m23-autoTest

Hauke Goos-Habermann

Inhaltsverzeichnis

Einleitung

m23-autoTest wurde entwickelt, um m23-Funktionen automatisch in vielen Kombinationen aus m23-Client- und m23-Server-Konfigurationen massenhaft zu testen.

Mit *m23-autoTest* konnten erstmals m23-Server auf verschiedenen Plattformen (Debian 9 und 8 (jeweils 32- und 64-Bit), UCS 4.3, UCS 4.4, lokales m23-Entwicklungssytem, Raspberry Pi, m23-Installations-ISO) mit m23-Clients aller unterstützten Clientdistributionen systematisch untersucht werden. So konnten auch Fehler gefunden werden, die nur sporadisch auftreten.

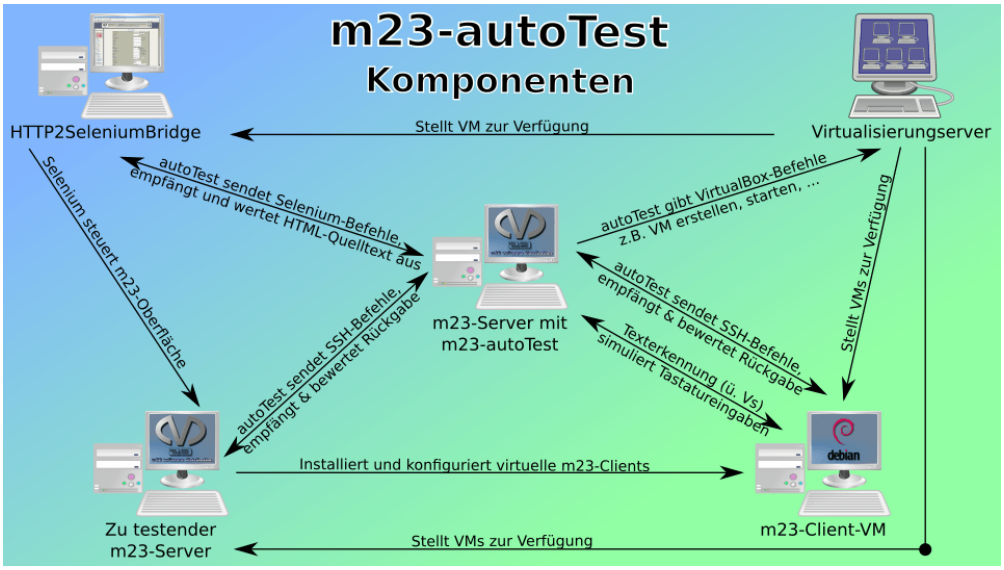
m23-autoTest simuliert hierbei das Verhalten eines menschlichen Administrators durch *Fernsteuerung* (Selenium) der m23-Weboberfläche. So geschieht z.B. das Anlegen von m23-Clients, das Partitionieren und Formatieren oder das Installieren von Distributionen mit grafischen Oberflächen über das automatische Ausfüllen der Formulare in den dazugehörigen Dialogen.

Ein *Testdurchlauf* beinhaltet die Installation *eines m23-Servers* und (aktuell) *18* durch diesen installierten *m23-Clients*. Pro Distribution werden zwei Clients als 32- und 64-Bit-Variante mit zufällig ausgewählten Desktops und abwechselnd mit deutschen, englischen und französischen Spracheinstellungen installiert.

Alle Test werden - für eine schnelle Durchführung und Wiederholbarkeit - in virtuellen Maschinen vorgenommen. Über diverse *Meßpunkte* (z.B. Meldungen auf dem Bildschirm der VM, die durch Texterkennung identifiziert werden, Nachrichten in der m23-Oberfläche oder Rückgabewerte von in der VM ausgeführten Kommandos) wird laufend der *Installationstatus überprüft und bewertet*. Diese ermöglichen es m23-autoTest, auf Ereignisse z.B. mit simulierten Tastendrücken oder Aktionen in der m23-Weboberfläche zu reagieren. Im Falle von kritischen Fehlern (z.B. nichtlaufende Serverdienste) kann m23-autoTest den Testlauf auch ganz abbrechen. Für die *anschließende Analyse* wird das Geschehen auf dem Bildschirm der VM in einer *Videodatei* aufgezeichnet, sowie ein *Installationsprotokoll* geschrieben, welches im Debug-Modus viele zusätzliche Informationen enthält.

Auch wenn der Funktionsumfang auf die Belange von m23 ausgerichtet ist, kann aber dennoch für andere Projekte nützlich sein.

Diese Dokumentation beschreibt die Installation und Konfiguration der Testumgebung und einzelnen Komponenten sowie die Benutzung des steuernden Kommandozeilenprogrammes `autoTest.php` nebst Erläuterung des Datenformates der *XML-Testbeschreibungsdateien*. Diese Testbeschreibungsdateien bündeln alle Aktionen, Kommandos und erwarteten Ergebnisse, die für einen kompletten Testlauf (z.B. Installation eines m23-Debian-Clients mit Mate-Deskop und französischen Spracheinstellungen, anschließender Nachinstallation des Midnight Commanders und Überprüfung eines LDAP-Benutzers) benötigt werden.



autoTest-Umgebung: Installation und Konfiguration

Installation

Die folgende Auflistung beschreibt die Funktionen der einzelnen Komponenten, die benötigt werden, um m23-Funktionen automatisiert mit autoTest durchzuführen. Bis auf den Virtualisierungsserver können die Komponenten in VMs installiert werden. Prinzipiell könnten z.B. Virtualisierungsserver, HTTP2SeleniumBridge und der autoTest-Steuersystem auf derselben (physikalischen) Maschine laufen.

autoTest-Steuersystem

Ein m23-Server, der autoTest ausführt und die anderen Systeme (Virtualisierungsserver, HTTP2SeleniumBridge und (indirekt) den zu testenden m23-Server) steuert.

Installation

- Ein normal installierter m23-Server
- Einen autoTest-Skript-Benutzer anlegen mit zugehörigem SSH-Schlüssel anlegen, der die autoTest-Skripte ausführen soll
- `settings.m23test` anpassen

Virtualisierungsserver

Ein Server, mit ausreichenden Kapazitäten (RAM, CPUs, Festplattenplatz), auf dem VirtualBox skriptgesteuert ausgeführt werden kann.

Installation

- Debian oder Ubuntu in 64-Bit-Version installieren
- VirtualBox (5.2.x empfohlen), tesseract, convert (ImageMagicl) und gocr installieren
- Einen Benutzer anlegen, der über `VBoxManage` virtuelle Maschinen anlegen und starten kann. Dazu den Benutzer in die Gruppe `vboxusers` aufnehmen. Den SSH-Schlüssel des autoTest-Skript-Benutzers importieren. Zum Ausführen von Tests muß dieser Benutzer grafisch (lokal oder per x2go) eingeloggt sein. In der X-Sitzung dieses Benutzers werden die VirtualBox-Fenster gestartet. Der Benutzer darf nur einmalig grafisch eingeloggt werden, da ansonsten die zu verwendenden X-Sitzung nicht zuverlässig erkannt werden kann.
- (Optional) x2go installieren

m23-Server: paßwortloser Zugriff vom autoTest-ausführenden Benutzer auf root per SSH oder sudo?

HTTP2SeleniumBridge

Zu testender m23-Server

Ein m23-Server, der verwendet wird, um einen m23-Client zu installieren. Wird von der HTTP2SeleniumBridge über die m23-Weboberfläche gesteuert.

Installation

- Ein normal installierter m23-Server

(Optional) Debian-Minimal-VMs zum Testen der m23-Server-Pakate

- Debian-32Bit und Debian-64Bit VMs in VirtualBox anlegen:
  - Netzwerkbrücke
  - keine Soundkarte
  - 15GB Festplatte
  - PAE/NX bei i386 aktivieren
- Die beiden VMs mittels der Debian-Netinst-ISOs installieren
  - IP nach dem Schema: 192.168.1.<32Bit = 3, 64Bit = 64>
    z.B. Debian 9 32Bit: 192.168.1.93
  - Paßwort immer “test”. Benutzer “test”
  - Softwareauswahl *NUR*: SSH und Standardwerkzeuge
- Nach Abschluß der Installation die VM neu starten

- In `/etc/ssh/sshd_config` “PermitRootLogin yes” setzen
- Festplatte säubern:
  - `apt-get clean; dd if=/dev/zero of=/z; rm /z; poweroff`
- Sicherungspunkt mit Namen “vor” erstellen
  - Beschreibung: Standard-Debian-Netzwerinstallation (nur SSH und Standardwerkzeuge)
- VMs als OVA exportieren

(Optional) UCS-VM zum Testen der m23-Server-Pakate

- Debian-64Bit VM in VirtualBox anlegen:
  - Netzwerkbrücke
  - keine Soundkarte
  - 50GB Festplatte
- Die VM mittels des UCS-ISOs installieren
  - IP nach dem Schema: 192.168.1.1
    z.B. UCS 4.4: 192.168.1.144
  - Paßwort (temporär) “testtest”.
  - Konfiguration: Neue Domäne, keine Dienste installieren
- Nach Abschluß der Installation die VM neu starten
- Als root einloggen und per `passwd` das Paßwort auf “test” setzen
- Festplatte säubern:
  - `apt-get clean; dd if=/dev/zero of=/z; rm /z; poweroff`
- Sicherungspunkt mit Namen “vor” erstellen
  - Beschreibung: UCS X.Y frisch installiert
- VMs als OVA exportieren

(Optional) Raspberry Pi

- Partitionen aus dem Abbild auf Gerätedateien umlenken: `kpartx -a -v 20XX-YY-ZZ-raspbian-stretch-lite.img`
- Boot-Partition einhängen: `mount /dev/mapper/loop0p1 /mnt/loop/`
- SSH-Server beim Booten aktivieren: `touch /mnt/loop/ssh`
- Erste Partition aushängen: `umount /mnt/loop`
- System-Partition einhängen: `mount /dev/mapper/loop0p2 /mnt/loop/`
- In `/mnt/loop/etc/ssh/sshd_config` “PermitRootLogin yes” setzen
- In `/mnt/loop/etc/shadow` die bestehende “root”-Zeile ersetzen durch (Paßwort = test):

  root:6Lc6S7BJt$/Rc4cLBYjzK012BxttQOnPpa9Sh3IEuxEAyzf/oKcfwwQKOqvadHI0uEhDrkeQRsarJuqP3d0I1Z0/Y
- In `/mnt/loop/etc/shadow` die bestehende “pi”-Zeile ersetzen durch (Paßwort = test):

  pi:6rMmaUT.QhJnCOC5L$T6kh.rOkTnc6TSllFa5F1qUF3DU9wQNJagEvMTtopg6cpCU3HihGY8EsndlHMjSAwS9d
- In `/mnt/loop/etc/network/interfaces` folgendes einfügen:

allow-hotplug eth0 iface eth0 inet static address 192.168.1.122 netmask 255.255.255.0 network 192.168.1.0 broadcast 192.168.1.255 gateway 192.168.1.5 dns-nameservers 192.168.1.5

- In `/mnt/loop/etc/resolv.conf` folgendes einfügen: nameserver 192.168.1.5
- Zweite Partition aushängen: `umount /mnt/loop`
- Umlenkung aufheben: `kpartx -d -v 20XX-YY-ZZ-raspbian-stretch-lite.img`
- Abbild komprimieren: `bzip2 -9 20XX-YY-ZZ-raspbian-stretch-lite.img`

HTTP2SeleniumBridge: Beispiele und Installation

HTTP2SeleniumBridge.py ist ein Python-Skript, das einen Webserver auf Port 23080 öffnet, um ausgewählte Selenium-Befehle via REST-API auszuführen. Der Funktionsumfang beinhaltet alles, was nötig ist, um die m23-Oberfläche zu steuern (z.B. m23-Clients anlegen und Betriebssysteme installieren). *HTTP2SeleniumBridge.py* kann zwar auch *eigenständig* verwendet werden, wurde aber konzipiert, um über die PHP-AUTOTEST-Funktionen bzw. CAutoTest-Klassenmethoden von m23 aufgerufen zu werden.

Beispiele

Die Beispiele gehen davon aus, daß HTTP2SeleniumBridge auf einem Rechner mit der IP 192.168.1.153 läuft. Das Ergebnis jeder Operation wird in die Datei `s.log` geschrieben.

```
# Freie Selenium-Webdriver-ID abfragen. Muß bei jedem weiteren Aufruf angegeben werden!
wget 'http://192.168.1.153:23080/nextdriverid' -O s.log
# Öffnen der m23-Oberflächenseite
wget 'http://192.168.1.153:23080/run?cmd=open&url=https://god:m23@192.168.1.143/m23admin/\
index.php&driverid=ID' -O s.log
# Auswählen von "de" aus der Liste verfügbarer Sprachen
wget 'http://192.168.1.153:23080/run?cmd=selectFrom&ID=LB_language&val=de&driverid=ID' -O s.log
# Klick auf den Button
wget 'http://192.168.1.153:23080/run?cmd=clickButton&name=BUT_lang&driverid=ID' -O s.log
# Simulieren der Texteingabe in das Feld "ED_name"
wget 'http://192.168.1.153:23080/run?cmd=typeInto&ID=ED_name&text=vorname&driverid=ID' -O s.log
# Haken beim NTP-Checkbutton entfernen
wget 'http://192.168.1.153:23080/run?cmd=setCheck&name=CB_getSystemtimeByNTP&checked=0\
&driverid=ID' -O s.log
# Beim LDAP-Radiobutton "write" auswählen
wget 'http://192.168.1.153:23080/run?cmd=selectRadio&name=SEL_ldaptype&val=write\
&driverid=ID' -O s.log
# Aktuellen HTML-Quelltext herunterladen
wget 'http://192.168.1.153:23080/run?cmd=getsource&driverid=ID' -O s.log
```

Installation

Die Installation wurde unter einem mit m23 aufgesetzen 64-Bit-m23-Client mit Ubuntu 18.04 und Budgie-Desktop getestet. Prinzipell spricht nichts dagegen, daß HTTP2SeleniumBridge.py und das HTTP2SeleniumBridge-Paket auch unter anderen Distributionen und Desktops funktioniert. Das dazugehörige HTTP2SeleniumBridge-Debian-Paket richtet das System so ein, daß HTTP2SeleniumBridge unter dem Benutzer *"sel"* automatisch bei jedem Systemstart gestartet wird.

Installation der Distribution

Es wird **nachdrücklich** empfolen, für HTTP2SeleniumBridge eine eigene VM zu verwenden und die Distribution mit folgenden Parametern zu installieren:

- 64-Bit-Ubuntu 18.04 (z.B. über m23) installieren
- Als Anmeldungsname "sel" für den Hauptbenutzer wählen
- Optional x2go (z.B. über m23) mitinstallieren

Nach der Installation der Distribution

- GeckoDriver von https://github.com/mozilla/geckodriver/releases/latest herunterladen (z.B. geckodriver-v0.23.0-linux64.tar.gz) und installieren:

```
wget https://github.com/mozilla/geckodriver/releases/download/v0.23.0/\
geckodriver-v0.23.0-linux64.tar.gz -O geckodriver.tar.gz
tar xfvz geckodriver.tar.gz
mv geckodriver /usr/bin/
```

- Das HTTP2SeleniumBridge-Debian-Paket von https://sourceforge.net/projects/dodger-tools/files/debs/20XX-YY-ZZ/HTTP2SeleniumBridge_..._all.deb nach /tmp herunterladen und installieren mit:

```
adduser sel

export DEBIAN_FRONTEND=noninteractive
echo 'lightdm shared/default-x-display-manager select nodm
nodm nodm/daemon_name string /usr/sbin/nodm
nodm nodm/enabled boolean true
nodm nodm/first_vt string 7
nodm nodm/min_session_time string 60
nodm nodm/user string sel
nodm nodm/x_options string -nolisten tcp
nodm nodm/xsession string /etc/X11/Xsession
nodm nodm/x_timeout string 300
nodm shared/default-x-display-manager select nodm' | debconf-set-selections

echo /usr/sbin/nodm > /etc/X11/default-display-manager

apt install -y /tmp/HTTP2SeleniumBridge_1.00-*_all.deb

apt remove -y gnome-screensaver

reboot
```

Hinweis zu HTTP2SeleniumBridge.py

HTTP2SeleniumBridge.py verwendet das Selenium-Modul aus dem pip3-Repository. Dieses kommuniziert über den GeckoDriver mit Firefox. Jedes dieser Einzelteile kann plötzlich und unvorhergesehen so geändert (z.B. durch Aktualisierung) werden, daß es allein oder mit den anderen zusammen nicht mehr funktioniert. Daher sollte die virtuelle Maschine nach erfolgreicher Installation **unbedingt gesichert** werden.

Hinweis zum HTTP2SeleniumBridge-Paket

Das HTTP2SeleniumBridge-Paket funktioniert nur, wenn es einen Benutzer mit dem Namen *"sel"* gibt, da es das System so konfiguriert, daß sich *"sel"* automatisch über `nodm` anmeldet. Ein Autostart-.desktop-Datei sorgt wiederum dafür, daß HTTP2SeleniumBridge.py nach dem Anmelden gestartet und jedesmal neu gestartet wird, wenn HTTP2SeleniumBridge.py abstürzt.

Fehlendes

In HTTP2SeleniumBridge.py sind zusätzlich folgende Kommandos implementiert, die aber nicht in CAutoTest.php verwendet werden:

- close
- deselectFrom
- quit

Testparcours mit autoTestScriptGenerator.php erstellen

Das Skript `autoTestScriptGenerator.php` erstellt BASH-Dateien, die auf m23-Server-Plattformen (z.B. Debian 9 und 8 (32- und 64-Bit), UCS 4.3, UCS 4.4, m23-Installations-ISO, ...) Tests durchführen. Hierbei werden auf der Ziel-VM die m23-Serverpakete installiert oder ein neuer m23-Server mit dem m23-Serverinstallations-ISO aufgesetzt. Ein *Testdurchlauf* beinhaltet die Installation *eines m23-Servers* und (aktuell) *18* durch diesen installierten *m23-Clients*. Pro Distribution werden zwei Clients als 32- und 64-Bit-Variante mit zufällig ausgewählten Desktops und abwechselnd mit deutschen, englischen und französischen Spracheinstellungen installiert.

`autoTestScriptGenerator.php` kann mit dem Namen einer Paketquellenliste als optionalen Parameter aufgerufen werden. Wird dieser gesetzt, so enthalten die generierten Skripte ausschließlich Tests für die angegebene Distribution allerdings mit allen verfügbaren Desktops.

Zusätzliche Eigenschaften der BASH-Skripte

- Die so generierten BASH-Skripte erzeugen Logdateien, die bis auf die Dateiendung (`.log` statt `.sh`) wie die BASH-Skripte heißen.
- Gibt es im Verzeichnis eines BASH-Skriptes eine Stop-Datei (Dateiendung `.stop`), so wird die aktuell laufende Clientinstallation bis zum Ende ausgeführt, aber keine neue mehr begonnen.

autoTest.php-Kommandozeile und XML-Definition

Installation und Konfiguration

Für autoTest.php wird eine Testumgebung benötigt. Deren Installation und Konfiguration wird weiter in den vorigen Abschnitten beschrieben.

autoTest starten

Das Starten von autoTest geschieht über das Skript `autoTest.php`, welches auf jedem m23-Server vorhanden ist:

`/mdk/autoTest/autoTest.php <Testbeschreibungsdatei (.m23test)> <Parameter>`

Der Ablauf einer jeden Installation ist in einer Testbeschreibungsdatei beschrieben, die zusätzliche Parameter über die Kommandozeile anfordern kann.

Testbeschreibungsdateien

Die Testbeschreibungsdateien mit der Endung *".m23test"* beinhalten Testblöcke, die die einzelnen Schritte (z.B. Anlegen der virtuellen Maschine, in der m23-Oberfläche einzugebende Werte bzw. anzuklickende Elemente, ...) und (erwartete) Ergebnisse zum Installieren eines m23-Clients oder -Servers enthalten. Andere Teile der Datei definieren die Parameter, die über die Kommandozeile angeben werden und die Dimensionierung der anzulegenden virtuellen Maschine.

Mitgelieferte Testbeschreibungsdateien

Im Verzeichnis `/mdk/autoTest/` befinden sich auf dem m23-Server diverse Testbeschreibungsdateien. Beispielhaft seien die folgenden erwähnt:

- `1m23client-distro-install.m23test`: Installiert einen m23-Clienten in einer neuen VM (löscht ggf. eine zuvor vorhandene VM mit selben Namen nebst dazugehörigem Eintrag aus der m23-Weboberfläche) mit grafischer Oberfläche. Erkennt den Login-Manager und installiert und deinstalliert den Midnight Commander.
- `1m23server-auf-debian-installieren.m23test`: Installiert die m23-Server-Pakete von 192.168.1.77 auf einer gesicherten VM (Sicherungspunkt "vor").
- `1m23server-iso-install.m23test`: Installiert das m23-Server-Installations-ISO in einer VM.

Allgemeiner Aufbau

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<testcase>
    <variables>
    <TEST_TYPE>VM</TEST_TYPE>
        <VM_RAM>1024</VM_RAM>
        <VM_HDSIZE>8192</VM_HDSIZE>
    </variables>
    <cli>
        <VM_NAME description="Name der VM"></VM_NAME>
        <OS_PACKAGESOURCE description="Paketquellenliste"></OS_PACKAGESOURCE>
        <OS_DESKTOP description="Desktop"></OS_DESKTOP>
        <VM_NAME description="Name der VM"></VM_NAME>
        <VM_IP description="IP der VM"></VM_IP>
    </cli>
    <sequence>
        <test timeout="180" vmScreenChangeIntervall="60" description="Client anlegen">
            <trigger type="sel_hostReady"></trigger>
            <action type="sel_open">${TEST_M23_BASE_URL}/index.php?page=addclient%26clearSession=1</a
            <action type="sel_typeInto" ID="ED_login">test</action>
            <action type="sel_selectFrom" ID="SEL_boottype">pxe</action>
            <action type="sel_setCheck" name="CB_getSystemtimeByNTP">0</action>
            <action type="sel_selectRadio" name="SEL_ldaptype">read</action>
            <action type="sel_clickButton" name="BUT_submit"></action>
            <good type="sel_sourcecontains">$I18N_client_added</good>
```

```
            <warn type="sel_sourcecontains">unwichtig</warn>
            <bad type="sel_sourcecontains">$I18N_addNewLoginToUCSLDAPError</bad>
        </test>
        <include>langDe.m23testinc</include>
        <test timeout="600" description="VM erstellen und starten">
            <trigger type="true"></trigger>
            <action type="fkt">AUTOTEST_VM_create</action>
            <action type="fkt">AUTOTEST_VM_start</action>
            <good type="ocr">|{Warte|minutes}</good>
        </test>
    </sequence>
</testcase>
```

Begriffserklärung

Die einzelnen Zeilen sind folgendermaßen aufgebaut, wobei die Begriffe *Tag*, *Attribut* und *Parameter* verwendet werden:

```
<Tag Attribut1="..." Attribut2="...">Parameter</Tag>
```

Variablen und Konstanten

Grundlegende Test-Einstellungen stehen in der globalen Datei `settings.m23test` sowie in der aktuellen m23test-Datei. `settings.m23test` wird zuerst im Heimatverzeichnis des Benutzer gesucht, der `autoTest.php` startet. Wird `settings.m23test` nicht gefunden, wird die Datei im aktuellen Verzeichnis gesucht.

Die Einstellungen werden als internen autoTest-Variablen *und* als Konstanten (für Rückwärtskompatibilität) gespeichert. Die Dopplung wird aktuell benötigt, damit die Werte in den Bedingungen der `runIf`-Attribute verwendet werden können.

Intern verwendete Variablennamen:

- TEST_SELENIUM_URL: Die URL, um auf die HTTP2SeleniumBridge zuzugreifen. z.B. http://192.168.1.153:23080
- TEST_VBOX_HOST: Auflösbarer Hostname oder IP des Systems, auf dem die VirtualBoxen laufen sollen. z.B. `tuxedo`
- TEST_VBOX_USER: Benutzer (muß in der Guppe *vboxusers* sein), der `vboxmanage` zum Erstellen, Starten, etc. aufruft.
- TEST_VBOX_NETDEV: Netzwerkschnittstelle, die der echten Netzwerkkarte entspricht und zum Anlegen der Netzwerbrücke verwendet werden soll. z.B. `enp1s0f0`
- TEST_VBOX_IMAGE_DIR: Verzeichnis auf dem VirtualBox-Gastegebersystem, in dem die virtuellen Maschinen gespichert werden sollen. z.B. `/media/vms/`
- TEST_M23_BASE_URL: Komplette URL mit Benutzer und Paßwort zur m23-Weboberfläche. z.B. `https://god:m23@192.168.1.143/m23admin`
- TEST_M23_IP: Die aus `TEST_M23_BASE_URL` extrahierte IP-Adresse.
- TEST_VBOX_MAC: Beim Starten zufällig generierte MAC-Adresse mit ":" als Trenner nach jeweils zwei Zeichen. z.B. `aa:bb:cc:dd:ee:ff:00:11`
- SEL_VM_MAC: Dieselbe Zufalls-MAC, allerdings ohne den Trenner. z.B. `aabbccddeeff0011`
- TEST_TYPE: "VM", wenn VirtualBox verwendet wird. Soll nur die m23-Oberfläche getestet werden: "webinterface". Zum alleinigen Testen der XML-Testbeschreibungsdatei: "xmltest".
- VM_RAM: RAM-Größe der VM in MB.
- VM_HDSIZE: Größe der virtuellen Festplatte in MB.
- VM_IP: IP-Adresse, die genutzt werden soll, um die VM per SSH anzusprechen. Ansonsten wird der Name der VM (VM_NAME) als Hostname verwendet.
- VM_NAME: Name der VM und ggf. Hostname zum Ansprechen der VM per SSH.
- AT_debug: Gesetzt, wenn autoTest im Debug-Modus ist und zusätzliche Informationen ausgeben soll.
- AT_M23_SSH_PASSWORD: Das Paßwort, um den m23-Server per SSH als Benutzer root zu erreichen.

In der `settings.m23test` sollten minimal folgende Konstanten gesetzt sein:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<settings>
    <variables>
```

```
        <VM_RAM>1024</VM_RAM>
        <VM_HDSIZE>8192</VM_HDSIZE>
        <TEST_VBOX_HOST>vmhost</TEST_VBOX_HOST>
        <TEST_VBOX_USER>vboxbenutzer</TEST_VBOX_USER>
        <TEST_VBOX_NETDEV>enp1s0f0</TEST_VBOX_NETDEV>
        <TEST_VBOX_IMAGE_DIR>/media/vms/</TEST_VBOX_IMAGE_DIR>
        <TEST_SELENIUM_URL>http://192.168.1.153:23080</TEST_SELENIUM_URL>
        <TEST_M23_BASE_URL>http://god:m23@192.168.1.143/m23admin</TEST_M23_BASE_URL>
    </variables>
</settings>
```

Testblöcke

Ein Testblock umfaßt immer alle Teile eines Tests, die folgendermaßen abgearbeitet werden:

1. Die Bedingung des trigger-Tags wird solange wiederkehrend überprüft, bis diese zutrifft oder das Zeitlimit überschritten ist. Bei einem Überschreiten wird das Skript abgebrochen.
2. Die einzelnen action-Tags werden in der angegbenen Reihenfolge abgearbeitet, wenn die Bedingung des trigger-Tags zutraf.
3. Die good/warn/bad-Tags werden immer wieder durchlaufen, bis eine Bedingung zutrifft. `bad` führt zum Abbruch, die anderen (nur) zu einem Eintrag in die Logdatei und Ausführen des nächsten Testblocks.

`timeout` (in Sekunden) gibt an, wie lange auf den Trigger und das Abschließen durch ein good-Tag gewartet werden soll. Nach Überschreiten um mehr als zwei Minuten wird eine Warnung ausgegeben, nach mehr als 5 Minuten wird das Skript mit einem Fehler abgebrochen.

`vmScreenChangeIntervall` gibt das Zeitintervall (in Sekunden) zwischen dem Erstellen von zwei Screenshots der VM an. Nach Ablauf der Zeit werden die Screenshots pixelweise miteinander verglichen und die die Anzahl der abweichenden Pixel ermittelt. Liegt die Anzahl unter 50 Pixeln, so geht m23-autoTest davon aus, daß die VM nicht mehr reagiert.

`description` ist die Beschreibung, die in den Logdateien vermerkt wird.

```
    <test timeout="600" description="VM erstellen und starten">
        <trigger type="true"></trigger>
        <action type="fkt">AUTOTEST_VM_create</action>
        <action type="fkt">AUTOTEST_VM_start</action>
        <good type="ocr">|{Warte|minutes}</good>
    </test>
```

Kommandozeilenparameter

Die im `cli`-Block definierten Tags müssen in derselben Reihenfolge auf der Kommandozeile angegeben werden. Der jeweilige Tag-Name wird als Konstante gespeichert und kann in den Ersetzungen verwendet werden. `VM_NAME` wird intern für die Aufrufe von einige Funktionen z.B. `AUTOTEST_VM_keyboardWrite` oder `AUTOTEST_sshTunnelOverServer` verwendet und muß in den meisten Fällen angegebn werden.

Das Attribut `description` ist die Beschreibung des jeweiligen Tags/Kommandozeilenparameters, die ausgegeben wird, wenn nicht die korrekte Anzahl an Parametern übergeben wird.

Beispiel:

```
    <cli>
        <VM_NAME description="Name der VM"></VM_NAME>
        <OS_PACKAGESOURCE description="Paketquellenliste"></OS_PACKAGESOURCE>
        <OS_DESKTOP description="Desktop"></OS_DESKTOP>
    </cli>
```

Ersetzungen

Innerhalb des Parameters können Teile ersetzt oder für Suchen verwendet werden:

- `${...}`: "..." wird durch den Wert einer vorher definierte Konstante ersetzt.

- `|{str1|str2|str3}`: str1 ... str3 sind alternative Zeichenketten, von denen beim Vergleichen nur eine übereinstimmen muß.
- `$I18N_...`: Wird nacheinander durch die Übersetzungen in allen Sprachen ersetzt und jeweils verglichen. Hierbei muß nur eine Übersetzung übereinstimmen.
- `!`: Bei `good/warn/bad` kann die Bedingung durch ein vorgestelltes "!" umgekehrt werden.
  Die folgende Bedingung trifft zu, wenn die Zeichenkette "C0C" NICHT gefunden wurde: `<bad type="ssh_commandoutput" password="test" sshanswer="!C0C" description="Ausfall">cat m</bad>`
- `<include>DATEI</include>`: Fügt den Inhalt der angegebene Datei an der Stelle dynamisch ein.

Bedingtes Ausführen von test-Blöcken

m23-autoTest bietet die Möglichkeit, test-Blöcke nur dann auszuführen, wenn interne Variablen oder Umgebungsvariablen einen bestimmten Wert haben.

Das Setzen der internen Variablen geschieht auf zwei Wegen:

1. Mittels des Attributes `setVar` beim Auslösen eines good/warn/bad-Ereignisses
2. Durch (BASH-)Umgebungsvariablen beim Aufruf von `autoTest.php`, die, falls sie mit "AT_" beginnen, in den internen Variablenspeicher importiert werden.

test-Blöcke werden ausgeführt wenn:

1. sie kein `runIf`-Attribut besitzen
2. die Bedingung des `runIf`-Attributes zutriff

Bedingungen sind die folgenden Vergleichsoperationen:

- `>`: Interne Variable größer als Vergleichswert (Zahl)
- `>=`: Interne Variable größer als oder gleich dem Vergleichswert (Zahl)
- `==`: Interne Variable und Vergleichswert sind identisch (Zahl oder Zeichenkette)
  - **Sonderfall**: Ist der Vergleichswert "NULL", so wird nur überprüft, ob die interne *nicht* Variable gesetzt ist.
    Beispiel: `runIf="AT_test==NULL"` (Ausführen, wenn AT_test *nicht* einen Wert hat)
- `<`: Interne Variable kleiner als Vergleichswert (Zahl)
- `<=`: Interne Variable kleiner oder gleich dem Vergleichswert (Zahl)
- `!=`: Interne Variable ungleich dem Vergleichswert (Zahl oder Zeichenkette)
  - **Sonderfall**: Ist der Vergleichswert "NULL", so wird nur überprüft, ob die interne Variable gesetzt ist.
    Beispiel: `runIf="AT_test!=NULL"` (Ausführen, wenn AT_test einen Wert hat)

Beispiel: Aufruf von autoTest.php mit Umgebungsvariablen

```
AT_deleteClient=1 ./autoTest.php 1VariablenKonstanten-test.m23test blasadkfbasldfkb
```

Beispiel: XML-Testbeschreibung mit Variablen und Bedingungen

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<testcase>
    <variables>
        <TEST_TYPE>webinterface</TEST_TYPE>
        <VM_RAM>1024</VM_RAM>
        <VM_HDSIZE>8192</VM_HDSIZE>
    </variables>
    <cli>
        <VM_NAME description="Name der VM"></VM_NAME>
    </cli>
    <sequence>
        <test timeout="180" description="Client in Löschliste suchen" runIf="AT_deleteClient==1">
            <trigger type="true"></trigger>
            <action type="sel_open">${TEST_M23_BASE_URL}/index.php?page=clientsoverview%26action=delete</action>
            <good type="sel_sourcecontains" setVar="INT_deleteClient=1">client=${VM_NAME}</good>
            <warn type="sel_sourcenotcontains" setVar="INT_deleteClient=0">client=${VM_NAME}</warn>
        </test>
        <test timeout="180" description="Client löschen wenn gefunden" runIf="INT_deleteClient==1">
```

```
            <trigger type="true"></trigger>
            <action type="sel_clickMatchingURL">client=${VM_NAME}°page=deleteclient</action>
            <good type="sel_sourcecontains">$I18N_get_deleted</good>
        </test>
        <test timeout="600" description="Client löschen" runIf="INT_deleteClient==1">
            <trigger type="true"></trigger>
            <action type="sel_clickButton" name="BUT_delete"></action>
            <good type="sel_sourcecontains">$I18N_was_deleted</good>
        </test>
    </sequence>
</testcase>
```

Selenium-Funktionen

Hier sind trigger- und action-Tags aufgelistet, die über die HTTP2SeleniumBridge Selenium-Befehle ausführen.

Trigger/good/warn/bad

sel_hostReady (Trigger)

Wird ausgelöst, wenn HTTP2SeleniumBridge unter der `TEST_SELENIUM_URL` erreichbar ist

sel_sourcecontains (Trigger/good/warn/bad)

Wird ausgelöst bzw. sendet eine Nachricht, wenn der Parameter im aktuellen HTML-Quelltext des Selenium-Browsers gefunden wird.

- Parameter: Zu suchender Text.

sel_sourcenotcontains (Trigger/good/warn/bad)

Wird ausgelöst bzw. sendet eine Nachricht, wenn der Parameter im aktuellen HTML-Quelltext des Selenium-Browsers NICHT gefunden wird.

- Parameter: Zu suchender Text.

Action

Selenium-Aktionen benötigen (überwiegend) das Attribut `ID` oder `name` für die Identifikation des HTML-Elementes, auf das sie angewendet werden sollen.

sel_clickButton

Klickt auf einen Button.

- Parameter: Nichts

sel_open

Öffnet eine URL im Browser.

- Parameter: URL z.B. `${TEST_M23_BASE_URL}/index.php?page=addclient%26clearSession=1`. Hierbei müssen einige Zeichen URL-kodiert angeben werden (z.B: '&' => '%26').

sel_selectFrom

Wählt ein Element aus einer Drop-Down-Liste.

- Parameter: Der Wert (nicht der angezeigte Text) des auszuwählenden Elements.

sel_selectRadio

Wählt ein Element eines Radiobuttons.

- Parameter: Der Wert (nicht der angezeigte Text) des auszuwählenden Elements.

sel_setCheck

Setzt oder entfernt den Haken einer Checkbox.

- Parameter: 0 zum Entfernen des Hakens, 1 zum Setzen.

sel_typeInto

Ersetzt den Text eines Eingabefeldes (`<TEXTAREA></TEXTAREA>`, `<INPUT type="text">...</INPUT>`).

- Parameter: Einzugebender Text.

SSH-Funktionen

Trigger/good/warn/bad

ssh_commandoutput

Führt einen Befehl per SSH aus und überprüft, ob in der Ausgabe der gewünschte Text vorhanden ist. Sind autoTest-System (lokale IP) und m23-Server (Konstante: `TEST_M23_IP`) identisch, so wird der SSH-Client direkt vom autoTest-System aus aufgerufen. Ansonsten wird der Befehl mit Umweg über den m23-Server ausgeführt. Ist die Konstante `VM_IP` gesetzt, so wird die darin hinterlegte IP bzw. der Hostname zu Kontaktieren des Zielsystems verwendet. Ansonsten wird verwendet, was in der Konstante `VM_NAME` gespeichert ist.

- Parameter: Kommando, das auf dem Zielsystem ausgeführt werden soll.
- Attribut `sshanswer`: In der Ausgabe der SSH-Abfrage vorkommender Text.
- Optionales Attribut `password`: SSH-Paßwort, wenn kein SSH-Schlüssel verwendet werden soll.
- Optionales Attribut `description`: Beschreibung, die ausgegeben und ins Protokoll geschrieben wird, wenn good/warn/bad ausgelöst wird.

~~~~ {ssh_commandoutput-Parameter .xml .numberLines} cat /tmp/y cat /tmp/x ~~~~~

vsssh_commandoutput

Wie `ssh_commandoutput`, nur daß das Kommando auf dem Virtualisierungsserver ausgeführt wird.

Action

ssh_command

Funktioniert prinzipiell wie `ssh_commandoutput` mit dem Unterschied, daß nichts überprüft wird und der Befehl als XML-Parameter und nicht als Attribut übergeben wird.

- Parameter: Kommando, das auf dem Zielsystem ausgeführt werden soll.
- Optionales Attribut `password`: SSH-Paßwort, wenn kein SSH-Schlüssel verwendet werden soll.

~~~~ {ssh_commandoutput-Parameter .xml .numberLines} LC_ALL=C apt-get dist-upgrade -u &> /tmp/update.log;
echo ? > /*tmp/update.ret*; *x* =(grep "^0 upgraded" -c /tmp/update.log);
echo -n "X${x}Y" > /tmp/update.combi;
cat /tmp/update.ret >> /tmp/update.combi ~~~~~

vsssh_command

Wie `ssh_command`, nur daß das Kommando auf dem Virtualisierungsserver ausgeführt wird.

VirtualBox-Funktionen

Action

key

Sendet eine Tastensequenz (Text) an die VM (Konstante: `VM_NAME`). Nichtdruckbare Tasten (z.B. `Enter`) werden in "°" eingeschlossen. z.B. °enter°.

- Parameter: Zu sendender Text.

Trigger/good/warn/bad

ocr

Erstellt einen Screenshot der laufenden VM (Konstante: `VM_NAME`) und versucht den Text mit verschiednene `gocr`-Parametern zu erkennen. Wird im erkannten Text der Parameter gefunden, so wird der Trigger ausgelöst bzw. eine Nachricht gesendet.

- Parameter: Gewünschter Text.

Funktionen zum Aufrufen anderer Funktionen

Action

fkt

Führt unter CAutoTest::executePHPFunction aufgelistete Funktionen aus.

- Parameter: Name der unter CAutoTest::executePHPFunction aufgelisteten Funktion.

Sonstige Funktionen

Trigger/good/warn/bad

true

Wird sofort ausgelöst.

Trigger/Action

wait

Löst erst nach einer gewissen Zeit aus.

- Parameter: Zeit in Sekunden, die bis zum Auslösen gewartet werden soll.