PTB – AG 7.54

Anleitung zu »relayServer«

Rolf Niepraschk

2018-01-19

Einleitung

Bei »relayServer« handelt es sich um ein auf »nodejs« basierendes Programm, welches als sogenannter »Dämon« auf einem Linux-Rechner gestartet wird, d. h. es läuft solange auch der Linux-Rechner läuft. »relayServer« bietet Netzwerkzugänge in Form mehrerer http-Server auf unterschiedlichen Ports. Mit dieser Hilfe empfängt »relayServer« Daten von außerhalb befindlichen Programmen (z. B. Web-Applikationen) und sendet daraufhin verschiedenartige Daten zurück. Der Sinn ist, auf diese Weise Funktionen anzubieten, die ohne »relayServer« nicht oder nur in komplizierter Weise zur Verfügung stehen würden.

1 relay-Server – Port: 55555

Die Komponente »relay-Server« bietet den Zugang zu einer Vielzahl von externen auf dem Linux-Rechner installierten Programmen und Netzwerkprotokollen. Web-Applikationen sind beispielsweise damit in der Lage die sich aus der sogenannten »Same-Origin-Policy« ergebenen Einschränkungen zu überwinden oder Daten per Netzwerkzugriff zu erhalten, die normalerweise über diesen Weg nicht zugänglich sind.

Die Anforderung an den »relay-Server« geschieht immer über eine Datenstruktur im JSON-Format. Diese enthält mindestens einen mit »Action« benannten Wert, der die auszuführende Aktion auswählt.

1.1 Allgemeingültige Parameter

Repeat – Positive Ganzzahl; gibt an, wie oft die betreffende Aktion wiederholt werden soll (optional, Standard: »1«). Dieser Parameter wird nicht von allen Aktionen unterstützt.

Wait – Positive Ganzzahl; gibt die Zeit in ms an, die nach Ende des vorherigen Ablaufs gewartet werden soll (optional, Standard: »0«). Diese Angabe ist nur wirksam, wenn »Repeat« größer als »1« ist. Manche Aktionen verwenden trotz »Wait:0« eine kleine Mindestwartezeit.

PostProcessing – String oder String-Array; JavaScript-Code, mit dem das Ergebnis einer vorher bereits beendeten Aktion nachbearbeitet werden kann. Die Umgebung, in der diese Nachbearbeitung stattfindet, ist limitiert, d. h. dass aus Sicherheitsgründen nur auf ausgewählte Komponenten zugegriffen werden kann. Das sind im Einzelnen die folgenden Variablen:

`_x` – das noch unbearbeitete Ergebnis der vorherigen Aktion.

`_` – Objekt, welches den Zugriff auf die exportierten Funktionen der Datei »`relay-add.js`« gestattet.

`_$` – Datenstruktur, die zum Anfang als JSON-String an »relayServer« geschickt wurde (enthält z. B. `Value` und `Action`),.

An die Rückgabe-JSON-Struktur werden alle die im JavaScript-Code definierten Variablen weitergereicht, die *nicht* mit `_` (Unterstrich) beginnen. Um den üblicherweise verwendeten Rückgabewert `Result` tatsächlich zu ändern, ist z. B. eine solche Zuweisung nötig: »`Result= 3.14 * _x + 17;`«

PreProcessing – String oder String-Array; JavaScript-Code, mit dem das aus dem Eingangs-JSON-String hervorgegangene Objekt gewandelt werden kann. Dies geschieht sehr ähnlich zu dem Ablauf bei »`PostProcessing`«. Im Gegensatz dazu ist die Variable `_$` nicht definiert und die Variable `_x` enthält das erwähnte Eingangsobjekt. Eine Modifizierung des Wertes von `Value` könnte z. B. folgendermaßen erfolgen: »`Value=_.encodeVACOM(_x.Value);`«. (»`encodeVACOM`« ist eine per »`relay-add.js`« bereitgestellte Funktion.)

1.2 Externe Aktionen

Aufruf eines auf dem Server installierten Programms. Aus Gründen der Sicherheit geschehen diese Aufrufe – wie auch der gesamte relayServer-Prozess – mit Rechten des Users »nobody«, d. h. mit sehr wenigen Rechten.

`EXECUTE`

Ausführen eines Programmes ggf. mit Paramtern, darunter auch zu interpretierender Programmcode. Die Parameter:

Cmd – Name des auszuführenden Programmes mit komplettem Pfad (zwingend).

Args0 – Parameter (String oder Array von Strings). Wird für den Programmaufruf direkt dem Programmnamen hinzugefügt (optional).

Body – Text (meist Programmcode) in Form eines einfachen Strings oder eines Arrays von Strings. Im zweiten Falle wird aus dem Array intern ein einzelner String erzeugt, wobei nach jedem Array-Element ein Zeilenumbruch (»`\n`«) eingefügt wird. Der gesamte Text wird beim Ausführen dieser Aktion in eine Datei in einem temporären Verzeichnis geschrieben. Der Name dieser Datei wird als Parameter dem Programmaufruf hinzugefügt (optional).

Args – Parameter (String oder Array von Strings). Folgen für den Programmaufruf direkt dem Namen der temporären Datei. Bei per »Body« angegebenem Programmcode können sie als Parameter für diesen genutzt werden. Ohne »Body« verhält sich »Args« identisch zu »Args0« (optional).

KeepFiles – »true« oder »1« verhindert, dass nach erfolgtem Programmaufruf das temporär angelegte Verzeichnis und dessen Inhalt gelöscht wird (optional, Standard: »false«, nur für Testzwecke).

Beispiel:

```
cat <<EOF | curl -T - -X PUT http://localhost:55555
{"Action":"EXECUTE","Cmd":"/usr/bin/date","Args":"+%Y-%m-%d"}
EOF
```

Die Antwort lautet:

```
{"t_start":1492593299399,"t_stop":1492593299409,
 "exitCode":0,"Result":"2017-04-19\n"}
```

Beispiel:

```
cat <<EOF | curl -T - -X PUT http://localhost:55555
{"Action":"EXECUTE","Cmd":"/usr/bin/Rscript",
"Body":["a <- 1:10","b <- which(a > 2 & a < 8)","b"],
"Args0":"--vanilla","Args":["foo","bar"]}
EOF
```

Das Senden dieses JSON-Codes führt auf dem Server zum Anlegen einer Datei `prg.dat` in einem temporären Verzeichnis mit folgendem Inhalt:

```
a <- 1:10
b <- which(a > 2 & a < 8)
b
```

Der komplette Programmaufruf im temporären Verzeichnis lautet:

```
/usr/bin/Rscript --vanilla prg.dat foo bar
```

Die Parameter »foo« und »bar« deuten Programmparameter des R-Programms an, die hier allerdings ignoriert werden. Zurückgesendet wird der vom aufgerufenen Programm auf die Standardausgabe ausgegebene Text (»Result«), in diesem Falle:

```
{"t_start":1492589451825,"t_stop":1492589452055,
 "exitCode":0,"Result":"[1] 3 4 5 6 7\n"}
```

1.3 Interne Aktionen

Es handelt sich um eine vom Server direkt ausführbare Aktion, bei der die Angabe des Namens eines externen Programms nicht notwendig ist bzw. es wird auf ein solches gänzlich verzichtet. Beginnt »`Action`« mit dem Zeichen »_«, so kennzeichnet sie einen internen Prozess zu administrativen Zwecken.

Action – kennzeichnet die zu erledigende Aufgabe.

`VXI11`

Kommunikation über das VXI-11-Protokoll mit Messgeräten. Die weiteren Parameter:

Host – IP-Adresse oder Rechnername des Messgerätes (zwingend)

Device – die Geräteadresse (zwingend)

Value – der String zum Auslösen eines bestimmten Gerätebefehls (zwingend)

Encoding – Format der Rückgabe (optional, Standard: utf8). »base64« sollte gewählt werden, wenn das Gerät Binärdaten schickt.

VxiTimeout – Alias für »readTimeout«

readTimeout – Zeit in ms, die bei einer Leseoperation auf eine Rückmeldung vom Gerät gewartet wird (optional, Standard: 2000). Der Wert »0« hat eine besondere Bedeutung: In diesem Falle wird beim Empfang der von dem angesprochenen Gerät gesendeten Daten ein »timeout error« ignoriert. Dieses Verhalten ist hilfreich bei Geräten, die tatsächlich Daten senden, aber fälschlich »timeout« signalisieren und bei Geräten, die überhaupt nichts zurücksenden.

ioTimeout – Zeit in ms für interne Operationen (optional, Standard: 10000)

lockTimeout – Zeit in ms für interne Operationen (optional, Standard: 10000)

lockDevice – Boolscher Wert, der bestimmt, ob der Zugriff auf das Gerät exklusiv ist oder nicht (optional, Standard: true).

termChar – Zahl oder String (optional, Standard: 0 – no term char)

```
echo '{"Action":"VXI11","Host":"e75481",
  "Device":"gpib0,5","Value":"*IDN?"}' | \
  curl -T - -X PUT http://localhost:55555
```

```
{"t_start":1401797586696,"t_stop":1401797586737,
  "Result":"HEWLETT-PACKARD,34970A,0,13-2-2\n"}
```

Die vom Gerät gesendete Antwort ist dem Kennwort »Result« zugeordnet. »t_start«/»t_stop« kennzeichnen die Dauer des Aufrufes in ms. Falls die absolute Zeit von Interesse ist, kann sie mit

```
date -d @$[1401797586696 / 1000]
```

in Sekundengenauigkeit dargestellt werden:

```
Di 3. Jun 14:13:06 CEST 2014.
```

`MODBUS`

Der »relayServer« verhält sich in diesem Modus wie ein Modbus-TCP-Server.

Host – IP-Adresse oder Rechnername des Modbus-Clients (zwingend)

Port – Port-Nummer (optional, Standard: 502)

FunctionCode – Modbus-Funktion. Derzeit (2015-10-22) werden nur die folgenden unterstützt:

- ReadHoldingRegisters
- WriteSingleRegister
- ReadInputRegisters
- ReadCoils
- ReadDiscreteInputs
- WriteSingleCoil

Ausführlich getestet wurden nur die beiden ersten.

Address – Anfangsadresse des zu erfragenden Speichers im Modbus-Client (zwingend)

Quantity – Anzahl der auszulesenden aufeinanderfolgenden Speicherzellen (optional, Standard: 1, Maximum: 125). Ist Quantity > 1, wird ein Array als Antwort (»Result«) zurückgeliefert. Im anderen Fall ist es ein Einzelwert.

Value – Der Wert, der in eine Speicherzelle geschrieben werden soll (bei Schreiboperationen zwingend). Bei numerischen Operationen (z. B. »WriteSingleRegister«) wird eine Ganzzahl verlangt. Bei diskreten Operationen (z. B. »WriteSingleCoil«) wird ein boolscher Wert (»true« oder »false«) erwartet. Bei Angabe einer Zahl wird diese als »true« interpretiert, es sei denn, es handelt sich um den Wert »0«. Ist »Value« ein Array, so wird angenommen, dass es sich um ein Bit-Array mit 0- oder 1-Werten handelt (Index 15 = Wertigkeit 16384).

OutMode – Gibt an, in welchem Format das Ergebnis geliefert werden soll (optional, Standard: »Uint16«). Ist nur bei den Modbus-Funktion, die numerische Werte liefern, wirksam (»ReadHoldingRegisters« und »ReadInputRegisters«). Folgende Angaben sind möglich:

Uint16 – Array of 16Bit-Integers oder einzelner 16Bit-Wert

16Bits – Array of Bit-Arrays (16 Bits)
Mit »Bit-Array« ist ein numerisches Feld mit 0- oder 1-Werten (Ja/Nein-Werten) gemeint.

8Bits – Array of Bit-Arrays (8 Bits)
Es werden nur die niederwertigen 8 Bit der jeweiligen 16-Bit-Zahl als Bit-Array dargestellt.

16Bits* – Bit-Array (Quantity * 16 Bits)
Es werden sämtliche Bits aller 16-Bit-Zahlen als ein Bit-Array dargestellt.

8Bits* – Bit-Array (Quantity * 8 Bits)
Es werden nur die niederwertigen 8 Bit aller 16-Bit-Zahlen als ein Bit-Array dargestellt.

Skip – Gibt an, wieviele der erfragten 16-Bit-Werte übersprungen werden, bevor ein weiterer als Ausgabewert verwendet wird (optional, Standard: 0; derzeit ist nur der Wert »1« wirksam). Der Parameter ist dann nützlich, wenn sich in regulär fortlaufender Folge wichtige und unwichtige Werte abwechseln. Jene können auf diese Art unterdrückt werden.

Die folgenden Beispiele zeigen die Abfrage der digitalen Eingänge von sechs Modulen »8DE« der FESTO-Ventilinsel (Modul 1: 24V an Eingang 1, Modul 2: 24V an Eingang 2, . . . , Modul 6: 24V an Eingang 6). Jeder zweite 16-Bit-Wert enthält sogenannte »Diagnosedaten«, die hier nicht von Interesse sind und daher im zweiten und dritten Beispiel ausgeblendet werden.

```
cat <<EOF | curl -T - -X PUT http://localhost:55555
{"Action":"MODBUS","Host":"10.0.0.31","Address":45395,"Quantity":11,
 "FunctionCode":"ReadHoldingRegisters", "OutMode":"Uint16"}
EOF
# ======== Ergebnis: ========
{
  "t_start": 1445500306261,
  "t_stop": 1445500306261,
  "Result": [1,0,2,0,4,0,8,0,16,0,32]
}
```

```
cat <<EOF | curl -T - -X PUT http://localhost:55555
{"Action":"MODBUS","Host":"10.0.0.31","Address":45395,"Quantity":11,
 "FunctionCode":"ReadHoldingRegisters", "OutMode":"8Bits","Skip":1}
EOF
# ======== Ergebnis: ========
{
  "t_start": 1445500592190,
  "t_stop": 1445500592197,
  "Result": [
    [1,0,0,0,0,0,0,0],
    [0,1,0,0,0,0,0,0],
    [0,0,1,0,0,0,0,0],
    [0,0,0,1,0,0,0,0],
    [0,0,0,0,1,0,0,0],
    [0,0,0,0,0,1,0,0]
  ]
}
```

```
cat <<EOF | curl -T - -X PUT http://localhost:55555
{"Action":"MODBUS","Host":"10.0.0.31","Address":45395,"Quantity":11,
 "FunctionCode":"ReadHoldingRegisters", "OutMode":"8Bits*","Skip":1}
```

```
EOF
# ======== Ergebnis: ========
{
  "t_start": 1445500905697,
  "t_stop": 1445500905724,
  "Result":
  [1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,
   0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0]
}
```

Es folgen Beispiele, die Schreiboperationen zeigen. Mit 40007 wird das Register einer FESTO-Ventilinsel adressiert, welches bei uns zum Schalten einzelner Ventile benutzt wird. Soll z. B. das 5te Ventil eingeschaltet werden, muss das 5te Bit den Wert »1« erhalten. Im folgenden Beispiel werden alle 16 Bits auf »1« gesetzt ($2^{16} - 1 = 65535$):

```
cat <<EOF | curl -T - -X PUT http://localhost:55555
{"Action":"MODBUS","Host":"172.30.56.46","Address":40007,
"FunctionCode":"WriteSingleRegister", "Value":65535}
EOF
```

Alternativ kann auch ein Array, welches 16 »0«- oder »1«-Werte enthält, gesendet werden:

```
cat <<EOF | curl -T - -X PUT http://localhost:55555
{"Action":"MODBUS","Host":"10.0.0.31","Address":40007,
"FunctionCode":"WriteSingleRegister",
"Value":[0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0]}
EOF
```

TEX

Bei dieser Aktion handelt es sich eigentlich auch um einen externen Programmauf. Aufgrund der Komplexität ist sie dennoch als »interne Aktionen« eingeordnet.

Es wird aus dem übergebenen TEX-Code eine Datei erzeugt, die mit einem der unterstützten TEX-Compiler in PDF-Code (PDF-Stream) konvertiert wird.

Body – TEX-Code in Form eines Strings oder eines Arrays von Strings (zwingend). Im zweiten Falle wird aus dem Array intern ein einzelner String erzeugt, wobei nach jedem Inhalt eines Array-Elements ein Zeilenumbruch (»\n«) eingefügt wird.

Command – Der Name des TEX-Compilers (optional, Standard: »pdflatex«). Es werden nur TEX-Compiler unterstützt, die auf direktem Wege PDF-Dateien erzeugen, d. h. »pdflatex«, »lualatex«, »xelatex«, »pdftex«, »luatex«, »xetex«.

KeepFiles – »true« oder »1« verhindert, dass nach erfolgtem Programmaufruf das temporär angelegte Verzeichnis und dessen Inhalt gelöschte wird (optional, Standard: »false«, nur für Testzwecke).

`TCP`

Host – IP-Adresse oder Rechnername (zwingend)

Port – Port-Nummer (zwingend)

Value – Zu sendende Daten (zwingend)

Timeout – Zeit in ms, die bei einer Leseoperation auf eine Rückmeldung vom Gerät gewartet wird (optional, Standard: 10000).

Die Antwort der Gegenstelle (`"Result"`) ist ein String.

`HTTP`

URL – (zwingend)

Body – Zu sendende Daten (optional)

Method – HTTP-Methode (»GET«, »POST« oder »PUT«; optional). Ist sie nicht angegeben, so wird im Falle von vorhandenem Body-Eintrag »POST« gewählt, sonst »GET«.

Json – »true« oder »false« (optional, Standard: false).

Headers – »{}« (optional)

Encoding – (optional, Standard: utf8).

Timeout – (optional, Standard: 20...120 sec).

Beispiel:

```
cat <<EOF | curl -T - -X PUT http://localhost:55555
{"Action":"HTTP","Url":"http://a73434:55555",
"Body":{"Action":"_os_release"},"Json":true}
EOF
```

Es wird dem lokalen »relayServer« der Auftrag erteilt, per HTTP-Protokoll einen anderen entfernten »relayServer« anzusprechen und mit diesem die Action »`_os_release`« auszuführen. Die zurückgelieferten JSON-Daten

```
{"t_start":1538464194174,"t_stop":1538464194184,
"Result":{"Result":{"NAME":"openSUSE Leap",
"VERSION":"42.3","ID":"opensuse","ID_LIKE":"suse",
"VERSION_ID":"42.3","PRETTY_NAME":"openSUSE Leap 42.3",
"ANSI_COLOR":"0;32","CPE_NAME":"cpe:/o:opensuse:leap:42.3",
"BUG_REPORT_URL":"https://bugs.opensuse.org",
"HOME_URL":"https://www.opensuse.org/"}}}
```

enthalten in diesem Falle Angaben zum Betriebssystem des entfernten Rechners.

`UDP`

Host – IP-Adresse oder Rechnername (zwingend)

Port – Port-Nummer (zwingend)

Value – Zu sendende Daten (zwingend)

Die Antwort (`"Result"`) ist, abgesehen von drastischen Fehlerfällen, immer der String `"OK"`.

`EMAIL`

Host – SMTP-Server (zwingend, z. B. `smtp-hub`)

Port – Port-Nummer (optional, Standard: 25)

From – Muss wie Email-Adresse aussehen (zwingend)

ReplyTo – Gültige Antwort-Email-Adresse (optional)

To – Gültige Email-Adresse (zwingend)

Text – Zu sendender einfacher Text (optional, String oder String-Array, Standard: `""`)

Html – Zu sendender HTML-Code (optional und alternativ, String oder String-Array, Standard: `""`)

Subject – Betreff (optional, aber dringend empfehlenswert)

Cc, Bcc – Verteiler (optional)

Secure – `true|false` (optional, Standard: `false`)

Zu diesen und weiteren Parametern, die teilweise noch ungetestet sind, siehe: `http://www.nodemailer.com/`. Beispiel:

```
cat <<EOF | curl -T - -X PUT http://localhost:55555
{"Action":"EMAIL", "Host":"smtp-hub", "From": "VacLab <invalid@ptb.de↩
  >",
 "To":"Rolf.Niepraschk@ptb.de", "ReplyTo":"Thomas.Bock@ptb.de",
 "Subject":"Von Diwan", "Text":"Hallo Rolf!"}
EOF
```

`XLSX-OUT`

Eine JSON-Datenstruktur, die ein oder mehrere Arbeitsblätter mit Datenfeldern enthält, wird in das XML-basierte EXCEL-Datenformat gewandelt und als Binär-Stream zurückgesendet.

Value – JSON-Objekt mit enthaltenen Arbeitsblättern. Das JSON-Objekt muss folgende Struktur haben:

```
cat <<EOF | curl -T - -X PUT http://localhost:55555 > hugo.xlsx
{"Action":"XLSX-OUT",
  "Value":[
    {
      "name": "worksheet 1",
      "data": [
```

```
        ["A1", "B1", "C1"],
        ["A2", "B2", "C2"]
      ]
    },
    {
      "name": "worksheet 2",
      "data": [
        ["X1", "Y1", "Z1"],
        ["X2", "Y2", "Z2"]
      ]
    }
  ]
}
EOF
```

Als Datentypen werden Strings und numerische Werte unterstützt.

`XLSX-IN`

Der BASE64-kodierte Inhalt einer XML-basierten EXCEL-Datei wird in eine JSON-Datenstruktur (siehe »XLSX-OUT«) gewandelt und selbige zurückgesendet.

Value – EXCEL-Daten als BASE64-kodierter String

```
cat <<EOF | curl -T - -X PUT http://localhost:55555
{"Action":"XLSX-IN","Value":"$(base64 -w 0 foo.xlsx)"}
EOF
# ======== Ergebnis: ========
{
  "t_start": 1475227458498,
  "t_stop": 1475227458548,
  "Result": [{
    "name": "Tabelle1",
    "data": [
      ["Spalte X", "Spalte Y", "Spalte Z"],
      ["X1", "Y1", "Z1"],
      ["X2", "Y2", "Z2"],
      ["X3", "Y3", "Z3"],
      [117, 118, 119]
    ]
  }]
}
```

`RANDOM`

Es gibt hier keine weiteren Parameter. Die Antwort (`"Result"`) ist eine Fließkommazahl (Stringform) zwischen 0 und 1.

`TIME`

Es gibt hier keine weiteren Parameter. Die Antwort (`"Result"`) ist eine Zeitangabe in der Form `"HH:MM:SS"`.

`_killRepeats`

Es gibt hier keine weiteren Parameter. Ziel dieser Aktion ist es, alle anderen vorher laufenden Aktionen, die sich jeweils in einer durch `"Repeat"` festgelegten Schleife befinden, vorfristig zu beenden.

`_version`

Es gibt hier keine weiteren Parameter. Die Antwort (`"Result"`) ist die Angabe zur Version des laufenden Node.js-Relais-Servers sowie der nodejs-Version.

`_nodejsVersion`

Es gibt hier keine weiteren Parameter. Die Antwort (`"Result"`) ist die Angabe zur Version des Node.js-Programms.

`_environment`

Es gibt hier keine weiteren Parameter. Die Antwort (`"Result"`) ist die Angabe der dem Node.js-Relais-Server bekannten Umgebungsvariablen des Betriebssystems in Form eines JSON-Objektes.

`_exit`

Es gibt hier keine weiteren Parameter. Die Antwort (`"Result"`) ist `"OK"` und der Rückkehrcode 0. Der Server wird direkt nach Senden der Antwort beendet. Sofern dafür Sorge getragen wird, dass der »relayServer« sofort wieder neu gestartet wird (z. B. per »systemd«), kann »`_exit`« dazu verwendet werden, eine erneuerte Programmversion wirksam werden zu lassen.

2 Gitlab-Hook-Server – Port: 3420

Anmerkung:
Die im Folgenden beschriebene Serverfunktionalität ist nicht mehr innerhalb des zuvor beschriebenen »relayServer« enthalten (2014-10-08). Zu einem späteren Zeitpunkt wird es ein eigenständiges Paket mit dem Gitlab-Hook-Server geben.

Die Web-Applikation »GitLab« gestattet ähnlich zu dem Konkurrenzprodukt »GitHub« einen bequemen Zugriff auf das Versionskontrollsystem »GIT«. Zu jedem in »GitLab« registrierten Repositorium kann über »Settings«/»Web hooks« eine http-Adresse angegeben werden. Nach Änderung des betreffende Repositoriums (push-Aktion) wird an diese Adresse eine Datenstruktur gesendet. Sie enthält etliche Angaben zu dem Repositorium, wie Name des Repositoriums, URL für GIT-Aktionen, Nutzername, IDs der letzten Commits u. v. a. »relayServer« bietet die Funktionalität eines Gitlab-Hook-Servers und ist somit in der Lage, diese Informationen von »GitLab« zu empfangen und auszuwerten. Die Adresse, die in »GitLab« angegeben werden muss, lautet im Falle des auf »a73434« laufenden »relayServer« folgendermaßen:

```
http://a73434.berlin.ptb.de:3420
```

Zur Definition dessen, was nach Eintreffen der Informationen von »GitLab« zu tun ist, muss eine Datei `gitlabhook.conf` angelegt werden. Als Beispiel sei hier der Inhalt angeführt, der dazu führt, dass sich automatisch mit Änderung des Repositoriums »vaclabpage« html-Seiten, die vom Webserver ausgeliefert werden, erneuern:

```
{
  "tasks": {
    "vaclabpage": [
      "exec 1>/dev/null",
      "exec 2>/dev/null",
      "git clone %h",
      "cd %r",
      "cp -p --parents $(git ls-files) /srv/www/htdocs/vaclabpage/"
    ]
  },
  "keep":false
}
```

Zur Erklärung: Unter »tasks« können ein oder mehrere Namen von GIT-Repositorien aufgeführt werden. Jedem dieser Namen ist ein String oder ein String-Array zugeordnet. Darin enthalten sind Unix-Kommandozeilenaufrufe. »`%h`« ist ein Platzhalter für die zum Clonen des GIT-Repositorium verwendbare URL. »`%r`« steht für den Namen des Repositoriums. Mit »`"keep":false`« wird festgelegt, dass temporär erstellte Verzeichnisse nicht erhalten bleiben sollen (`"true"` ist nur für Testzwecke nützlich). Die Beschreibung zu dem NodeJS-Module »node-gitlab-hook« enthält weitere Hinweise zur Syntax in `gitlabhook.conf`:

`https://github.com/rolfn/node-gitlab-hook`

Der konkrete Ablauf im temporären Verzeichnis des Rechners (`/tmp`) zur Erneuerung der Home-Page des Vakuumlabors ist also der folgende:

1. Lokales Duplikat des GIT-Repositoriums anlegen.
2. In das Verzeichnis mit dem Namen des Repositoriums wechseln.
3. Alle relevanten Dateien zum Webserver-Verzeichnis kopieren.

Es ist zu beachten, dass alle unter »relayServer« laufenden Prozesse im Falle von openSUSE mit den Rechten des Nutzers »wwwrun« laufen.

3 Logging-Zugriff per Websocket-Protokoll – Port: 9001

Um detaillierte Informationen über den internen Ablauf beim Ansprechen der unter »relayServer« laufenden Server-Prozesse zu erhalten, kann über Port `9001` per Websocket-Protokoll Kontakt aufgenommen werden. Zu diesem Zweck steht das Kommandozeilen-Programm `vlLogging` zur Verfügung. Ohne Parameter nimmt es Kontakt zum lokal laufenden »relayServer« auf. Wird als Parameter ein entfernter Rechner (Rechnername oder IP-Adresse) angegeben, kann auch auf dessen Informationen zugegriffen werden. Die Anzahl der kontaktierten Prozesse ist nicht beschränkt. Es wird das folgende Ausgabeformat geliefert:

```
2013-10-17 08:53:10.862 - LEVEL: [FILE_NAME:LINE_NUMBER:FUNCTION_NAME] MESSAGE
```

- LEVEL: »debug«, »error«, »warn« oder »info«
- FILE_NAME: Die Datei, in der sich der abgearbeitete Code befindet.
- LINE_NUMBER: Aktuelle Zeilennummer in FILE_NAME
- FUNCTION_NAME: Funktion, in der sich der abgearbeitete Code befindet.
- MESSAGE: Konkrete Informationen zum betreffenden Code-Teil.

Diese Informationen dienen der Fehlersuche im Programmcode von »relayServer« und auch zur allgemeinen Beobachtung des Ablaufes beispielsweise bei der Kommunikation mit Messgeräten. Die Angabe LEVEL wird farblich hervorgehoben. Die rot gekennzeichneten Fehlermeldungen fallen besonders auf. Künftig könnte der Websocket-Zugriff auch von einer Web-Applikation (Web-Browser) aus erfolgen.

Hinweise zu Verbesserungen dieses Dokuments bitte als Issue-Eintrag des git-Projektes »relayServer« (siehe »GitLab«) oder per E-Mail an Rolf.Nieprasch@ptb.de.