriptes ausgewertet und ermöglichen die Abfrage von Benutzerparametern. Die Benennung des Menüs erfolgt mit dem Schlüsselwort **##instruction**, Eingabeaufforderungen werden mit **##input** definiert.

Menübenennung

Eine Kurzbeschreibung wird durch das Keywort **##application** definiert:

```
##application(text:de="Konfiguration eines Lichtsignal der DB",
text:en="Configuration of a light signal of DB")
```

Die genauere Beschreibung wird durch das Keywort **##instruction** definiert. Durch die Verwendung von
 kann ein Zeilenumbruch forciert werden.

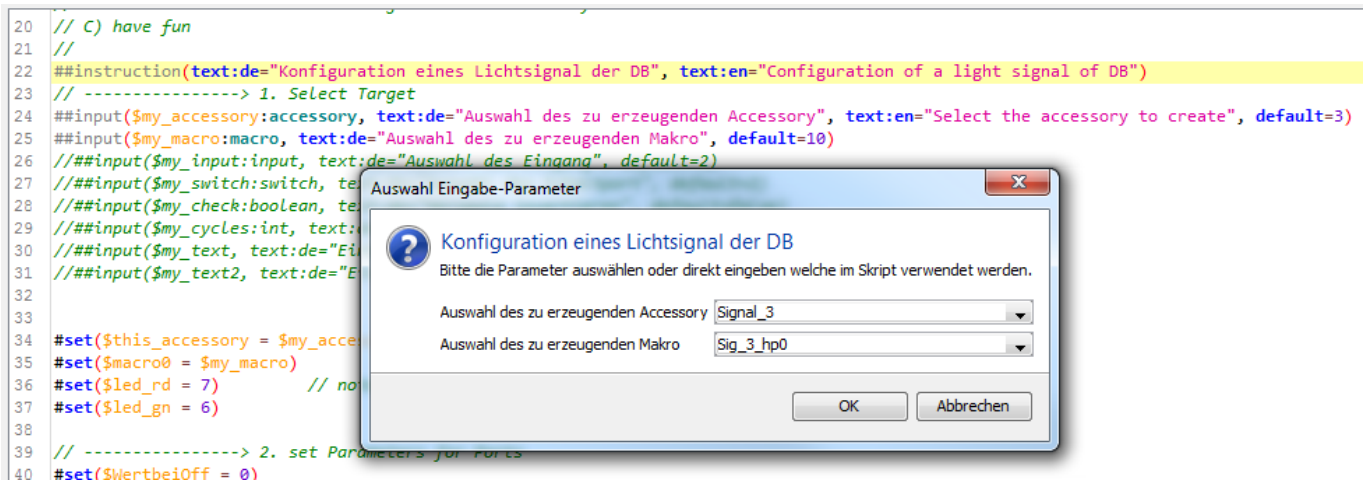
```
##instruction(text:de="Konfiguration eines Lichtsignal der DB: <br>Erster
Lightport = rt, ge <br>Erstes Macro = Hp0, Hp1", text:en="Configuration of a
block signal (DB): first lightport = rt, ge")
```

Ab Build 2662 kann in der ##instruction ein **Link** (z.B. link=„http://www.bidib.org“) angegeben werden:

```
##instruction(text:de="Konfiguration eines Lichtsignal der DB: <br>Erster
Lightport = rt, ge <br>Erstes Macro = Hp0, Hp1", text:en="Configuration of a
block signal (DB): first lightport = rt, ge", link="http://www.bidib.org")
```

Um längere Texte hinterlegen zu können, kann die ##instruction auch pro Sprache auf jeweils einer eigenen Zeile angegeben werden.

```
##instruction(text:de="Konfiguration eines Lichtsignal der DB: <br>Erster
Lightport = rt, ge <br>Erstes Macro = Hp0, Hp1")
##instruction(text:en="Configuration of a block signal (DB): first lightport
= rt, ge")
```



Eingabeaufforderung

Eine Eingabeaufforderung beginnt mit `##input` und enthält eingeschlossen in `(…)` eine Liste mit dem Parameter, welcher abgefragt wird, den darzustellenden Text und Defaultwert. Der Parameter wird als Variable interpretiert und muss deshalb auch mit dem $-Prefix angegeben werden.

```
##input($my_accessory:accessory, text:de="Hier Accessory eingeben",
default=3)
```

Direkt hinter `##input` wird der Variablenname angegeben, welcher mit dieser Inputanweisung befüllt wird. In der Voreinstellung ist das ein String, durch Angabe eines Modifiers (`:` gefolgt von einen Typ) können auch andere Variablentypen erzeugt werden.

modifier	Typ der Variable	Eingabeunterstützung
`:int`	integer	Textfeld zur Eingabe einer Ganzzahl
`:string`	string	Textfeld (string ist Voreinstellung)
`:boolean`	boolean	Checkbox, true / false
`:accessory`	integer	Auswahlbox der möglichen Accessory des Knotens
`:macro`	integer	Auswahlbox der möglichen Makros des Knotens
`:input`	integer	Auswahlbox der möglichen Eingangsports des Knotens
`:light`	integer	Auswahlbox der möglichen Lichtports des Knotens
`:servo`	integer	Auswahlbox der möglichen Servoausgänge des Knotens
`:switch`	integer	Auswahlbox der möglichen Schaltausgänge des Knotens
`:flag`	integer	Auswahlbox der möglichen Flags des Knotens

VID/PID prüfen

Um die VID und PID zu prüfen, kann die `##require`-Anweisung benutzt werden:

```
##require(vid="13", pid="129")
```

Mit der obigen Anweisung wird geprüft ob der Knoten VID 13 und PID 129 hat. Ist dies nicht der Fall wird eine Fehlermeldung angezeigt. Die Werte für VID und PID müssen als Dezimalwerte angegeben werden.

Um ein Skript auf mehrere PID zu beschränken können die PIDs in einer komma-separierten Liste

angegeben werden.

```
##require(vid="13", pid="129,130,145")
```

Die Angabe der VID(PID kann auch als Hex-Zahl erfolgen mit 0x-Prefix:

```
## ab Wizard-Build 4153
##require(vid="0x0D", pid="0xA0,0x9F,0x53")
```

Autor

Mit der Anweisung ##author wird der Autor in der Auswahlseite angezeigt, wenn das Skript ausgewählt wird:

```
##author(your name here)
```

Vordefinierte Variablen

Scripte - gerade wenn sie universell für verschiedene Knoten formuliert sein sollen - brauchen auch Informationen über den Zielknoten. Deshalb gibt es vordefinierte Variablen, die man im Script auswerten kann.

- dimm_range
  Diese Variable wird beim Aufruf des Scriptes abhängig von den Dimmfähigkeiten des Knotens mit übergeben. Mögliche Werte: 8 oder 16. Damit kann die Vorgabe der Dimmgeschwindigkeit automatisch an die Fähigkeiten des Knotens angepasst werden.
  Beispiel:

  ```
  #if (${dimm_range} == 8)
  #set($DimmzeitOff = 10)
  #set($DimmzeitOn = 10)
  #else
  #set($DimmzeitOff = 1010)
  #set($DimmzeitOn = 1010)
  #end
  ```

- power_user
  Diese Variable wird beim Aufruf des Scriptes abhängig von der Anwenderwahl ('ich bin Power-User') mit übergeben.
- uniqueid
  In dieser Variable wird die UniqueId beim Aufruf des Scriptes abhängig von dem gewählten Knoten übergeben. Damit kann man ein Script z.B. auf bestimmte Knoten beschränken oder Abhängigkeiten vom Knoten erfassen.
- uniqueidhex
  In dieser Variable wird die UniqueId als Hex-Wert vom Typ String übergeben, z.B. 0x05000D7F002200.
- vid

In dieser Variable wird die VendorId beim Aufruf des Scriptes abhängig von dem gewählten Knoten übergeben.
- pid
  In dieser Variable wird die ProductId beim Aufruf des Scriptes abhängig von dem gewählten Knoten übergeben.
- node_firmware_version
  in dieser Variable wird die aktuelle Firmware-Version des Knoten übergeben (ab Build 2764).
- node_accessory_count
  Diese Variable wird mit der Anzahl der möglichen Accessories des Knotens belegt (Feature: FEATURE_ACCESSORY_COUNT).
- node_accessory_macro_mapped
  Diese Variable wird mit der Anzahl der möglichen Aspekte pro Accessory belegt, welche der Knoten unterstützt (Feature: FEATURE_ACCESSORY_MACROMAPPED).
- node_macro_count
  Diese Variable wird mit der maximalen Anzahl der Makros für diesen Knoten belegt (Feature: FEATURE_CTRL_MAC_COUNT).
- node_macro_size
  Diese Variable wird mit der maximalen Anzahl der Makroschritte pro Makro für diesen Knoten belegt (Feature: FEATURE_CTRL_MAC_SIZE).
- node_backlight_count
  Anzahl Backlight Ports (Feature: FEATURE_CTRL_BACKLIGHT_COUNT)
- node_input_count
  Anzahl Input Ports (Feature: FEATURE_CTRL_INPUT_COUNT)
- node_light_count
  Anzahl Light Ports (Feature: FEATURE_CTRL_LIGHT_COUNT)
- node_servo_count
  Anzahl Servo Ports (Feature: FEATURE_CTRL_SERVO_COUNT)
- node_switch_count
  Anzahl Switch Ports (Feature: FEATURE_CTRL_SWITCH_COUNT)
- node_switchpair_count
  Anzahl SwitchPair Ports
- node_enabled_switch_count
  Anzahl der tatsächlich aktiven Switch Ports auf dem Knoten. Bei umschaltbaren Ports kann dieser Wert von node_switch_count abweichen.
- node_enabled_switchpair_count
  Anzahl der tatsächlich aktiven SwitchPair Ports auf dem Knoten. Bei umschaltbaren Ports kann dieser Wert von node_switchpair_count abweichen.
- user_lang
  atkuelle Sprache, z.B. en oder de (ab Wizard 1.12.4)

Script-Bausteine und Variablen

Mit der folgenden Anweisung wird eine Checkbox erstellt, welche dem Anwender die Möglichkeit gibt zu wählen, ob Namen (Labels) für Accessorys, Makros und Ports ersetzt werden sollen oder nicht.

```
##input($prevent_replace_labels:boolean, text:de="Keine Namen für
Accessorys, Makros, Ports ersetzen", text:en="Prevent replace labels for
accessory, macro and ports", default=false)
```

Im nachfolgenden Script-Snippet wird die Variable `prevent_replace_labels` ausgewertet um das Umbenennen von Ports, Makros, Accessorys anhand des vom Anwender gewählten Wertes auszuführen oder zu unterbinden.

```
## Setzt label, wenn ${prevent_replace_labels} NICHT true
#if (!${prevent_replace_labels})
set light ${led_vr_ge1}
name="${AccessoryVorsignal}${myAcc_Vorsignal}_${led_vr_ge1}_ge1"
set macro ${macro_vr0} name="${AccessoryVorsignal}_${myAcc_Vorsignal}_Vr0"
set accessory ${myAcc_Vorsignal}
name="${AccessoryVorsignal}_${myAcc_Vorsignal}"
#end
```

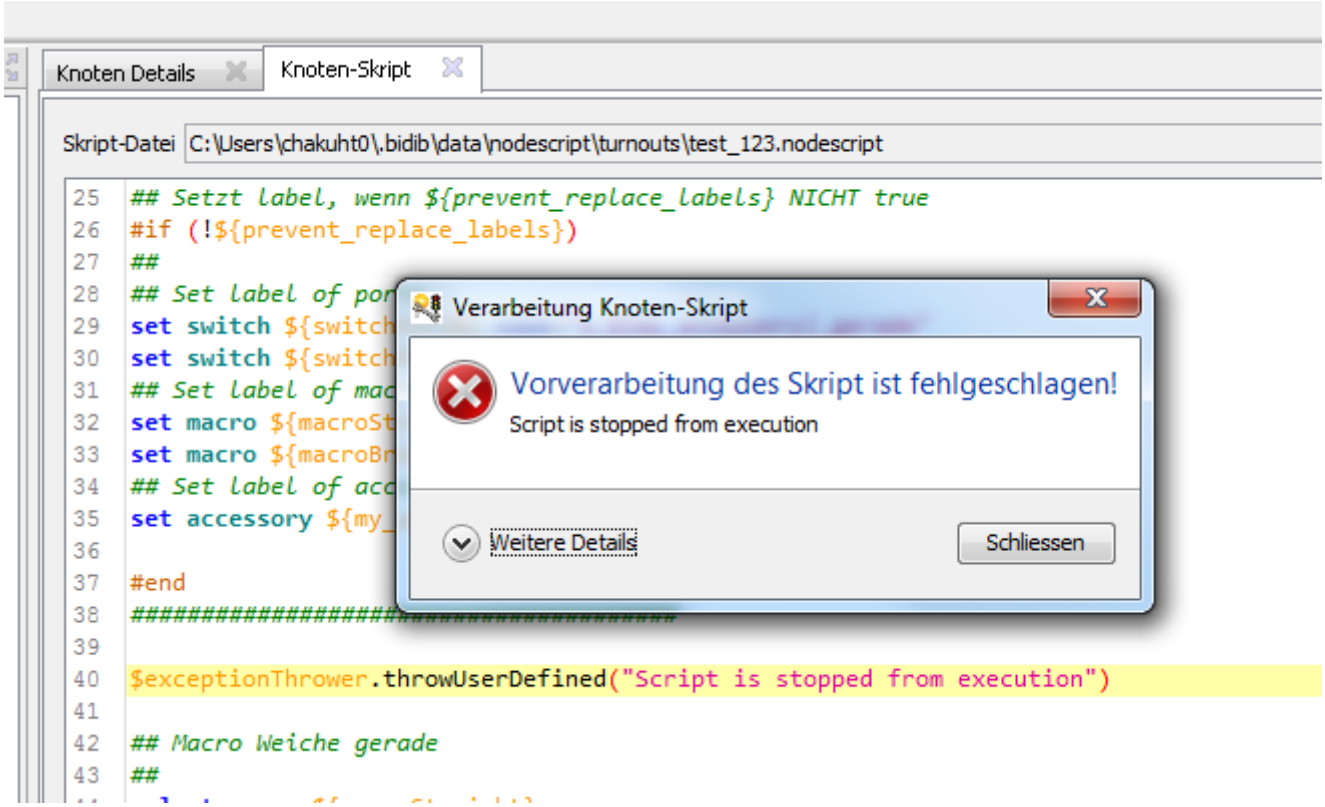
 Der NodeScript-Editor des Wizard bietet für die obige `##input`-Anweisung eine Autovervollständigung (ShorthandCompletion) an. Durch Eingabe von `pre` (für `prevent`) und `Ctrl+Enter` wird die komplette Zeile erzeugt.

Abbrechen eines Script

Die Ausführung eines Script kann durch die folgende Anweisung abgebrochen werden:

```
$exceptionThrower.throwUserDefined("Script is stopped from execution")
```

Der Text `Script is stopped from execution` ist frei wählbar und wird in einem Dialog angezeigt:



Wenn diese Anweisung in einen #if Block gepackt, dann kann die Ausführung selektiv abgebrochen werden.

Sprachabhängige Fehlermeldung

Eine sprachabhängige Fehlermeldung kann durch Auswertung der `$user_lang` erfolgen:

```
#if ($node_macro_size < $led_count)
    #if ($user_lang == "de")
    $exceptionThrower.throwUserDefined("Maximale Anzahl der Strassenlampen
pro Makro auf diesem Knoten: $node_macro_size")
    #else
    $exceptionThrower.throwUserDefined("Maximum supported streetlamps per
macro on this node: $node_macro_size")
    #end
#end
```

Tricks mit Velocity

String in Integer parsen

Um einen String in einen Integer zu parsen, muss man folgenden Trick anwenden:

```
#set($bar = "32")
```

```
#set($Integer = 0)
#set($foo2 = $Integer.parseInt($bar))
The value is: $foo2
```

Dabei wird zuerst eine Variable `$Integer` mit `0` initialisiert. Anschliessend kann man Java-Integer-Operationen darauf anwenden.

Der nachfolgende Code macht das gleiche, hat aber nur 1 Variable `$Integer` statt `$foo2` wie im obigen Snippet:

```
#set($bar = "32")
#set($Integer = 0)
#set($Integer = $Integer.parseInt($bar))
The value is: $Integer
```

Versionsnummer des Knotens zerlegen

Die Version des Knotens wird in der Variable ${node_firmware_version} geliefert. Wenn man diesen Wert auf die einzelnen Komponenten zerlegen will, dann kann man dies mit dem folgenden Skript erreichen. Das Array enthält dann die einzelnen Versionsnummern als String. Innerhalb der If-Bedingung werden daraus Integer-Werte berechnet.

```
#set($Integer = 0)
#set($version_major = 0)
#set($version_sub   = 0)
#set($version_run   = 0)
#set($version_string="${node_firmware_version}")
#set($version_array = $version_string.split("\."))
#if($version_array.size() == 3)
 #set($version_major = $Integer.parseInt($version_array[0]))
 #set($version_sub   = $Integer.parseInt($version_array[1]))
 #set($version_run   = $Integer.parseInt($version_array[2]))
#end
```

Textreste

Tipps zu velocity
Der Kontext ist eine Map<String, Object>, also eine Registry, in der man unter einem Schlüssel (Key) einen Wert (Value) reinhängen kann. Der Kontext wird immer übergeben (kann auch leer sein). Die #set-Zeilen im Script legen dann die Variablen in den Kontext rein und diese werden dann später beim Ersetzen verwendet. Die aufgerufene Funktion kann die Werte mit dem Key wieder rausholen. Erläuterung hierzu:
https://github.com/castleproject/MonoRail/blob/master/MR2/NVelocity/src/NVelocity.Tests/Test/ContextTest.cs

Text wieder anzeigen: velocityEngine.Evaluate()

Alles ab hier ist noch Sammelsurium !

Vor der Ausführung des Scriptes werden zuerst alle Befehle syntaktisch geprüft. Es erfolgt ggf. die Ausgabe aller gefundenen Fehler. Nur bei syntaktisch korrektem Code wird die Ausführung des Scripts begonnen. Tritt während der Ausführung ein Fehler, auf bricht das Script sofort ab. Alle Fehler werden unter Angabe der Zeilennummer des Befehls und dem Fehlergrund ausgegeben.

Bei erfolgreicher Ausführung wird auch eine Erfolgsmeldung ausgegeben.

Script Header

Aktuell noch kein Support in Wizard und Monitor

Jedes Script hat mit einem Header in der ersten Zeile zu beginnen. In diesem Header wird festgelegt für welchen Knotentyp und für welche Script Engine Version das Script erstellt wurde.

```
<Head PID=125 VID=13 VER=1.0>
```

Mit der Kombination aus PID / VID Produktidentifikation wird geprüft, ob das Script zu dem gewählten Knoten des Herstellers passt. Wird ein Script nicht auf dem passenden Knoten ausgeführt, bricht die Verarbeitung ab. Mit der Version der Scripting Engine kann die Script Engine prüfen, ob sie das Script korrekt ausführen kann.

Zusätzlich ist der Macro-Level zu prüfen. Der Macro-Level bezeichnet den Funktionsumfang der Makro-Engine und wird vom Knoten per Feature bekannt gegeben.

Platzhalter

Bei der Zuweisung eines Schlüsselwortes an einen Platzhalter wird im BiDiB-Wizard der Name des Schlüsselwortes durch den Platzhalternamen ausgetauscht.

Eine Platzhalteranweisung kann an einer beliebigen Stelle innerhalb eines Scripts stehen. Vor der Verwendung eines Platzhalters muss dieser definiert werden. ~~Der Wert eines Platzhalters kann innerhalb des Scripts mehrfach geändert werden.~~

```
## this is a test script for Herz8
define herz1A "Herzrelais 1AC"
define herz1B "Herzrelais 1BC"
define weiche1Gerade "Weiche 1 gerade"
define weiche1Abzweig "Weiche 1 abzweig"
define weiche1 "Weiche 1"
```

Platzhalter dürfen folgende Zeichen beinhalten: Zahlen, Buchstaben, '_', '+', '-'

From:
https://forum.opendcc.de/wiki/ - **BiDiB Wiki**

Permanent link:
https://forum.opendcc.de/wiki/doku.php?id=tools:scripting-node-syntax&rev=1599837267

Last update: **2020/09/11 17:14**