



# **Administrator-Handbuch**

## **Release 1.8.0**

# Inhaltsverzeichnis

1. Administrator-Handbuch	3
1.1 Test-Editor konfigurieren	4
1.2 AUT Typ identifizieren	6
1.3 Bibliothek verwalten	7
1.3.1 Bibliothek pflegen	8
1.3.2 Bibliothekeneinträge automatisch erstellen	12
1.3.2.1 Ui - Scanner Konfigurieren	16
1.4 Bestehende Test-Treiber verwenden	18
1.4.1 SwingFixture	19
1.4.2 WebFixture	26
1.4.3 WebserviceRestFixture	30
1.4.4 WebserviceSoapFixture	32
1.5 Eigenen Test-Treiber entwickeln	34
1.6 Ältere Projekte migrieren	39
1.7 Tests Headless in der Konsole ausführen	50
1.8 CI-Build aufsetzen	51
1.9 Testbestand prüfen	53
1.10 Version des Testeditors und Version der Config.tpr	55
1.11 Unterstützte SVN-Version	56
2. Glossar	57
2.1 Akzeptanztest	59
2.2 AUT	60
2.3 Element-Liste	61
2.4 Fachsprache (DSL)	62
2.5 FitNesse	63
2.6 Fixture	64
2.7 Masken und Testschritte	65
2.8 Port	66
2.9 RAP	67
2.10 REST	68
2.11 SOAP	69
2.12 Szenario	70
2.13 Szenariosuite	71
2.14 Test	72
2.15 Testfall	73
2.16 Testlog	74
2.17 Teststruktur	75
2.18 Testsuite	76
2.19 Test-Treiber	77
2.20 Widget	78
2.21 XML	79
3. Release Notes	80

# Administrator-Handbuch

Der *Test-Editor* ist ein Open Source Projekt zur Erfassung und automatischen Ausführung von [Akzeptanztests](#). Die Anwendung stellt eine intuitiv zu bedienene Oberfläche bereit, so dass [Testfälle](#) auch ohne Entwickler Know-how erfasst werden können. Als Unterbau (Backend) wird das Open-Source Framework [FitNesse](#) genutzt. Der *Test-Editor* ist ein Open-Source Projekt und wird u.a. von der [akquinet](#) und der [Signal Iduna](#) entwickelt.

Das Administrator-Handbuch beinhaltet technische Details, um den Test-Editor an individuelle Projekte anzupassen, während das Benutzer-Handbuch die Funktionsweise der grafischen Oberfläche beschreibt.

Da der *Test-Editor* auf verschiedenen Systemen entwickelt wird, können die eingesetzten Screenshots vom Aussehen aus voneinander abweichen. Inhaltlich wird die Anwendung auf Windows, Mac und Linux aber identisch reagieren, was durch die unterschiedlichen Entwicklungssysteme sichergestellt wird.

# Test-Editor konfigurieren

Der *Test-Editor* ist eine Fat-Client Anwendung, die nicht im klassischen Sinne installiert werden muss, aber auf dem lokalen System nach dem Entpacken über eine Startdatei (testeditor.exe, Linux: testeditor, Mac OS: testeditor.app) gestartet wird. Dieser Abschnitt erklärt die Systemvoraussetzungen und die Konfigurationsmöglichkeiten.

## Installation

Der *Test-Editor* wird als Zip-Archiv für alle gängigen Betriebssysteme unter [www.testeditor.org](http://www.testeditor.org) bereitgestellt. Dabei werden zwei Versionen bereitgestellt:

- **inklusive JRE:** Diese Version beinhaltet bereits die passende Java Runtime Edition (JRE), d.h. auf dem Rechner muss kein Java installiert sein (wird nur für Windows bereit gestellt)
- **ohne JRE:** Diese Version ist von der Dateigröße des ZIP-Archivs kleiner, setzt allerdings eine installierte Java Runtime ab Version 7.0 oder höher voraus, siehe <http://java.com/de/download/index.jsp>

## Unterstützte Betriebssysteme

Betriebssystem	inkl. JRE	ohne JRE
Windows	<ul style="list-style-type: none"><li>▪ 32 Bit: TestEditor-win32.win32.x86.zip</li><li>▪ 64 Bit: TestEditor-win32.win32.x86_64.zip</li></ul>	<ul style="list-style-type: none"><li>▪ 32 Bit: TestEditor_JRE-win32.win32.x86.zip</li><li>▪ 64 Bit: TestEditor_JRE-win32.win32.x86_64.zip</li></ul>
Linux	---	<ul style="list-style-type: none"><li>▪ 64 Bit: TestEditor-linux.gtk.x86_64.zip</li></ul>
Mac OS X	---	<ul style="list-style-type: none"><li>▪ 64 Bit: TestEditor-macosx.cocoa.x86_64.zip</li></ul>

Das oben angegebene Zip-Archiv kann in ein beliebiges Verzeichnis extrahiert werden, anschließend kann mit Hilfe der Startdatei der *Test-Editor* gestartet werden.

## Konfiguration

### Workspace des Test-Editor ändern

Standardmäßig wird im Benutzerverzeichnis unter .testeditor ein neuer Workspace für die Projektdateien des Test-Editors angelegt. Dieser Pfad kann über den **Parameter -data** beim Starten des *Test-Editors* auf einen anderen Pfad geändert werden.

#### Beispiele Pfadangabe

```
testeditor.exe -data c:\users\Max\myworkspace (für Windows)
testeditor -data ../workspace (für Linux)
testeditor.app -data ../anyWorkspace (für Mac OS X)
```

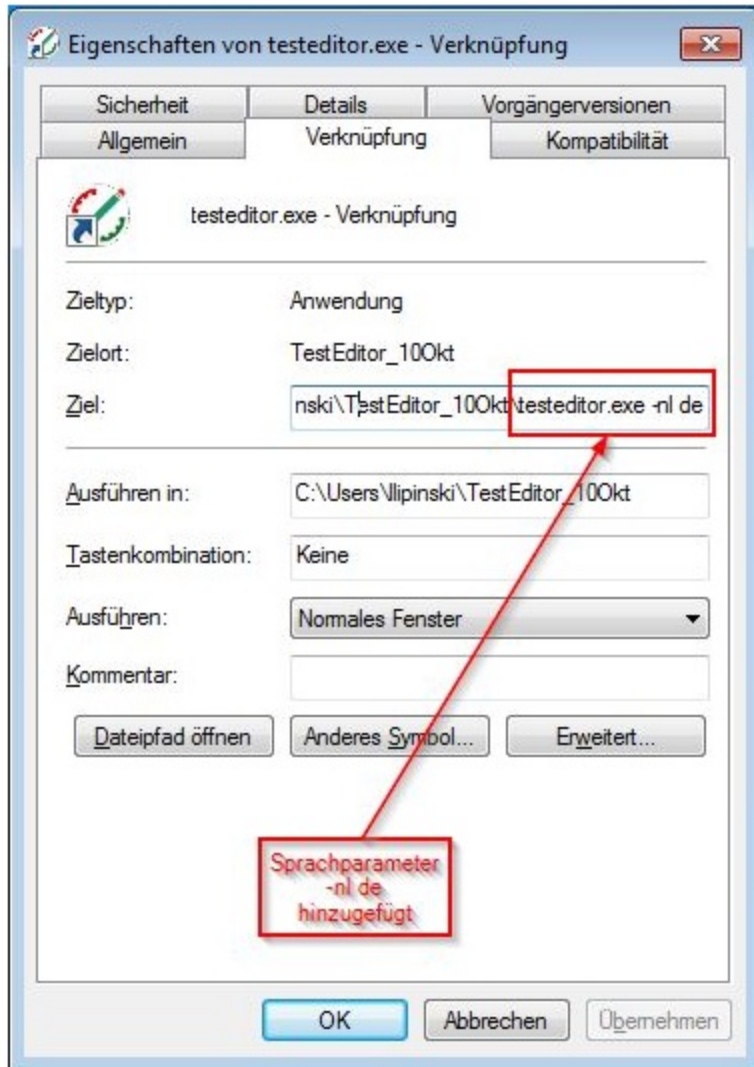
## Sprache einstellen

Der *Test-Editor* unterstützt die Sprachen **Deutsch** und **Englisch**. Standardmäßig wird die Systemsprache verwendet. Soll eine andere Sprache verwendet werden, kann dies über den **Parameter -nl** geändert werden. (z.B. zur Umstellung auf die Deutsche Sprache -nl de\_de, auf Englisch -nl en\_en).

### Beispiele Sprache einstellen

```
testeditor.exe -nl de_de (für Windows)
testeditor -data -nl de_de (für Linux)
testeditor.app -nl de_de (für Mac OS X)
```

Unter Windows lässt sich diese Einstellung wie auch die Einstellung des Workspaces über die *Test-Editor* Verknüpfung einstellen (vgl. folgender Screenshot):



# AUT Typ identifizieren

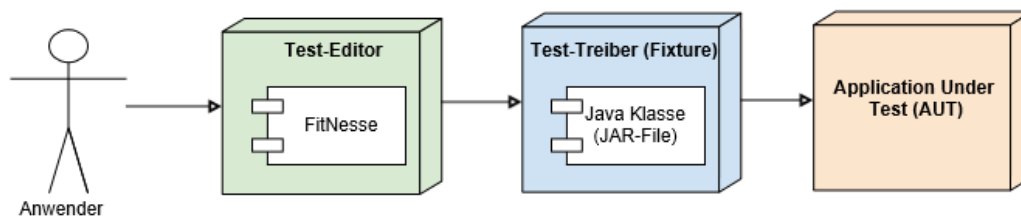
Der *Test-Editor* unterstützt verschiedene Typen von zu testenden Applikationen, z.B. Web-Anwendungen, Desktop-Anwendungen oder Webservices. In diesem Zusammenhang wird häufig der Begriff **AUT (Application Under Test)** genannt, er beschreibt das zu testende System und wird in dieser Dokumentation entsprechend verwendet.

In Abhängigkeit davon, welcher Typ von Anwendung vorliegt werden unterschiedliche **Test-Treiber** benötigt. Ein Test-Treiber (oft auch als **Fixture** bezeichnet) ist eine JAR-Datei, die ein oder mehrere **Java-Klassen** beinhaltet. Die Klassen ansich bestehen aus einfachen public-Methoden, die eine automatische Steuerung (z.B. klicke auf Button, gebe in das Textfeld ein) realisieren. Sie werden bei der Test-Ausführung aufgerufen und steuern die AUT fern. Der *Test-Editor* bietet bereits einige Test-Treiber, die je nach Bedarf erweitert werden können. Der Vorteil an der Architektur des *Test-Editors* ist es, dass eigene, Projekt-spezifische Treiber entwickelt werden und bei der Ausführung des Tests aufgerufen werden können.

Der *Test-Editor* ansich stellt jediglich eine optimiertes Benutzerinterface bereit. Intern wird das weit verbreitete Open-Source Framework **FitNesse** genutzt um die Struktur von **Testfällen**, **Suiten** etc. abzubilden und **Test** auszuführen.

Folgende Grafik veranschaulicht den generellen Aufbau:

Zusammenspiel Test-Editor -> Test-Treiber -> AUT



## Unterstützte AUT Typen

Folgende Arten von Application Under Test (AUT) werden bereits standardmäßig durch den *Test-Editor* unterstützt:

AUT Typ	Erfahrung	Test-Framework	Fixture
Web-Anwendungen	Web-Anwendungen werden durch ein umfangreiches Basis-Set an Funktionen unterstützt. Für Web-Anwendungen, die mit dem Eclipse Rich Application Platform (RAP) erzeugt werden, gibt es eine seperate Implementierung, die ebenfalls mit ausgeliefert wird und speziell für diese Anwendungen optimiert ist.	Selenium/Web-Driver	TestEditorFixtureWeb
Swing Fat-Client	Swing-Anwendungen werden durch ein umfangreiches Basis-Set an Funktionen unterstützt.	FEST (Swing)	TestEditorFixtureSwing
SWT/RCP Fat-Client	RCP Anwendungen basierend auf Eclipse E4 werden unterstützt, allerdings ist spezielles Know-How notwendig um eine Intergration zu realisieren. Technisch können auch Anwendungen mit älteren Versionen bzw. nur SWT realisiert werden, dafür ist eine Erweiterung der Fixture-Implementierung notwendig.	SWTBot	TestEditorFixtureSWT
Webservices	Für Webservice (REST, SOAP) gibt es einen generischen Ansatz, der zeigt wie diese Webservices getestet werden können.	REST/SOAP Client	TestEditorFixtureRest TestEditorFixtureSoap
Mainframe-Anwendungen	Mainframe-Anwendungen werden umfangreich unterstützt, es wird jedoch ein proprietärer Treiber benötigt, weswegen die Fixture nicht im Test-Editor ausgeliefert wird.	HACL (IBM)/ Rational Host on Demand	nicht Open Source

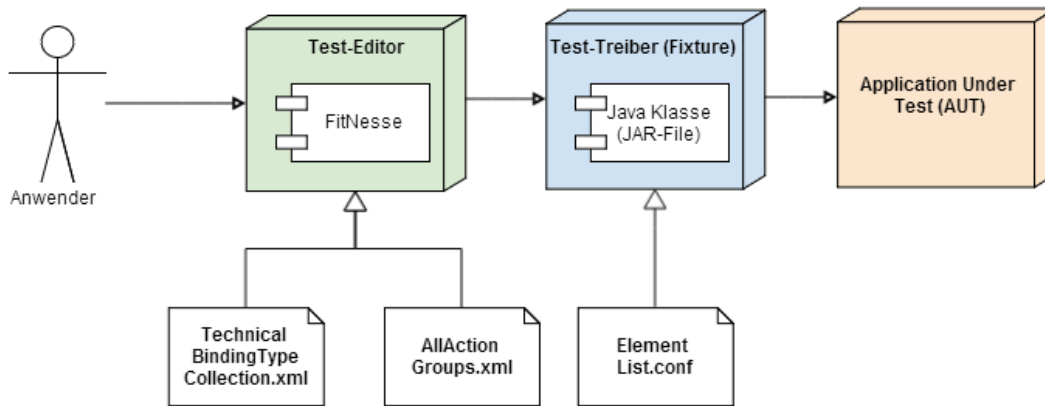
## Bibliothek verwalten

## Bibliothek pflegen

Die Bibliothek eines Projektes im *Test-Editor* ist eine Art Meta-Ebene zu einer *AUT*. Um den *Test-Editor* einsetzen zu können müssen verschiedene Bibliotheken verwaltet werden.

Folgende Grafik veranschaulicht, wo welche Konfigurationsdateien genutzt werden:

### Zusammenspiel Test-Editor -> Test-Treiber -> AUT



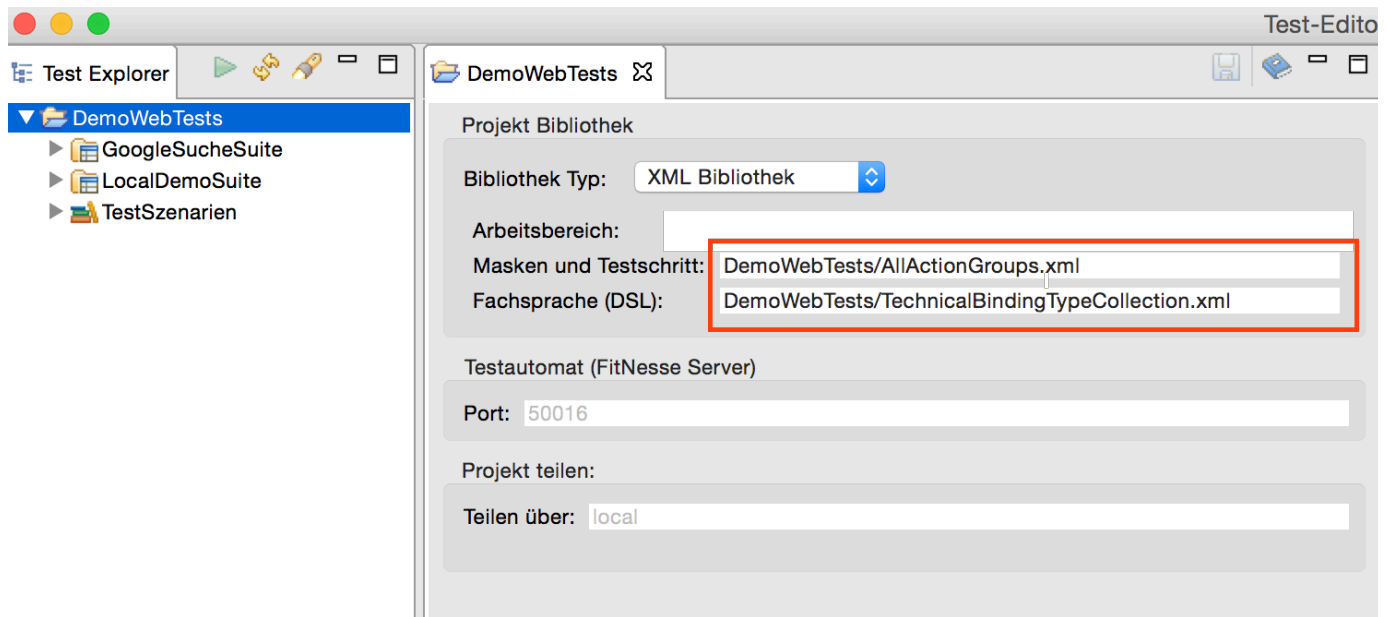
- **Fachsprache (TechnicalBindingTypeCollection.xml)** beinhaltet die Beschreibung eines Testschrittes, dadurch werden Testschritte für den Anwender leicht verständlich und selbsterklärend. Änderungen sind in der Regel nur dann vorzunehmen, wenn ein Projekt eine eigene Fachsprache entwickelt.
- **Masken und Testschritte (AllActionGroups.xml)** beschreibt welche Testschritte auf den jeweiligen Masken der AUT überhaupt möglich sind. Wichtig ist zu unterscheiden, dass hier keine konkreten Testfälle gespeichert werden, sondern nur die Meta-Informationen zu einem Projekt, also welche Aktionen überhaupt möglich sind (z.B. Klicke auf **Login**, Gebe in **Name** einen **Wert** ein usw.). Die AllActionGroups.xml verweist dabei auf den Inhalt der TechnicalBindingTypeCollection.xml.
- **Element-Liste (ElementList.conf)** besteht aus Key-Value Paaren. Sie muss ebenfalls vom Projekt gepflegt werden. Der Key ist identisch mit dem "Locator" in der ActionGroup.xml und der Value ist jeweils der technische Schlüssel um ein Element auf dem AUT zu adressieren (also z.B. eine HTML ID oder ein X-Path Ausdruck).

Wie die Bibliotheken individuell auf ein Projekt angepasst werden können wird an diesem [Beispiel](#) beschrieben.

## Pfad der Konfigurationsdateien ändern

Die TechnicalBindingTypeCollection.xml und die AllActionGroups.xml befinden sich in dem Root-Verzeichnis des entsprechenden Projektes. Der Pfad kann über die Einstellungen des Projektes konfiguriert werden:





## Aufbau TechnicalBindingTypeCollection.xml

Die **TechnicalBindingTypeCollection.xml** ist eine [XML-Bibliothek](#), entsprechend wird mit XML-Tags gearbeitete, deren Aufbau nach Möglichkeit simpel gehalten wurde.

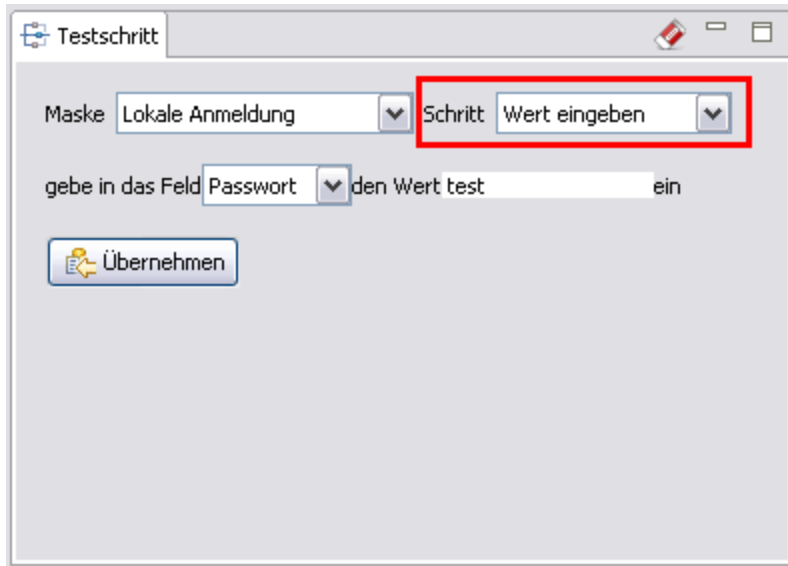
### TechnicalBindingTypeCollection.xml

```
<TechnicalBindingType id="Eingabe_Wert" name="Wert eingeben">
  <actionPart position="1" type="TEXT" value="gebe in das Feld" />
  <actionPart position="2" type="ACTION_NAME"/>
  <actionPart position="3" type="TEXT" value="den Wert" />
  <actionPart position="4" type="ARGUMENT"/>
  <actionPart position="5" type="TEXT" value="ein" />
</TechnicalBindingType>
```

Innerhalb des `TechnicalBindingType`-Tags werden ID und Name vergeben. Der Name erscheint im *Test-Editor* und sollte dem Anwender beschreiben was für einen Testschritt er hier auswählt. Die ID muss der ID in der `AllActionGroups.xml` entsprechen, da über diese ID die `AllActionGroups.xml` auf die Einträge in der `TechnicalBindingTypeCollection.xml` verweist.

In den Action-Part-Tags werden Position, Typ und Wert angegeben:

- Die **position** zeigt die Position im Satz der [Fachsprache](#) im *Test-Editor* an
- Der **type** kann folgende Werte repräsentieren:
  - `TEXT` wird als einfacher Text dargestellt und dient zur Beschreibung des Testschritts
  - `ACTION_NAME` beschreibt welches Element adressiert werden soll (z.B. ein konkretes Eingabefeld)
  - `ARGUMENT` erscheint dem Benutzer als Eingabefeld ohne Typbeschränkung
- Der **value** stellt den statischen Text eines Testschrittes dar



## Aufbau AllActionGroups.xml

Auch bei der AllActionGroups.xml ist eine XML-Bibliothek.

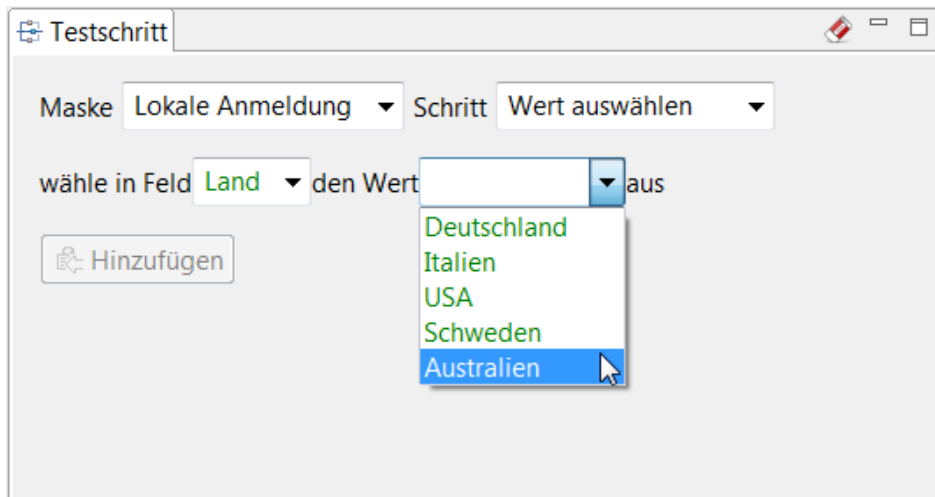
### AllActionGroups.xml

```
<action technicalBindingType="Auswahl_Wert">
  <actionName locator="land">Land</actionName>
  <argument id="argument1">
    <value>Deutschland</value>
    <value>Italien</value>
    <value>USA</value>
    <value>Schweden</value>
    <value>Australien</value>
  </argument>
</action>
```

Innerhalb des Action-Tag wird der TechnicalBindingType angegeben, der der ID aus der TechnicalBindingTypeCollection.xml entspricht.

Der ActionName-Tag stellt über den Locator die Verbindung zur ElementList.conf her. Der hier angegebene Wert ist die Überschrift, unter deren Namen der Benutzer die Werte des Argument-Tags auswählen kann. Somit fungieren die Value-Tags innerhalb des Argument-Tags als Vorbelegungsliste für den *Test-Editor*.

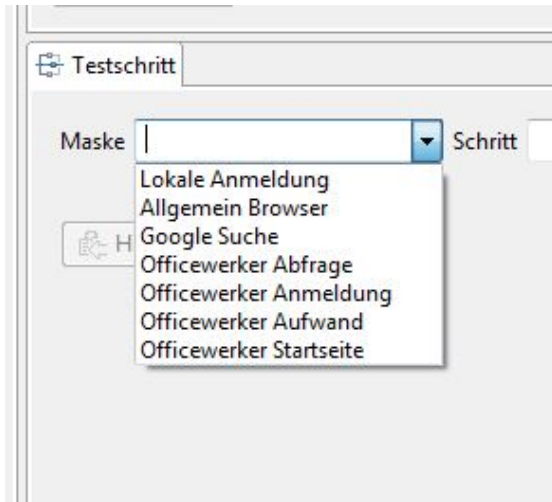
Einem TechnicalBindingType können mehrere Locator hinzugefügt werden, so dass aus einer Combo-Box auch mehrere Felder auswählbar sind.



Standardmäßig werden die Einträge in den Combo-Boxen **Maske** und **Schritt** alphabetisch sortiert. Diese Sortierung kann man durch hinzufügen des Tags **"sort=<Numerischer Wert>"** beeinflussen.

TechnicalBindingType für "Starte Browser"

```
<ActionGroup name="Allgemein Browser" sort="30">
    ...
</ActionGroup>
<ActionGroup name="Lokale Anmeldung" sort="10">
    ...
</ActionGroup>
```



## Aufbau ElementList.conf

Die ElementList.conf ist eine Textdatei, in der Key-Value Paare abgelegt werden, die über den *Test-Editor* eingelesen und verbinden die Verarbeitung im *Test-Treiber* mit den Feldern der AUT. Somit müssen Eingabeelemente wie Buttons, Eingabefelder, Combo-Boxen usw. in der ElementList.conf eingepflegt werden.

ElementList.conf

```
# Lokale Demo Login Seite
headline = hlTextField
user = userTextField
password = psTextField
land = landTextField
```

Der Locator aus AllActionGroups.xml wird hier als landTextField hinterlegt und System intern entsprechend interpretiert.

## Bibliothekeneinträge automatisch erstellen

### UI-Scanner

#### Kurzerklärung:

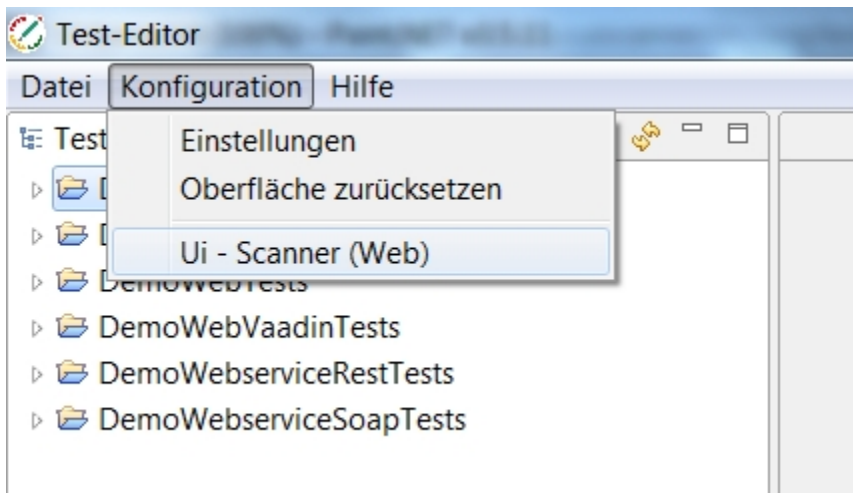
Der Ui - Scanner erleichtert das Erstellen der [AllActionGroup.xml](#) und der [ElementList](#). Der Scanner wurde dazu konzipiert ein einfaches Auslesen der IDs einer Website zu ermöglichen. Zudem können die ausgelesenen IDs bearbeitet, ausgewählt und anschließend in die gewünschte Form ([AllActionGroup.xml](#), [ElementList](#)) übertragen werden.

### Installation des Ui-Scanners im TestEditor

Ist noch zu klären.

### Starten des UI-Scanners und Bedienelemente

Den Scanner finden sie im TestEditor unter: Konfiguration -> Ui – Scanner (Web).



Der gestartete Scanner sieht wie folgt aus.



## Websites Scannen

Um eine Website zu scannen startet man den Browser indem man die gewünschte URL in das URL Textfeld eingibt, den Browser auswählt und anschließend auf Browser starten drückt. Der nun geöffnete Browser kann minimiert werden. Abschließend wählt man im Feld für die Filter die gewünschten Elemente aus und drückt anschließend auf "Website scannen".

Der Scanvorgang kann, je nach Größe der Webseite und Auswahl der Einstellungen, unterschiedlich lange dauern. Sobald die Webseite vollständig gescannt wurde, werden die gefunden Elemente in der Tabelle angezeigt.

Webelements	AllActionGroup	Elementlist				
	Typ	Name	Locator	Technische ID	Value	
1	<input type="checkbox"/>	button	login_ID	login_ID		
2	<input checked="" type="checkbox"/>	button	reset_ID	reset_ID		
3	<input type="checkbox"/>	checkbox	salami_ID	salami_ID		
4	<input checked="" type="checkbox"/>	checkbox	pilze_ID	pilze_ID		
5	<input checked="" type="checkbox"/>	radio	mastercard_ID	mastercard_ID		
6	<input type="checkbox"/>	radio	visa_ID	visa_ID		
7	<input type="checkbox"/>	radio	americ...ess_ID	american_express_ID		
8	<input checked="" type="checkbox"/>	input	user	user		
9	<input type="checkbox"/>	input	password	password		
10	<input checked="" type="checkbox"/>	select	land	land	Deutschland, Italien,... Schweden, Australien	

Welche der gewünschten Elemente in die AllActionGroup und ElementListe übernommen werden sollen kann man mithilfe der Checkboxes auf der linken Seite auswählen. Über die Reiter der Tabelle sind die jeweiligen Listen auswählbar.

Webelements	AllActionGroup	Elementlist
<pre>&lt;?xml version="1.0" encoding="UTF-8" standalone="yes"?&gt; &lt;ActionGroups&gt;   &lt;ActionGroup name="UiScanner"&gt;     &lt;action technicalBindingType="Button_Druecken"&gt;       &lt;actionName locator="reset_ID"&gt;reset_ID&lt;/actionName&gt;     &lt;/action&gt;     &lt;action technicalBindingType="Eingabe_Wert"&gt;       &lt;actionName locator="pilze_ID"&gt;pilze_ID&lt;/actionName&gt;     &lt;/action&gt;     &lt;action technicalBindingType="Eingabe_Wert"&gt;       &lt;actionName locator="mastercard_ID"&gt;mastercard_ID&lt;/actionName&gt;     &lt;/action&gt;     &lt;action technicalBindingType="Leere_Wert"&gt;       &lt;actionName locator="user"&gt;user&lt;/actionName&gt;     &lt;/action&gt;     &lt;action technicalBindingType="Eingabe_Wert"&gt;       &lt;actionName locator="user"&gt;user&lt;/actionName&gt;     &lt;/action&gt;     &lt;action technicalBindingType="Auswahl_Wert"&gt;       &lt;actionName locator="land"&gt;land&lt;/actionName&gt;       &lt;argument id="argument_land"&gt; </pre>		

Webelements	AllActionGroup	Elementlist
<div>#UiScanner reset_ID=reset_ID pilze_ID=pilze_ID mastercard_ID=mastercard_ID user=user land=land</div>		

## Ui - Scanner Konfigurieren

### Konfiguration des Scanners

Als Standard liegt bereits eine Datei zur Konfiguration des UI-Scanners bei. Möchte man das Verhalten des Scanners während des Scann-Vorgangs beeinflussen kann man diese Textdatei entsprechend anpassen.

#### Typ Block Beispiel

```
input= ((tagname = textarea) | (  
    (tagname = input) & !(  
        (type = radio) |  
        (type % submit) |  
        (type = reset) |  
        (type = checkbox)  
    )  
));
```

Der erste Teil beschreibt zu welchem Typ diese Aussage gehört. In diesem Beispiel ist es der Typ "input" (weitere typen: button, checkbox, radio, select). Jeder Ausdruck beginnt mit einer „(" und endet mit einer „)“.

#### Basis Ausdruck:

Der Basis Ausdruck setzt sich wie folgt zusammen (Attribut = value). Attribut bezeichnet das Attribut des Webelements (tagname, type, class, etc.). Value steht hierbei für den Wert auf den geprüft wird. Im Basis Ausdruck kann man die Operationen gleich „=“ oder contains „%“ verwenden.

#### Weitere Ausdrücke:

Die bereits erwähnten Basis Ausdrücke können über weitere Ausdrücke verbunden werden. Diese sind und "&", oder "|" und not "!". Die "&" und "|" Operanden stehen in den Klammern. Einzig der "!" Ausdruck wird vor die Klammer geschrieben wie im Beispiel.

Geschlossen wird der Typ-Block mit einem Semikolon ";".



```
input=    (
            (tagname = textarea) |      (
                (tagname = input) &      ! (
                    (type = radio) | (type = submit) | (type = reset) |      (type =
checkbox)
                )
            )
        );

button=    (
            (tagname = button) | (
                (tagname = input) & (
                    (type = submit) | (type = reset)
                )
            ) | (
                (tagname = div) & (class % button)
            )
        );

radio=    (
            (tagname = radio) | (
                (tagname = input) & (type = radio)
            )
        );

select=    (tagname = select);
checkbox=    (
            (tagname = checkbox) | (
                (tagname = input) & (type = checkbox)
            )
        );
```

## Bestehende Test-Treiber verwenden

Um die [Test](#) automatisiert ablaufen lassen zu können, müssen [Test-Treiber](#) als Schnittstellen zu den jeweiligen Systemen aufgebaut werden. Dabei bedient jeder Treiber ein anderes System. So ist Selenium z. B. für Web-Anwendungen konzipiert und [FEST](#) für Swing-Anwendungen.

Um den Einstieg in die verschiedenen Technologien zu vereinfachen werden mit dem *Test-Editor* eine Reihe von Demos ausgeliefert. Entsprechend gibt es zu jedem Test-Treiber der eine oder mehrere Demos bedient eine [Fixture](#). Diese Fixtures können als Basis für andere Projekte dienen und entsprechend ausgebaut werden.

Zur Übersichtlichkeit werden im Folgenden die bereits vorhandenen Methoden der verschiedenen Fixtures beschrieben.

Die mitgelieferten Fixtures befinden sich im Installationsverzeichnis des Test-Editors.

- ./plugins/org.testeditor.fixture.lib
  - fixture (hier befinden sich alle Fixturetypen)
  - lib (hier befinden sich alle abhängigen jars)

Die Einbindung eines Fixtures in das Projekt erfolgt durch Erweiterung des Klassenpfades auf der Einstellungsseite des Projektes, direkt auf dem FitNesse-Server.

### Einstellungsseite FitNesse

```
!path ${FIXTURE_LIB_BUNDLE_PATH}/fixture/TestEditorFixtureWeb-*.jar
!path ${FIXTURE_LIB_BUNDLE_PATH}/lib/*.jar
```

Die Umgebungsvariable `FIXTURE_LIB_BUNDLE_PATH` wird direkt im Test-Editor gesetzt

## SwingFixture

Dieser Test-Treiber kann universell zur **Fernsteuerung von Swing Services** verwendet werden. Die Implementierung agiert dabei als Swing-App. Im Folgenden sind die einzelnen Services beschrieben, die entsprechend in den jeweiligen Projekten genutzt werden können.

### Demo-Projekte (siehe Benutzer-Handbuch)

Für Swing Applikationen gibt es ein Demo-Projekt **DemoSwingTests**, welches im Benutzer-Handbuch beschrieben ist.

- Anwendung
  - Starte Anwendung
  - Beende Anwendung
- Eingabeelemente
  - Button Druecken
- Eingabe von Werten
  - Wert eingeben
  - Wert löschen
  - Text prüfen
  - Text ungültig prüfen
- Combobox
  - Wert auswählen
  - Wert prüfen
  - Wert ungültig prüfen
- Checkboxes
  - Checkbox aktivieren
  - Checkbox deaktivieren
- Radiobuttons
  - Radiobutton aktivieren
  - Radiobutton deaktivieren
- Tabellen
  - Tabellenzeile auswählen
  - Tabellenzeile doppelklicken
  - Tabelleneintrag prüfen

## Anwendung

### Starte Anwendung

Startet eine Swing Anwendung.

Service-Name	Starte_Anwendung
Fachsprache	starte Anwendung <a href="#">path.zur.main</a>
Verwendung (Beispiel)	<pre>&lt;action technicalBindingType="Starte_Anwendung" /&gt;</pre>
Fixture Methode	<b>public void</b> startApplication(final String path)

### Beende Anwendung

Beendet die laufende Swing Anwendung.

Technical Binding	Beende_Anwendung
Fachsprache	beende Anwendung

Verwendung (Beispiel)	<pre>&lt;action technicalBindingType="Beende_Anwendung" /&gt;</pre>
Fixture Methode	<b>public void</b> stopApplication()

## Eingabeelemente

### Button Druecken

Klickt auf einen Button.

Service-Name	Button_Druecken
Fachsprache	starte Anwendung <b>mein.path.zur.main</b>
Verwendung (Beispiel)	<pre>&lt;action technicalBindingType="Button_Druecken"&gt;   &lt;actionName locator="anyButtonLocator"&gt;meinButtonName&lt;/actionName&gt; &lt;/action&gt;</pre>
Fixture Methode	<b>public void</b> clickButton(String elementListKey)

## Eingabe von Werten

### Wert eingeben

Einen beliebigen Wert in ein Feld eingeben.

Technical Binding	Wert_eingeben
Fachsprache	gebe in das Feld <b>meinEingabeFeld</b> den Wert <b>meinTestWert</b> ein
Verwendung (Beispiel)	<pre>&lt;action technicalBindingType="Wert_eingeben"&gt;   &lt;actionName locator="anyFieldLocator"&gt;meinFeldName&lt;/actionName&gt; &lt;/action&gt;</pre>
Fixture Methode	<b>public void</b> insertIntoTextField(String elementListKey, String text)

### Wert löschen

Löscht einen Wert aus einem Feld.

Technical Binding	Wert_löschen
Fachsprache	lösche aus dem Feld <b>meinFeld</b> den Wert <b>meinTestWert</b>
Verwendung (Beispiel)	<pre>&lt;action technicalBindingType="Wert_löschen"&gt;   &lt;actionName locator="anyFieldLocator"&gt;meinFeldName&lt;/actionName&gt; &lt;/action&gt;</pre>

<b>Fixture Methode</b>	<b>public void</b> deleteTextField(String elementListKey)
------------------------	---

## Text prüfen

Prüft ob der Text eines Feldes mit dem Vergleichstext übereinstimmt.

<b>Technical Binding</b>	<b>Text_prüfen</b>
<b>Fachsprache</b>	prüfe ob im Feld <b>meinFeld</b> der Text <b>meinTestText</b> eingetragen ist
<b>Verwendung (Beispiel)</b>	<pre>&lt;action technicalBindingType="Text_prüfen"&gt;   &lt;actionName locator="anyTextfFieldLocator"&gt;meinTextFeld&lt;/actionName&gt; &lt;/action&gt;</pre>
<b>Fixture Methode</b>	<b>public boolean</b> checkIfTextEquals(String elementListKey, String text)

## Text ungültig prüfen

Prüft ob der Text eines Feldes mit dem Vergleichstext nicht übereinstimmt.

<b>Technical Binding</b>	<b>Text_ungültig_prüfen</b>
<b>Fachsprache</b>	prüfe ob im Feld <b>meinFeld</b> der Text <b>meinTestText</b> nicht eingetragen ist
<b>Verwendung (Beispiel)</b>	<pre>&lt;action technicalBindingType="Text_ungültig_prüfen"&gt;   &lt;actionName locator="anyTextfFieldLocator"&gt;meinTextFeld&lt;/actionName&gt; &lt;/action&gt;</pre>
<b>Fixture Methode</b>	<b>public boolean</b> checkIfTextNotEquals(String elementListKey, String text)

## Combobox

### Wert auswählen

Wählt einen Wert aus der Combobox aus.

<b>Technical Binding</b>	<b>Wert_auswählen</b>
<b>Fachsprache</b>	wähle in Feld <b>meineCombobox</b> den Wert <b>meinTestText</b> aus

<b>Verwendung (Beispiel)</b>	<pre> &lt;action technicalBindingType="Wert_auswählen"&gt;   &lt;actionName locator="anyComboboxLocator"&gt;meineCombobox&lt;/actionName&gt;   &lt;argument id="argument1"&gt;     &lt;value&gt;exampleValue 1&lt;/value&gt;     &lt;value&gt;exampleValue 2&lt;/value&gt;     &lt;value&gt;...&lt;/value&gt;   &lt;/argument&gt; &lt;/action&gt; </pre>
<b>Fixture Methode</b>	<b>public void</b> selectComboBoxItemByName(String elementListKey, String item)

## Wert prüfen

Prüft ob in einer Combobox ein bestimmter Wert ausgewählt worden ist.

<b>Technical Binding</b>	<b>Wert_prüfen</b>
<b>Fachsprache</b>	prüfe ob im Feld <b>meineCombobox</b> der Wert <b>meinTestText</b> ausgewählt ist
<b>Verwendung (Beispiel)</b>	<pre> &lt;action technicalBindingType="Wert_prüfen"&gt;   &lt;actionName locator="anyComboboxLocator"&gt;meineCombobox&lt;/actionName&gt;   &lt;argument id="argument1"&gt;     &lt;value&gt;exampleValue 1&lt;/value&gt;     &lt;value&gt;exampleValue 2&lt;/value&gt;     &lt;value&gt;...&lt;/value&gt;   &lt;/argument&gt; &lt;/action&gt; </pre>
<b>Fixture Methode</b>	<b>public boolean</b> checkIfSelectedItems(String elementListKey, String text)

## Wert ungültig prüfen

Prüft ob in einer Combobox ein bestimmter Wert nicht ausgewählt worden ist.

<b>Technical Binding</b>	<b>Wert_ungültig_prüfen</b>
<b>Fachsprache</b>	prüfe ob im Feld <b>meineCombobox</b> der Wert <b>meinTestText</b> nicht ausgewählt ist
<b>Verwendung (Beispiel)</b>	<pre> &lt;action technicalBindingType="Wert_ungültig_prüfen"&gt;   &lt;actionName locator="anyComboboxLocator"&gt;meineCombobox&lt;/actionName&gt;   &lt;argument id="argument1"&gt;     &lt;value&gt;exampleValue 1&lt;/value&gt;     &lt;value&gt;exampleValue 2&lt;/value&gt;     &lt;value&gt;...&lt;/value&gt;   &lt;/argument&gt; &lt;/action&gt; </pre>
<b>Fixture Methode</b>	<b>public boolean</b> checkIfSelectedItemsNot(String elementListKey, String text)

## Checkboxes

### Checkbox aktivieren

Aktiviert eine Checkbox

Technical Binding	Checkbox_aktivieren
Fachsprache	aktiviere Checkbox <b>meineCheckbox</b>
Verwendung (Beispiel)	<pre>&lt;action technicalBindingType="Checkbox_aktivieren"&gt;   &lt;actionName locator="anyCheckboxLocator"&gt;meineCheckbox&lt;/actionName&gt; &lt;/action&gt;</pre>
Fixture Methode	<b>public void</b> checkCheckBox(String elementListKey)

### Checkbox deaktivieren

Deaktiviert eine Checkbox

Technical Binding	Checkbox_deaktivieren
Fachsprache	deaktiviere Checkbox <b>meineCheckbox</b>
Verwendung (Beispiel)	<pre>&lt;action technicalBindingType="Checkbox_deaktivieren"&gt;   &lt;actionName locator="anyCheckboxLocator"&gt;meineCheckbox&lt;/actionName&gt; &lt;/action&gt;</pre>
Fixture Methode	<b>public void</b> uncheckCheckBox(String elementListKey)

## Radiobuttons

### Radiobutton aktivieren

Aktiviert einen Radiobutton

Technical Binding	RadioButton_aktivieren
Fachsprache	aktiviere Radio-Button <b>meinRadioButton</b>
Verwendung (Beispiel)	<pre>&lt;action technicalBindingType="RadioButton_aktivieren"&gt;   &lt;actionName locator="anyRadioButtonLocator"&gt;meinRadioButton&lt;/actionName&gt; &lt;/action&gt;</pre>
Fixture Methode	<b>public void</b> checkRadioButton(String elementListKey)

## Radiobutton deaktivieren

Deaktiviert einen Radiobutton

Technical Binding	RadioButton_deaktivieren
Fachsprache	deaktiviere Radio-Button <b>meinRadioButton</b>
Verwendung (Beispiel)	<pre>&lt;action technicalBindingType="RadioButton_deaktivieren"&gt;   &lt;actionName locator="anyRadioButtonLocator"&gt;meinRadioButton&lt;/actionName&gt; &lt;/action&gt;</pre>
Fixture Methode	<b>public void</b> uncheckRadioButton(String elementListKey)

## Tabellen

### Tabellenzeile auswählen

Wählt einen Zeile einer Tabelle aus.

Technical Binding	Tabellenzeile_auswählen
Fachsprache	in der Tabelle <b>meineTabelle</b> wähle die Zeile <b>meineZeilenId</b> aus
Verwendung (Beispiel)	<pre>&lt;action technicalBindingType="Tabellenzeile_auswählen" &gt;   &lt;actionName locator="anyTableLocator"&gt;meineTabelle&lt;/actionName&gt; &lt;/action&gt;</pre>
Fixture Methode	<b>public void</b> selectTableRowById(String elementListKey, int Id)

### Tabellenzeile doppelklicken

Führt einen Doppelklick auf eine Tabellenzeile aus

Technical Binding	Tabellenzeile_doppelklicken
Fachsprache	in der Tabelle <b>meineTabelle</b> doppelklicke die Zeile <b>meineZeilenId</b>
Verwendung (Beispiel)	<pre>&lt;action technicalBindingType="Tabellenzeile_doppelklicken" &gt;   &lt;actionName locator="anyTableLocator"&gt;meineTabelle&lt;/actionName&gt; &lt;/action&gt;</pre>
Fixture Methode	<b>public void</b> doubleClickTableRowById(String elementListKey, int Id)

### Tabelleneintrag prüfen

Prüft ob in der letzten Zeile einer Tabelle in einer Spalte ein bestimmter Wert eingetragen ist.

Technical Binding	Tabelleneintrag_prüfen
-------------------	------------------------



<b>Fachsprache</b>	prüfe ob in der Tabelle <b>meineTabelle</b> in der letzten Zeile in Spalte <b>meineSpaltenId</b> der Wert <b>meinTestWert</b> eingetragen ist
<b>Verwendung (Beispiel)</b>	<pre> &lt;action technicalBindingType="Tabelleneintrag_prüfen" &gt;   &lt;actionName locator="anyTableLocator"&gt;meineTabelle&lt;/actionName&gt;   &lt;argument id="argument1"&gt;     &lt;value&gt;exampleValue 1&lt;/value&gt;     &lt;value&gt;exampleValue 2&lt;/value&gt;   &lt;/argument&gt; &lt;/action&gt; </pre>
<b>Fixture Methode</b>	<b>public void</b> selectTableRowById(String elementListKey, int Id)

## WebFixture

Mit diesem Test-Treiber werden Web-Anwendungen ferngesteuert, die im Wesentlichen aus reinem HTML bestehen.

- Anwendung
  - [Starte Anwendung](#)
  - [Beende Anwendung](#)
  - [Navigiere zur URL](#)
- Eingabebelemente
  - [Element drücken](#)
- Eingabe von Werten
  - [Wert eingeben](#)
  - [Wert löschen](#)
  - [Text prüfen](#)
  - [Text ungültig prüfen](#)
- Warten
  - [Warte auf ein Element](#)
  - [Warte eine gewisse Zeit](#)

Der *Test-Editor* verwendet für die Steuerung von Web-Anwendungen die Version 2.35.0 von SeleniumHQ (WebDriver).

Browser	Offiziell unterstützte Versionen	weitere lauffähige Versionen	Anmerkungen
Firefox	3.6 bis 23 (unter Windows, Mac, Linux)		<ul style="list-style-type: none"><li>• Bei der Ausführung der Tests darf keine weitere Instanz des Firefox geöffnet sein.</li><li>• Der Pfad zum Browser muss gesetzt sein (path.browser).</li><li>• Der Pfad zum Browser sollte keine Leerzeichen enthalten.</li></ul>
Google Chrome	Alle Versionen		<ul style="list-style-type: none"><li>• Als Browser-Pfad muss der Pfad zum Chrome-WebDriver angegeben werden (path.browser).</li></ul>
Internet Explorer	6, 7, 8 unter XP und 9 unter Windows 7	11 unter Windows 7 mit dem 32bit Treiber	<ul style="list-style-type: none"><li>• Die Sicherheitseinstellungen des Browsers müssen ggf. angepasst werden.</li><li>• Keine XPath-Unterstützung ab Version 10 des Internet Explorer in der Element-Liste.</li></ul>

## Anwendung

### Starte Anwendung

Öffnet eine konkrete Browser-Instanz.

Service-Name	Starte_Browser
Fachsprache	starte Browser <b>meinBrowser</b>
Verwendung (Beispiel)	<pre>&lt;action technicalBindingType="Starte_Browser"&gt;   &lt;actionName locator="Firefox"&gt;Firefox&lt;/actionName&gt; &lt;/action&gt; &lt;action technicalBindingType="Starte_Browser"&gt;   &lt;actionName locator="IE"&gt;IE&lt;/actionName&gt; &lt;/action&gt; &lt;action technicalBindingType="Starte_Browser"&gt;   &lt;actionName locator="Chrome"&gt;Chrome&lt;/actionName&gt; &lt;/action&gt;</pre>
Fixture Methode	<b>public boolean</b> openBrowser(String browserName, String browserPath)

### Beende Anwendung

Schließt eine konkrete Browser-Instanz.

Service-Name	Beende_Browser
Fachsprache	beende Browser
Verwendung (Beispiel)	<pre>&lt;action technicalBindingType="Beende_Browser" /&gt;</pre>
Fixture Methode	<b>public boolean</b> closeBrowser()

## Navigiere zur URL

Die angegebene URL wird in der aktuellen Browser-Instanz geöffnet.

Service-Name	Navigiere_auf_Seite
Fachsprache	navigiere auf die Seite <b>url.der.anwendung</b>
Verwendung (Beispiel)	<pre>&lt;action technicalBindingType="Navigiere_auf_Seite" /&gt;</pre>
Fixture Methode	<b>public boolean</b> navigateToUrl(String url)

## Eingabeelemente

### Element drücken

Klickt auf ein Element in der GUI. In der Regel handelt es sich dabei um einen Button, eine Checkbox oder einen Radiobutton.

Service-Name	Button_Druecken
Fachsprache	klicke auf <b>element</b>
Verwendung (Beispiel)	<pre>&lt;action technicalBindingType="Button_Druecken"&gt;   &lt;actionName locator="anyLocatorElement"&gt;meinElement&lt;/actionName&gt; &lt;/action&gt;</pre>
Fixture Methode	<b>public boolean</b> click(String elementListKey)

## Eingabe von Werten

### Wert eingeben

Tippt einen Wert in das Eingabefeld ein. In der Regel handelt es sich um ein Input oder eine Textarea.

Service-Name	Eingabe_Wert
Fachsprache	gebe in das Feld <b>element</b> den Wert <b>meinWert</b> ein

<b>Verwendung (Beispiel)</b>	<pre>&lt;action technicalBindingType="Eingabe_Wert"&gt;   &lt;actionName locator="anyLocatorElement"&gt;meinElement&lt;/actionName&gt; &lt;/action&gt;</pre>
<b>Fixture Methode</b>	<b>public boolean</b> insertIntoField(String value, String elementListKey)

## Wert löschen

Löscht die aktuelle Eingabe eines Eingabefeldes.

<b>Service-Name</b>	<b>Leere_Wert</b>
<b>Fachsprache</b>	leere das Feld <b>meinElement</b>
<b>Verwendung (Beispiel)</b>	<pre>&lt;action technicalBindingType="Leere_Wert"&gt;   &lt;actionName locator="anyLocatorElement"&gt;meinElement&lt;/actionName&gt; &lt;/action&gt;</pre>
<b>Fixture Methode</b>	<b>public boolean</b> clear(String elementListKey)

## Text prüfen

Prüft ob ein Text in dem Source-Code der zuletzt geladenen Seite vorhanden ist.

<b>Service-Name</b>	<b>Pruefe_Wert_vorhanden</b>
<b>Fachsprache</b>	überprüfe ob der Text <b>meinText</b> vorhanden ist
<b>Verwendung (Beispiel)</b>	<pre>&lt;action technicalBindingType="Pruefe_Wert_vorhanden" /&gt;</pre>
<b>Fixture Methode</b>	<b>public boolean</b> textIsVisible(String text)

## Text ungültig prüfen

Prüft ob ein Text in dem Source-Code der zuletzt geladenen Seite nicht vorkommt.

<b>Service-Name</b>	<b>Pruefe_Wert_nicht_vorhanden</b>
<b>Fachsprache</b>	überprüfe ob nicht der Text <b>meinText</b> vorhanden ist
<b>Verwendung (Beispiel)</b>	<pre>&lt;action technicalBindingType="Pruefe_Wert_nicht_vorhanden" /&gt;</pre>
<b>Fixture Methode</b>	<b>public boolean</b> textIsVisible(String text) - als Verneinung in der SzenarioLibrary

## Warten

## Warte auf ein Element

Wartet auf ein Element in der Anwendung und gibt TRUE zurück, wenn das Element innerhalb des Timeouts sichtbar wird.

Service-Name	Warte_Element
Fachsprache	warte auf Element <b>meinElement</b>
Verwendung (Beispiel)	<pre>&lt;action technicalBindingType="Warte_Element"&gt;   &lt;actionName locator="anyLocatorElement"&gt;meinElement&lt;/actionName&gt; &lt;/action&gt;</pre>
Fixture Methode	<b>public boolean</b> waitForElement(String elementListKey)

## Warte eine gewisse Zeit

Wartet die angegebene Zeit, indem der Thread schlafen gelegt wird.

Service-Name	Warte_Sekunden
Fachsprache	warte <b>Wert</b> Sekunden
Verwendung (Beispiel)	<pre>&lt;action technicalBindingType="Warte_Sekunden" /&gt;</pre>
Fixture Methode	<b>public boolean</b> waitSeconds(long timeToWait)

## WebserviceTestFixture

Dieser Test-Treiber kann universell zur **Fernsteuerung von REST Webservices** verwendet werden. Die Implementierung agiert dabei als REST-Client. Im Folgenden sind die einzelnen Services beschrieben, die entsprechend in den jeweiligen Projekten genutzt werden können.

### Demo-Projekte (siehe Benutzer-Handbuch)

Für REST Webservices gibt es ein Demo-Projekt **DemoWebserviceRestTests**, welches im Benutzer-Handbuch beschrieben ist.

- Anwendung
  - Aufruf Webservice Get
- Überprüfung Response Anzahl
- Überprüfung Response

## Anwendung

### Aufruf Webservice Get

Sendet ein GET an die Übergebene URL und schreibt die Antwort in ein Log.

Service-Name	Aufruf_Webservice_Get
Fachsprache	rufe den Webservice <b>beispielWebservice</b> auf
Verwendung (Beispiel)	<pre>&lt;action technicalBindingType="Aufruf_Webservice_Get" /&gt;</pre>
Fixture Methode	<b>public boolean</b> sendGet(String url)

## Überprüfung Response Anzahl

Überprüft die Anzahl der untergeordneten Elemente einer gegebenen X-Path-Position.

Technical Binding	Überprüfung_Response_Anzahl
Fachsprache	überprüfe ob der Response <b>meinResponseType</b> insgesamt <b>anzahlVonResponseElementen</b> Mal vorhanden ist
Verwendung (Beispiel)	<pre>&lt;action technicalBindingType="Überprüfung_Response_Anzahl"&gt;   &lt;actionName locator="meinXpath"&gt;Counter&lt;/actionName&gt; &lt;/action&gt;</pre>
Fixture Methode	<b>public boolean</b> checkChildrenCount(String xpath, String count)

## Überprüfung Response

Überprüft einzelne Elemente der REST Response Nachricht anhand der X-Path-Position.

Technical Binding	Überprüfung_Response
Fachsprache	überprüfe ob der Response <b>meinResponseElement</b> den Wert <b>abc</b> entspricht

Verwendung (Beispiel)	<pre>&lt;action technicalBindingType="Überprüfung_Response"&gt;   &lt;actionName locator="/meinXpath"&gt;abc&lt;/actionName&gt; &lt;/action&gt;</pre>
Fixture Methode	<b>public boolean</b> checkXmlResponse(String xpath, String response)

## WebserviceSoapFixture

Dieser Test-Treiber kann universell zur **Fernsteuerung von SOAP Webservices** verwendet werden. Die Implementierung agiert dabei als SOAP-Client. Im Folgenden sind die einzelnen Services beschrieben, die entsprechend in den jeweiligen Projekten genutzt werden können.

### Demo-Projekte (siehe Benutzer-Handbuch)

Für SOAP Webservices gibt es ein Demo-Projekt **DemoWebserviceSoapTests**, welches im Benutzer-Handbuch beschrieben ist.

## Anwendung

### Starte Anwendung

Viele Webservice verwenden Namespaces um einzelne Elemente (z.B. im Request oder Response) zu gruppieren. Mit dieser Hilfsmethode kann der Namespace festgelegt und dann im Request/Response verwendet werden.

Service-Name	Festlegung_Namespace
Fachsprache	setze den Prefix <b>beispielPrefix</b> für den Namespace <b>http://www.einBeliebigerNamespace.de</b>
Verwendung (Beispiel)	<pre>&lt;action technicalBindingType="Festlegung_Namespace"&gt;   &lt;actionName locator="anyLocatorNamespace"&gt;beispielPrefix&lt;/actionName&gt;   &lt;argument id="argument1"&gt;     &lt;value&gt;http://www.einBeliebigerNamespace.de&lt;/value&gt;   &lt;/argument&gt; &lt;/action&gt;</pre>
Fixture Methode	<b>public boolean</b> addPrefixNamespace(String prefix, String namespace)

### Eingabe Request

Setzt universell einen Request Parameter in einer SOAP Request Nachricht. Wurde zuvor der Service zur Festlegung des Namespaces genutzt, kann der jeweilige Namespace für den Aufbau des Elements verwendet werden.

Technical Binding	Eingabe_Request
Fachsprache	gebe als Request <b>meinRequestElement</b> den Wert <b>xyz</b> ein
Verwendung (Beispiel)	<pre>&lt;action technicalBindingType="Eingabe_Request"&gt;   &lt;actionName locator="anyLocatorRequest"&gt;meinRequestElement&lt;/actionName&gt; &lt;/action&gt;</pre>
Fixture Methode	<b>public boolean</b> setXPathValue(String path, String data)

### Aufruf Webservice

Ruft den eigentlichen Webservice auf. An dieser Stelle wird die Request-Nachricht aus den zuvor gesetzt Request-Elementen zusammengesetzt und an den Server (Webservice) geschickt. Der Response wird lokal zwischengespeichert und kann anschließend überprüft werden. Die eigentliche Request und Response Nachrichten (XML) werden im übrigen ins Log geschrieben, was hilfreich bei der Fehlersuche ist.

Technical Binding	Aufruf_Webservice
-------------------	-------------------



<b>Fachsprache</b>	rufe den Webservice <b>Beispiel-Service</b> auf
<b>Verwendung (Beispiel)</b>	<pre> &lt;action technicalBindingType="Aufruf_Webservice"&gt;   &lt;actionName locator="anyLocatorWebserviceUrl"&gt;Beispiel-Service&lt;/actionName&gt; &lt;/action&gt; </pre>
<b>Fixture Methode</b>	<b>public boolean</b> sendTo(String url)

## Überprüfung Response

Überprüft einzelne Elemente der SOAP Response Nachricht. Wurde zuvor der Service zur Festlegung des Namespaces genutzt, kann der jeweilige Namespace zum Auffinden der Elemente verwendet werden.

<b>Technical Binding</b>	<b>Überprüfung_Response</b>
<b>Fachsprache</b>	überprüfe ob der Response <b>meinResponseElement</b> den Wert <b>abc</b> entspricht
<b>Verwendung (Beispiel)</b>	<pre> &lt;action technicalBindingType="Überprüfung_Response"&gt;   &lt;actionName locator="anyLocatorResponse"&gt;meinResponseElement&lt;/actionName&gt; &lt;/action&gt; </pre>
<b>Fixture Methode</b>	<b>public boolean</b> checkResponseEqualsTo(String path, String value)

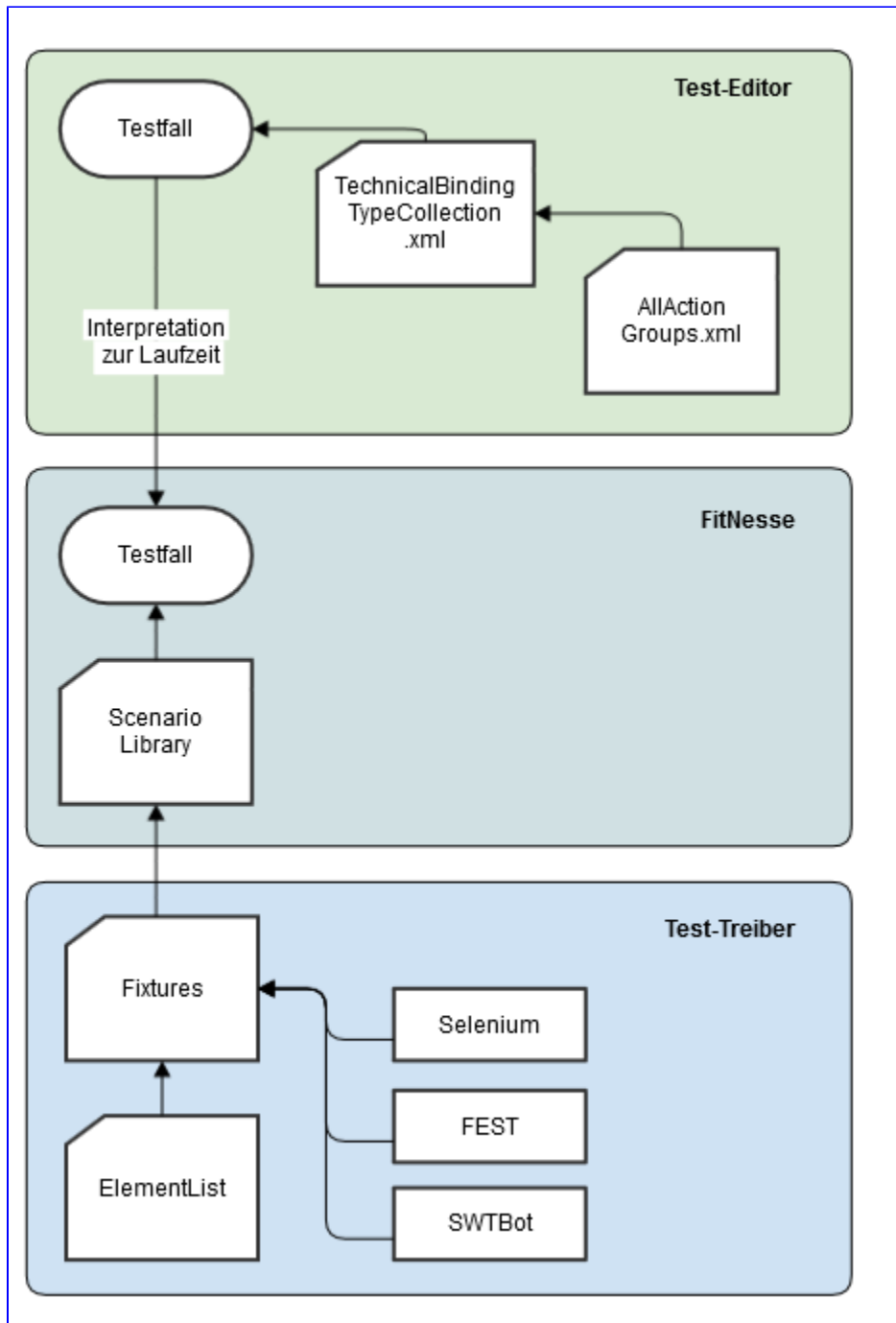
# Eigenen Test-Treiber entwickeln

Der *Test-Editor* baut auf das *FitNesse*-Framework auf. Daher baut die Struktur der im *Test-Editor* eingetragenen *Testfälle* auf *FitNesse* auf und werden von *FitNesse* entsprechend interpretiert.

Der *Test-Editor* benutzt die *TechnicalBindingTypeCollection.xml* und über diese die *AllActionGroups.xml* (*TechnicalBindingTypeCollection.xml* und *AllActionGroups.xml* anpassen), um die einzelnen *Testschritte* und Oberflächenelemente im Testfall anbieten zu können.

Wenn die Testfälle vollständig erfasst sind, löst *FitNesse* die Testfälle zur Laufzeit über die *ScenarioLibrary* auf. Dabei werden die Oberflächenelemente mit Hilfe der *ElementList.conf* identifiziert. Die *ScenarioLibrary* gibt die zu den Testschritten zugehörigen Java-Methoden der *Fixtures* an.

Die *Fixtures* sind spezifisch zu der jeweiligen Technologie des zu testenden Systems. So integriert beispielsweise das *Fixture* für Webanwendungen den *Selenium*-Treiber, um die Oberflächen zu steuern.



## Ein Beispiel

Als Beispiel wird im Folgenden die Anlage einer neuen Funktion anhand der DemoSwingTests beschrieben. DemoSwingTests ist eine der Demos, die mit dem *Test-Editor* ausgeliefert werden und kann über den Demo Wizard erstellt werden, so noch nicht geschehen.

### Schritt 1: Die Java-Funktion

In diesem Beispiel wird die Funktionalität zum Prüfen eines Tabelleneintrags hinzugefügt. Diese Funktionalität ist in der ausgelieferten Version bereits vorhanden, damit kann man das Beispiel auch direkt in den Dateien nachvollziehen.

In die Java Klasse **SwingFixture.java** fügen wir eine neue Funktion ein.

#### SwingFixture.java

```
...
import org.fest.swing.fixture.JTableFixture;
...

public boolean checkTableCellValue(String elementListKey, String value, String
column) {
    String locator = getLocatorFromElementList(elementListKey);
    int colLocator = Integer.parseInt(getLocatorFromElementList(column));
    String content = null;
    try {
        JTableFixture table = window.table(locator);
        BasicJTableCellReader cellReader = new BasicJTableCellReader();
        content = cellReader.valueAt(table.target, (table.rowCount() - 1), colLocator);
    } catch (Exception e) {
        LOGGER.error("could not select the Row from the tabel Error: " + e);
    }
    return (value.equals(content));
}
```

Da es sich um eine SwingDemo handelt, benutzt das Fixture **FEST** als Treiber. Es ist zu beachten, dass der verwendete Treiber mit den unterschiedlichen Demos variiert.

### Schritt 2: Die ScenarioLibrary

Die ScenarioLibrary\content.txt verbindet die vom Nutzer beschriebenen Eingaben auf der Oberfläche mit den Aufrufen der Java-Funktionen.

#### ScenarioLibrary\content.txt

```
...

'''Prüfe Tabelleneintrag'''
!|scenario|prüfe ob in der Tabelle|guidid|in der letzten Zeile in Spalte|columnno|der
Wert|pattern|eingetragen ist|
|checkTableCellValue;|@guidid|@pattern|@columnno|
```

Betrachtet man die unterste Zeile, sieht man den Namen der Funktion gefolgt von den notwendigen Übergabewerten. Die Übergabewerte werden dabei durch Pipe-Zeichen getrennt. Die mittlere Zeile ist die Verbindung zur TechnicalBindingTypesCollection.xml und dem Testfall. Die oberste Zeile ist eine interne Überschrift und unterstützt die Dokumentation.

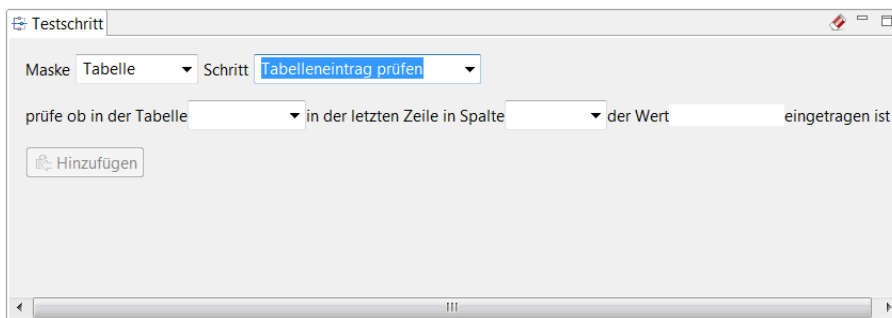
### Schritt 3: Die TechnicalBindingTypesCollection.xml

Die TechnicalBindingTypesCollection.xml legt die **Fachsprache**, also die für den Benutzer sichtbaren Schrittbeschreibungen fest. Hier muss beachtet werden, dass die Sätze der TechnicalBindingTypesCollection.xml den Sätzen der ScenarioLibrary entsprechen müssen. Wenn die Sätze nicht übereinstimmen wird der Testschritt nicht erkannt.

### TechnicalBindingTypesCollection.xml

```
...
<TechnicalBindingType id="Tabelleneintrag_prüfen" name="Tabelleneintrag prüfen">
  <actionPart position="1" type="TEXT" value="prüfe ob in der Tabelle" />
  <actionPart position="2" type="ACTION_NAME" />
  <actionPart position="3" type="TEXT" value="in der letzten Zeile in Spalte" />
  <actionPart position="4" type="ARGUMENT" id="argument1"/>
  <actionPart position="5" type="TEXT" value="der Wert" />
  <actionPart position="6" type="ARGUMENT" />
  <actionPart position="7" type="TEXT" value="eingetragen ist" />
</TechnicalBindingType>
...
```

Durch die Änderungen in der TechnicalBindingTypesCollection.xml wird die Fachsprache jetzt auch im Test-Editor angezeigt.



Um die Eingabe für den Benutzer zu vereinfachen und falsche Eingaben zu reduzieren, können die Felder auch vorgelegt werden. Dafür die Werte in der **Element-Liste** entsprechend angeben.

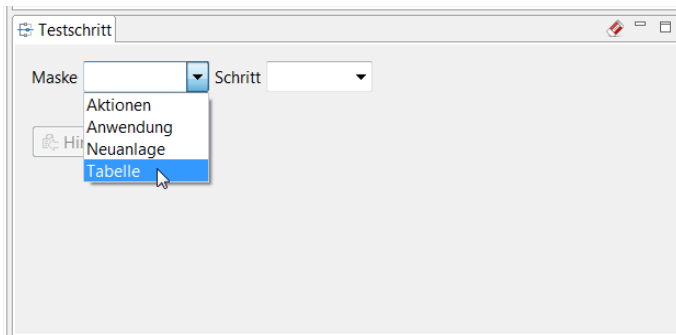
## Schritt 4: Die AllActionGroups.xml

Die AllActionGroups.xml ist ein Teil der Darstellung (GUI).

### AllActionGroups.xml

```
...
<ActionGroup name="Tabelle">
  ...
  <action technicalBindingType="Tabelleneintrag_prüfen" >
    <actionName locator="AlleAngestellten">Angestellte</actionName>
    <argument id="argument1">
      <value>Name</value>
      <value>Vorname</value>
    </argument>
  </action>
</ActionGroup>
...
```

Die ActionGroup ist eine Form von Gruppierung und dient der Übersichtlichkeit. Die ActionGroups erscheinen im *Test-Editor* unter "Maske".



Der Eintrag für das Beispiel in der AllActionGroups.xml gibt als technicalBindingType **"Tabelleneintrag\_prüfen"** an. Dieser Eintrag verbindet die AllActionGroups.xml mit der TechnicalBindingTypesCollection.xml.

## Schritt 5: Die ElementListe

Die ElementList.conf verbindet die Namen aus dem *Test-Editor*, die zum besseren Verständnis des Benutzers einfach gehalten werden sollten, mit den Werten, die im Fixture interpretiert werden. Die Daten aus der ElementList werden im Fixture eingelesen, um über den [Test-Treiber](#) auf das jeweilige Element in der Testanwendung zugreifen zu können. Ebenso ist es möglich, bestimmten Begriffen einen numerischen Wert zuzuweisen, wie es hier geschieht, um den Index der Tabellenspalten gleich geliefert zu bekommen.

Eine andere Möglichkeit wäre, im Fixture den Tabellenspaltennamen zu interpretieren und daraufhin den Index festzulegen.

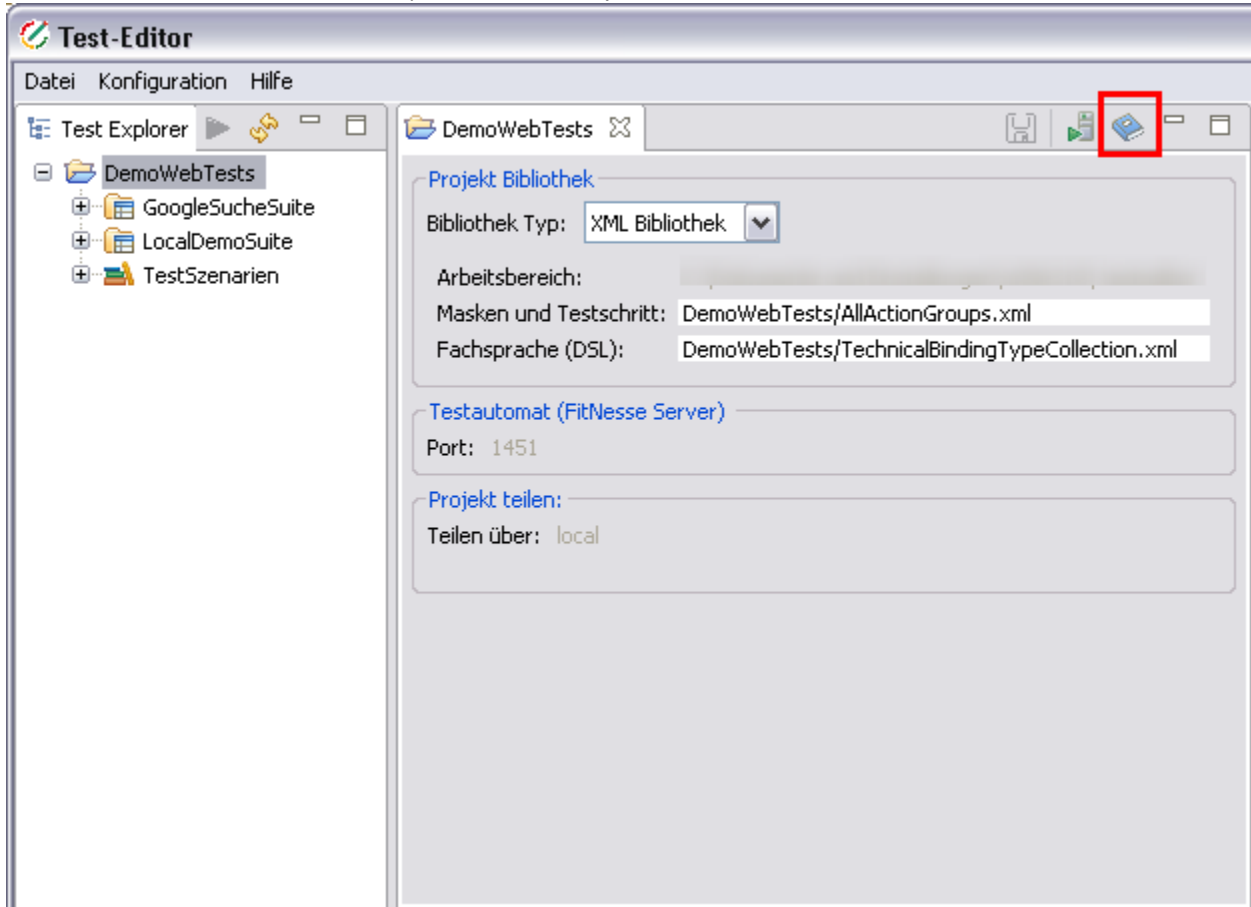
### ElementListe\content.txt

```
# erste Seite Eingabefelder
NameText = textFieldName
...
GeschlechtVerbergen = checkBoxGeschlecht
...

# Tabellenzuordnung
Name = 0
Vorname = 1
```

## Tipps und Tricks

- Die Rückgabewerte der Fixture-Methoden sollten **"boolean"** sein. **"true"** wird dann als korrekt ausgeführter Test interpretiert. Alle anderen Rückgabetypen werden vom Test-Editor nicht interpretiert.
- Bei Änderungen in den XML-Dateien muss der *Test-Editor* nicht neu gestartet werden. Wenn man die Dateien entsprechend in die Dateistruktur des laufenden *Test-Editors* kopiert, einfach auf Projektebene die Bibliotheken neu laden lassen.



- Nach einem neuen Build des Fixtures muss nur die neue JAR ausgeliefert werden. Der *Test-Editor* muss nicht neu gestartet werden.
- Sollten ScenarioLibrary oder ElementListe geändert werden, stehen diese Änderungen nach dem Kopieren in den laufenden *Test-Editor* sofort zur Verfügung.

# Ältere Projekte migrieren

Dieses Kapitel ist dann wichtig, wenn bereits **Test-Projekte mit einer älteren Version des Test-Editor** erstellt wurden und jetzt in der neueren Version genutzt werden sollen. Zwischen bestimmten Versionen ist es notwendig ein paar Änderungen an Konfigurationsdateien vorzunehmen, die hier beschrieben werden.

## Projekte der Test-Editor Version 1.6.1 auf 1.7.0 migrieren

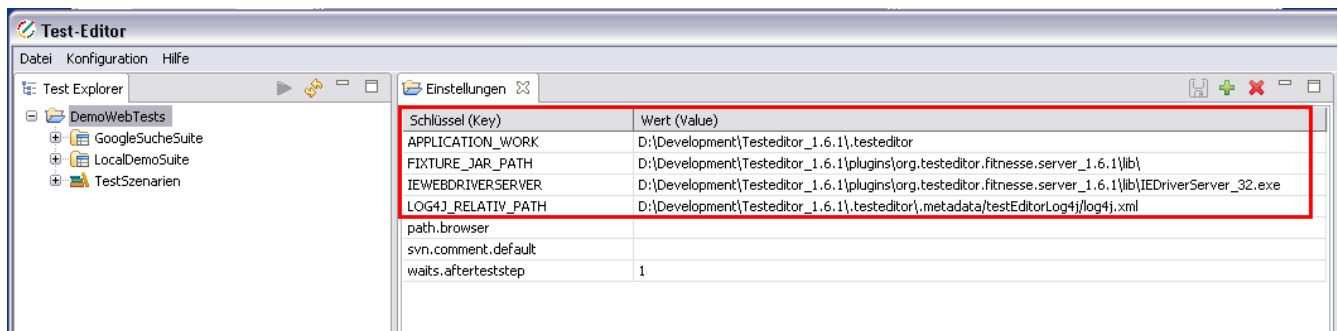
### ▼ Anpassen von Variablen auf der Einstellungsseite

#### Alte Einstellungen bis Version 1.6.1

Ab Version 1.7.0 wurden zur Vereinfachung, folgende Einträge aus den globalen Einstellungen des Test-Editors entfernt.

- APPLICATION\_WORK
- FIXTURE\_JAR\_PATH
- IEWEBDRIVERSERVER
- LOG4J\_RELATIV\_PATH

Um Fehler zu vermeiden, sollten die rot markierten Einträge über den Test-Editor aus den Einstellungen entfernt werden.



in der Test-Editor Version bis 1.6.1 sah die alte FitNesse Preferences Startseite der Projekte noch wie folgt aus

#### Alte Einstellungen bis Version 1.6.1

```
!*****> Preferences

!define TIMEOUT {10000}
!define TEST_SYSTEM {slim}

!path ${FIXTURE_JAR_PATH}TestEditorFixtureWeb-*.jar
!define BROWSER_PATH {${path.browser}}

!define LOGGING {${LOG4J_RELATIV_PATH}}
!define COMMAND_PATTERN { java -Dwaits.after.teststep=${waits.afterteststep}
-Dwebdriver.ie.driver=${SLIM_CMD_VAR_IEWEBDRIVERSERVER}
-Dlog4j.configuration=file:${SLIM_CMD_VAR_LOG4J_RELATIV_PATH} -cp %p %m -i
org.testeditor.fixture.core.interaction.TestEditorLoggingInteraction }

!define ELEMENT_LISTE {${APPLICATION_WORK}/DemoWebTests/ElementList.conf}

*****!

!contents -R2 -g -p -f -h
```

## Neue Einstellungen ab Version 1.7.0

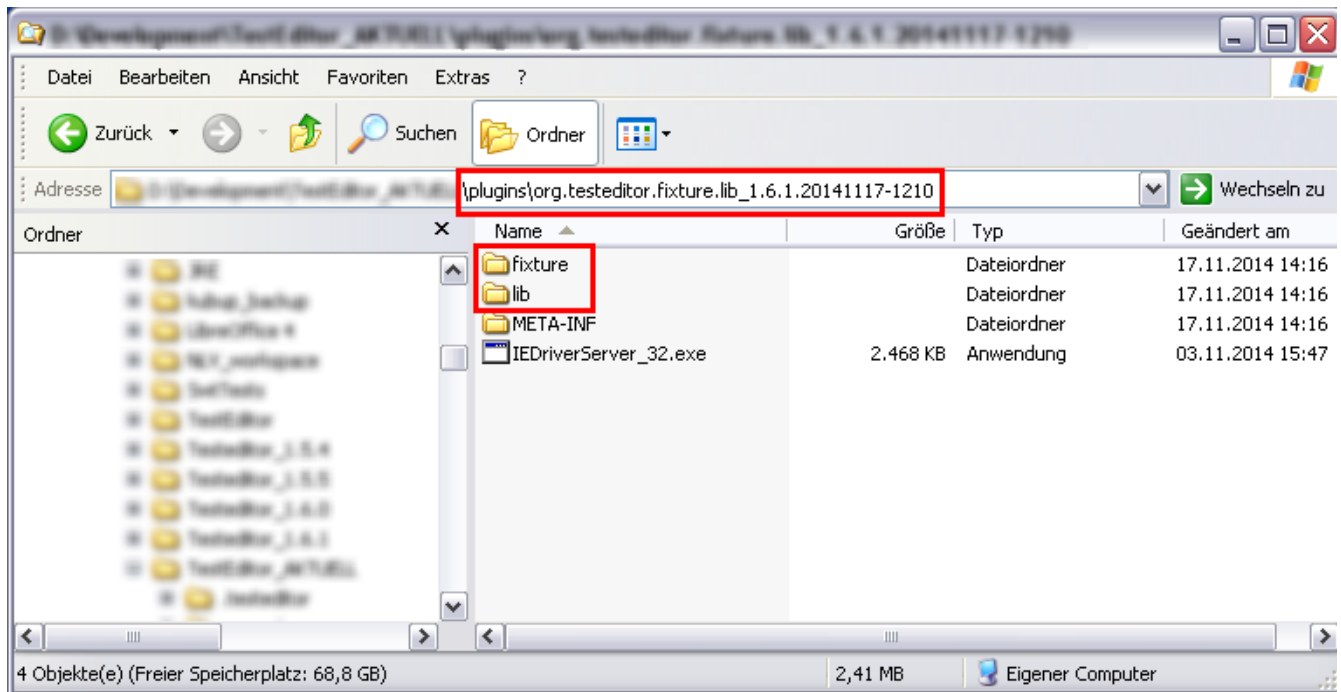
durch das Entfernen der obigen Einstellungen und der Einführung einer neuen Variablen, die beim Start des Test-Editors der Testumgebung zur Verfügung stehen und dadurch nicht über die globalen Einstellungen des Test-Editors gesetzt werden müssen, steht eine Möglichkeit zur Verfügung, die Pfade zu den Testtreibern (den sogenannten Fixtures) und den dazugehörigen Bibliotheken, anzugeben. Natürlich können diese Variablen für die bestehenden Projekte auf der FitNesse Preferences Seite (Hauptseite der jeweiligen Projekte) jederzeit angepasst werden.

### Neu hinzugefügte Variable

diese Variable führt zu dem Verzeichnis der Testtreiber (sog. Fixtures), bzw. unter dem untergeordneten Verzeichnis "/lib" auf alle benötigten Bibliotheken

Default-Verzeichniseinstellungen bei Neuanlage des DemoWebTests Projektes

FIXTURE\_LIB\_BUNDLE\_PATH = Test-Editor Installationsverzeichnis/plugins/



Auf der FitNesse Preferences Seite der jeweiligen Projekte können die Einstellungen für den Speicherort der Fixtures und der jeweiligen Bibliotheken folgendermaßen angepasst werden. Im Standardfall wird davon ausgegangen (siehe unteres Beispiel), dass die Fixtures in dem Installationsverzeichnis des Test-Editors gespeichert sind.



## Preferences Seite ab Version 1.7.0

```
!*****> Preferences

!define TIMEOUT {10000}
!define TEST_SYSTEM {slim}

!path ${FIXTURE_LIB_BUNDLE_PATH}/fixture/TestEditorFixtureWeb-*.jar
!path ${FIXTURE_LIB_BUNDLE_PATH}/lib/*.jar

!define BROWSER_PATH ${path.browser}

!define COMMAND_PATTERN {java -Dwaits.after.teststep=${waits.afterteststep}
-Dwebdriver.ie.driver=${SLIM_CMD_VAR_IWEBDRIVERSERVER}
-Dlog4j.configuration=file:${SLIM_CMD_VAR_LOG4J_RELATIV_PATH} -cp %p %m -i
org.testeditor.fixture.core.interaction.TestEditorLoggingInteraction }

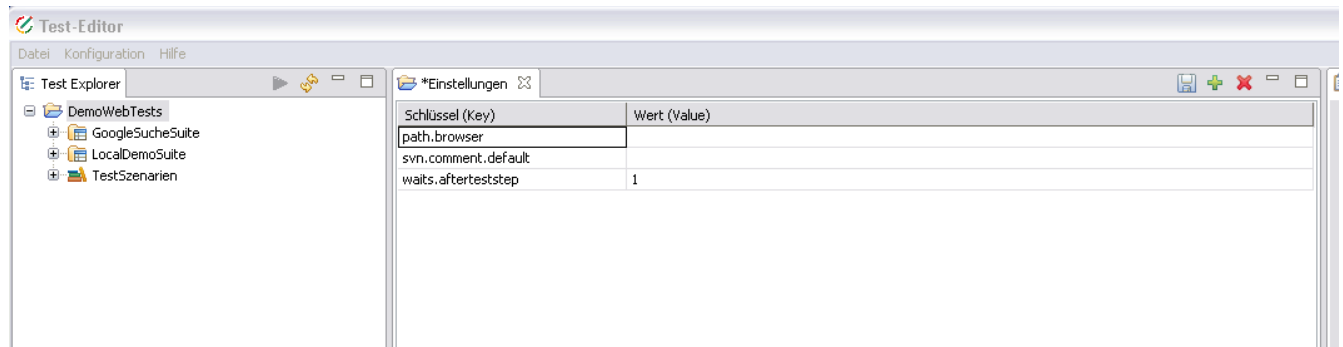
!define ELEMENT_LISTE ${APPLICATION_WORK}/DemoWebTests/ElementList.conf}

*****!

!contents -R2 -g -p -f -h
```

## Neue globale Einstellungsseite

Durch das Löschen der oben genannten 4 Einträge sieht die Standard Einstellungsseite folgendermaßen aus:



## Tipps zur Prüfung der Variablen

Es ist möglich auf der FitNesse Hauptseite die verwendeten Preferences anzuzeigen, hierfür muss man nur die Variable innerhalb des Preferences Blocks erneut eintragen und speichern.

```

!*****> Preferences

!define TIMEOUT {10000}
!define TEST_SYSTEM {slim}

!path ${FIXTURE_LIB_BUNDLE_PATH}/fixture/TestEditorFixtureWeb-*.jar
!path ${FIXTURE_LIB_BUNDLE_PATH}/lib/*.jar

!define BROWSER_PATH ${path.browser}}

!define COMMAND_PATTERN {java -Dwaits.after.teststep=${waits.afterteststep}
-Dwebdriver.ie.driver=${SLIM_CMD_VAR_IWEBDRIVERSEVER}
-Dlog4j.configuration=file:${SLIM_CMD_VAR_LOG4J_RELATIV_PATH} -cp %p %m -i
org.testeditor.fixture.core.interaction.TestEditorLoggingInteraction }

!define ELEMENT_LISTE ${APPLICATION_WORK}/DemoWebTests/ElementList.conf

BEISPIEL_FUER_DIE_PRUEFUNG_VON_VARIABLEN
Dies ist die Variable fuer FIXTURE_LIB_BUNDLE_PATH : ${FIXTURE_LIB_BUNDLE_PATH}
Dies ist die Variable fuer SLIM_CMD_VAR_LOG4J_RELATIV_PATH :
${SLIM_CMD_VAR_LOG4J_RELATIV_PATH}

*****!

!contents -R2 -g -p -f -h

```

dadurch kann man die übertragenen Werte prüfen (hier am Beispiel der Variable FIXTURE\_LIB\_BUNDLE\_PATH)



## DemoWebTests

Edit

Add

Tools

▼ Preferences
Expand All Collapse All

```

variable defined: TIMEOUT=10000
variable defined: TEST_SYSTEM=slim

classpath: D:\Development\TestEditor_AKTUELL\plugins\org.testeditor.fixture.lib_1.6.1.20141117-1210\fixture\TestEditorFixtureWeb-*.jar
classpath: D:\Development\TestEditor_AKTUELL\plugins\org.testeditor.fixture.lib_1.6.1.20141117-1210\lib\*.jar

variable defined: BROWSER_PATH=${path.browser}

variable defined: LOGGING=${LOG4J_RELATIV_PATH}
variable defined: COMMAND_PATTERN=java -Dwaits.after.teststep=${waits.afterteststep} -Dwebdriver.ie.driver=${SLIM_CMD_VAR_IWEBDRIVERSEVER}
-Dlog4j.configuration=file:${SLIM_CMD_VAR_LOG4J_RELATIV_PATH} -cp %p %m -i org.testeditor.fixture.core.interaction.TestEditorLoggingInteraction

variable defined: ELEMENT_LISTE=${APPLICATION_WORK}/DemoWebTests/ElementList.conf

BEISPIEL_FUER_DIE_PRUEFUNG_VON_VARIABLEN
Dies ist die Variable fuer FIXTURE_LIB_BUNDLE_PATH : D:\Development\TestEditor_AKTUELL\plugins\org.testeditor.fixture.lib_1.6.1.20141117-1210
Dies ist die Variable fuer SLIM_CMD_VAR_LOG4J_RELATIV_PATH : D:\Development\TestEditor_AKTUELL\testeditor\metadata\testEditorLog4j\log4j.xml

```

### ▼ Dynamische Portvergabe

Die Ports auf denen der Testserver in Zukunft startet wird dynamisch vergeben, es ist nicht mehr notwendig und auch nicht möglich diesen manuell zu setzen. Hierzu muß die config.tpr des Projektes angepasst werden, diese Datei befindet sich direkt im Wurzelverzeichnis des Projektes. Im folgende demonstriert am Beispiel der mitgelieferten Demoanwendung DemoWebTests.

config.tpr vorher

```
#####
# Test-Editor project configuration                                     #
# -----                                                         #
#                                                                    #
# Config file for the local settings of each Test-Editor project. #
# You may edit this file directly or via the Test-Editor's UI.    #
#####

tpr-version=1.2
# library (project specific meta data for any test case)
#####

library_id=org.testeditor.xmllibrary
library.xmllibrary.actiongroup=DemoWebTests/AllActionGroups.xml
library.xmllibrary.technicalbindings=DemoWebTests/TechnicalBindingTypeCollection.xml

# testautomat (FitNesse server settings)
#####

testautomat.port=8060
testautomat.serverid=fitnesse_based_1.2

# synchronization (team sharing e.g. via SVN)
#####

teamshare_plugin_id=
```

ändern in tpr-version=1.3

diese Zeile löschen

## config.tpr nachher

```
#####
# Test-Editor project configuration                                     #
# -----                                                         #
#                                                                    #
# Config file for the local settings of each Test-Editor project. #
# You may edit this file directly or via the Test-Editor's UI.    #
#####

tpr-version=1.3

# library (project specific meta data for any test case)
#####

library_id=org.testeditor.xmllibrary
library.xmllibrary.actiongroup=DemoWebTests/AllActionGroups.xml
library.xmllibrary.technicalbindings=DemoWebTests/TechnicalBindingTypeCollection.xml

# testautomat (FitNesse server settings)
#####

testautomat.serverid=fitnesse_based_1.2

# synchronization (team sharing e.g. via SVN)
#####

teamshare_plugin_id=
```

## Projekte der Test-Editor Version 1.5 zu 1.5.1 migrieren (nur bei Szenarien in Szenarien wichtig)

▼ [Test-Editor 1.5 auf 1.5.1 migrieren](#)

## Projekte der Test-Editor Version 1.5 zu 1.5.1 migrieren (nur bei Szenarien in Szenarien wichtig)

Für die Verwendung von **Szenarien in Szenarien** wurde ab dem Patch 1.5.1 das Layout der [Szenarien](#) geändert.

Für eine Migration ältere Szenarien sind folgende Schritte notwendig.

1. Sofern Szenarien in Szenarien verwendet werden, müssen alle Includes zu Beginn des Szenarios angegebene werden. Bisher waren die Includes mitten im Szenario angegeben. Dies interpretiert dann [FitNesse](#) nicht korrekt.  
Code Beispiel  
bisher:

```
!|scenario |LoginValidationSzenario _|Passwort, Land, TextVorhanden,
TextNichtVorhanden|
!include <DemoWebTests.TestKomponenten.OeffneStartseiteSzenario
!|script|
|Oeffne Startseite Szenario|
#
|note|Description: Dieses Szenario überprüft den Login auf der Starseite.| | | |
|note| Maske: Lokale Anmeldung|
|gebe in das Feld|password|den Wert|@Passwort|ein|
|wähle in Feld|land|den Wert|@Land|aus|
|klicke auf|login|
|note| Maske: Allgemein Browser|
|überprüfe ob der Text|@TextVorhanden|vorhanden ist|
|überprüfe ob nicht der Text|@TextNichtVorhanden|vorhanden ist|
|beende Browser|
```

neu:

```
!include <DemoWebTests.TestKomponenten.OeffneStartseiteSzenario
!|scenario |LoginValidationSzenario _|Passwort, Land, TextVorhanden,
TextNichtVorhanden|
!|script|
|Oeffne Startseite Szenario|
#
|note|Description: Dieses Szenario überprüft den Login auf der Starseite.| | | |
|note| Maske: Lokale Anmeldung|
|gebe in das Feld|user|den Wert|@Name|ein|
|gebe in das Feld|password|den Wert|@Passwort|ein|
|wähle in Feld|land|den Wert|@Land|aus|
|klicke auf|login|
|note| Maske: Allgemein Browser|
|überprüfe ob der Text|@TextVorhanden|vorhanden ist|
|überprüfe ob nicht der Text|@TextNichtVorhanden|vorhanden ist|
|beende Browser|
```

2. Leerzeilen und leere Anweisungen, wie beispielsweise ein "#" müssen entfernt werden.

neu:

```

!include <DemoWebTests.TestSzenarien.ApplikationStartSzenario
!|scenario |LoginValidationSzenarioNeu _|Name, Passwort, Land, TextVorhanden,
TextNichtVorhanden|
|note|Description: Dieses Szenario überprüft den Login auf der Starseite.|
|ApplikationStartSzenario|
|note| Maske: Lokale Anmeldung| | | |
|gebe in das Feld|user|den Wert|@Name|ein|
|gebe in das Feld|password|den Wert|@Passwort|ein|
|wähle in Feld|land|den Wert|@Land|aus|
|klicke auf|login|
|note| Maske: Allgemein Browser|
|überprüfe ob der Text|@TextVorhanden|vorhanden ist|
|überprüfe ob nicht der Text|@TextNichtVorhanden|vorhanden ist|

```

### 3. Vor dem Aufruf eines Szenarios in einem Szenario muss die Zeile

```
|note|scenario|
```

eingefügt werden.

neu:

```

!include <DemoWebTests.TestSzenarien.ApplikationStartSzenario
!|scenario |LoginValidationSzenarioNeu _|Name, Passwort, Land, TextVorhanden,
TextNichtVorhanden|
|note|Description: Dieses Szenario überprüft den Login auf der Starseite.|
|note|scenario|
|ApplikationStartSzenario|
|note| Maske: Lokale Anmeldung| | | |
|gebe in das Feld|user|den Wert|@Name|ein|
|gebe in das Feld|password|den Wert|@Passwort|ein|
|wähle in Feld|land|den Wert|@Land|aus|
|klicke auf|login|
|note| Maske: Allgemein Browser|
|überprüfe ob der Text|@TextVorhanden|vorhanden ist|
|überprüfe ob nicht der Text|@TextNichtVorhanden|vorhanden ist|

```

## Projekte der Test-Editor Version 1.4 zu 1.5 migrieren

Folgende Schritte müssen manuell für ein solches Projekt durchgeführt werden:

- Anpassen des Fixturenamen in der ScenarioLibrary für Projekte, die das Web-Fixture verwenden. Die ScenarioLibrary liegt im Wurzelknoten des Projekts und ist über die FitNesse-Oberfläche erreichbar.



# DemoWebTests

► Preferences

## Contents:

- [Element Liste](#)
- [Google Suche Suite \\*](#)
  - [Suche Akquinet Test +](#)
  - [Suche Signal Iduna Test +](#)
- [Local Demo Suite \\*](#)
  - [Login Suite \\* ...](#)
  - [Login Szenario Suite \\* ...](#)
- [Officewerker Suite \\*](#)
  - [Aufwand Suite \\* ...](#)
  - [Eingangsrechnungen Suite \\* ...](#)
  - [Login Suite \\* ...](#)
- [Scenario Library](#)
- [Test Komponenten \\*+](#)
  - [Browser Start Szenario](#)
  - [Login Validation Szenario](#)
  - [Oeffne Startseite Szenario](#)
  - [Suche Google Szenario](#)

auswählen

Es muss der Name der Library von GenericFixture in WebFixture geändert werden:



DemoWebTests

## ScenarioLibrary



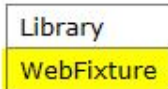
Library
GenericFixture

nach



DemoWebTests

## ScenarioLibrary



- Anpassen der Testkomponenten in den Properties vom Type "Suite" auf "static" ändern. Dazu muss der Properties-Dialog in der Fitness-Oberfläche geöffnet werden.



DemoWebTests

## TestKomponenten

Suite

Test

Edit

Add

Tools

Scenario Libraries

Expand

Properties  
Where Used

Die Eigenschaften der TestKomponenten von "Suite"



DemoWebTests

## TestKomponenten

Last modified anonymously

### Page properties

Page type:

☐ Static

☐ Test

☒ Suite

☐ Skip  
(Recursive)



nach "Static" ändern.



Last modified anonymously

## Page properties

**Page type:**

☒ Static  
☐ Test  
☐ Suite  
☐ Skip  
(Recursive)

Help text:   
Tags:   
VirtualWiki URL:

und anschließend speichern.

Weiterhin kann der Name TestSzenarien statt TestKomponenten verwendet werden.

- Handlungsempfehlung: Wenn man statt der fixen Versionsnummer des Fixtures in der Projektkonfiguration (die über die FitNesse-Oberfläche editiert werden kann) ein "\*" einfügt, so wird auch nach einer Versionsänderung immer das Fixture von dem angegebenen Typ gefunden.  
Z. B.: von "pfadZumWorkspace\org.testeditor.fitnesseserver\lib\TestEditorFixtureWeb-1.4.0.jar" nach "pfadZumWorkspace\org.testeditor.fitnesseserver\lib\TestEditorFixtureWeb-\*.jar" ändern.
- Bei der Umstellung von 1.4.0 nach 1.5.0 gehen die manuell angelegten Variablen im Menüpunkt Datei/Einstellungen und deren Inhalte verloren. Daher sollten diese gesichert werden. Diese müssen dann nach der Umstellung erneut angelegt werden.

## Projekte der Test-Editor Version kleiner als 1.4 zu 1.5 migrieren

### ✓ Test-Editor von 1.4 auf 1.5 migrieren

#### Projekte der Test-Editor Version kleiner als 1.4 zu 1.5 migrieren

Bei einigen Projekten sind im Test-Editor noch alte Versionen der

- TechnicalBindingTypeCollection.xml
- AllActionGroups.xml

enthalten.

Diese bedürfen einer inhaltlichen Änderung, da sich das Verzeichnis der XSD-Schemata geändert haben.

Der Inhalt der TechnicalBindingTypeCollection.xsd ist von



### altes Schema mit altem Verzeichnis

```
<?xml version="1.0" encoding="UTF-8"?>
<TechnicalBindingTypes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="../../../org.testeditor.fixture.scanner/resources/Tec
hnicalBindingTypeCollection.xsd"
    schemaVersion="1.1">
```

in

### neues Schema mit neuem Verzeichnis

```
<?xml version="1.0" encoding="UTF-8"?>
<TechnicalBindingTypes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="../../../org.testeditor.xmllibrary/resources/TechnicalBi
ndingTypeCollection.xsd"
    schemaVersion="1.1">
```

zu ändern.

Dieses Vorgehen ist analog mit der AllActionGroups.xml durchzuführen.

# Tests Headless in der Konsole ausführen

Der Test-Editor bietet die Möglichkeit Tests auszuführen, ohne dass die Benutzeroberfläche dafür gestartet werden muss. Dafür wird der Test-Editor als Java Prozess mit dem application Parameter gestartet:

```
java -jar
<TEST_EDITOR_HOME>plugins/org.eclipse.equinox.launcher_1.3.0.v20140415-2008.jar
-application org.testeditor.core.headlesstestrunner -consoleLog
ExecuteTest=<TestProjekt>.<TestSuite>
```

Die Parameter steuern folgendes:

-application org.testeditor.core.headlesstestrunner

Starten des Test-Editors ohne GUI um ein Test Object auszuführen

-consoleLog

Logausgaben in die Konsole umleiten

ExecuteTest=<TestProjekt>.<TestSuite>

Angabe welcher Test oder Testsuite ausgeführt werden soll.

Zusätzlich können Parameter angegeben werden, um das Setup flexibler zu gestalten:

-data <Workspace>

Mit -data lässt sich der default Workspace von <USER\_HOME>/testeditor überschreiben

-D<Variable>=<Wert>

Können Variablen in den Einstellungen des Test-Editors über den Kommandozeilen aufruf mit gegeben werden. Daraus lässt sich ein Beispiel für den Test-Editor zur Ausführung der Testsuite aus dem CI Build abbilden:

```
java
-DPATH_TO_TESTEDITOR_AUT=product/org.testeditor.product/target/products/TestEditor/linux/gtk/x86_64/testeditor -Daut.workspace.path=../testeditor_auth1
-Dwaits.afterteststep=10ms -jar
product/org.testeditor.product/target/products/TestEditor/linux/gtk/x86_64/plugins/org
.eclipse.equinox.launcher_1.3.0.v20140415-2008.jar -application
org.testeditor.core.headlesstestrunner -consoleLog -data ../test-editor-tests
ExecuteTest=TestEditorTests.AcceptanceForConDelivery
```

# CI-Build aufsetzen

Das folgende Kapitel beschreibt wie [Testfälle](#), die bereits vollständig automatisiert sind, im Kontext von einem **Continuous Integration Server (CI)** z.B. einmal am Tag als Regressionstest ausgeführt werden können.

Dafür können die Tests über einen FitNesse Server ausgeführt werden oder der Test-Editor führt diese Tests im Headless Modus aus. In beiden

## Fitnessse Server im CI nutzen

### CI Servers installieren

Zunächst muss ein CI Server installiert werden, die folgende Anleitung bezieht sich auf den **Jenkins**, der als Open Source Server unter folgender URL zur Verfügung steht: <http://jenkins-ci.org/>

Nach der Installation, muss über **Jenkins verwalten -> Plugin verwalten** das **Hudson FitNesse Plugin** installiert werden. Nach Neustart des CI-Servers ist das Plugin aktiv.

### Jenkins Job für Akzeptanttest einrichten

Anschließend kann ein neuer **Jenkins Job** für das zu testende Projekt angelegt werden. Im Folgenden wird ein Job für das DemoWebTests Projekt angelegt:

Der Neue Job wird als **"Free Stytle"-Softwareprojekt** bauen angelegt, in der Konfiguration wird ein **Build Schritt Exceution fitnessse tests** sowie ein **Post-Build Schritt Publish Fitnessse result report** hinzugefügt. Die Konfiguration wird entsprechend vorgenommen, ein Beispiel für die lokale Ausführung im Screenshot:

#### Buildverfahren

##### Execute fitnessse tests

Fitnessse Instance

☒ Fitnessse instance is already running

Fitnessse Host

Fitnessse Port

☐ Start new Fitnessse instance as part of build

Target

Target Page

Is target a suite? ☐

Output

HTTP Timeout (ms)

Test Timeout (ms)

Path to fitnessse xml results file

Löschen

Build-Schritt hinzufügen ▾

#### Post-Build-Aktionen

##### Publish Fitnessse results report

Path to fitnessse xml results file

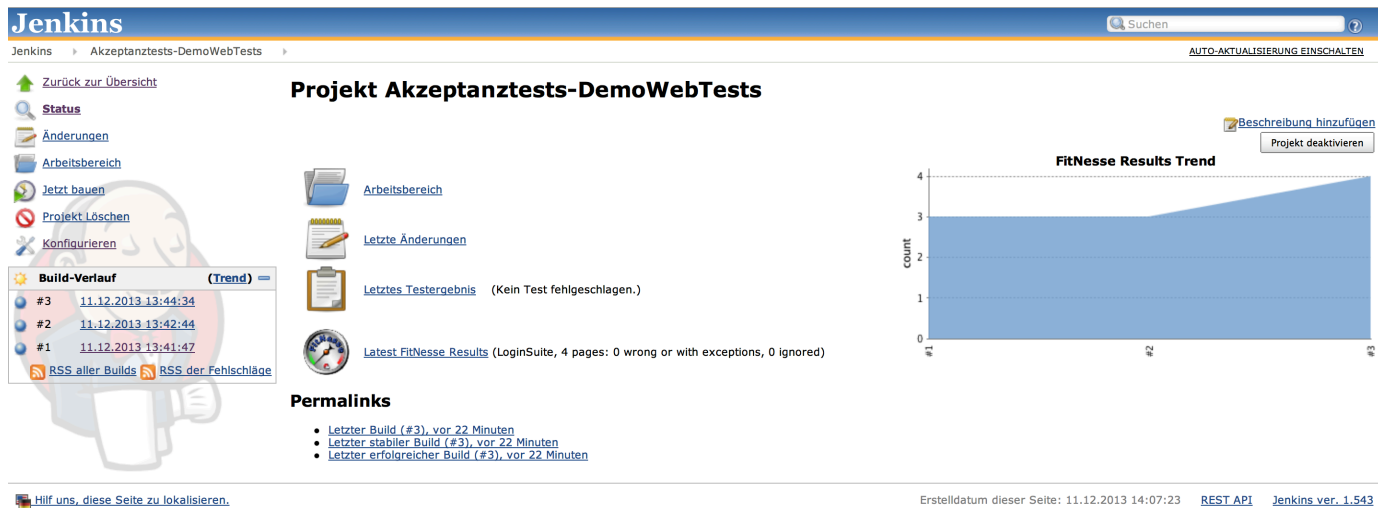
Löschen

Post-Build-Schritt hinzufügen ▾

Speichern Übernehmen

## Jenkins Job ausführen

Sobald der Jenkins Job ausgeführt wurde, wird ähnlich wie bei der Ausführung von JUnit-Tests ein **Test-Report** generiert. Das folgende Beispiel zeigt, dass im letzten Build vier Test ausgeführt wurden (im vorigen Build wurden drei Test ausgeführt). Sinnvoll ist es, diesen Job entsprechend so einzustellen, dass die Tests z.B. einmal am Tag ausgeführt werden.



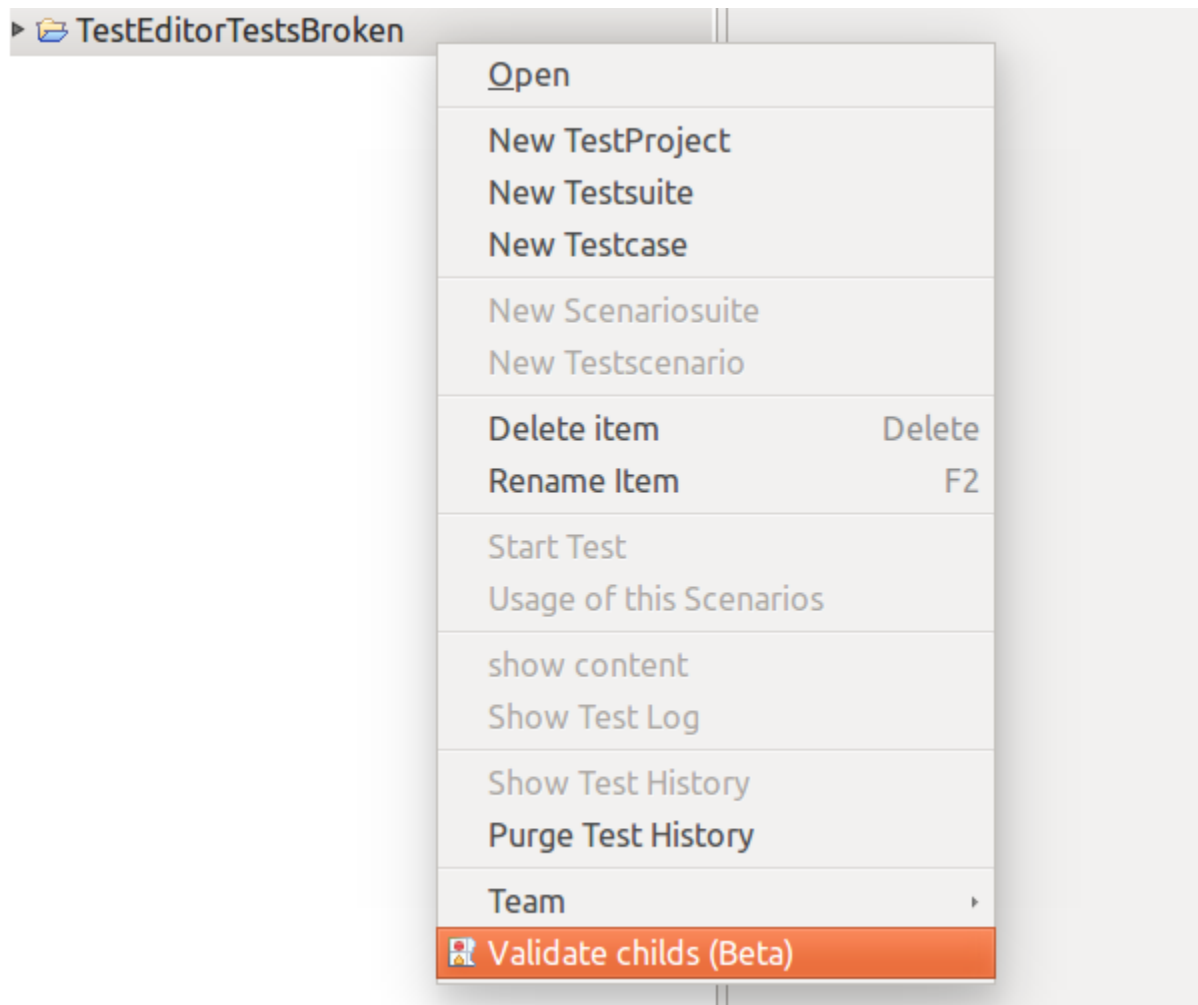
## Test-Editor Headless im CI ausführen

# Testbestand prüfen

## Prüfen ob Testfälle und Bibliothek zu einander passen

Wenn Testfälle überarbeitet oder migriert werden, kann es vorkommen, das die Testfälle nicht mehr zur Bibliothek passen. Andersherum kann natürlich auch die Bibliothek überarbeitet werden, was die, von dieser Bibliothek abhängigen Testfälle, fehlschlagen lassen kann.

Um nach solchen Änderungen die Testfälle auf ihre Konsistenz prüfen zu können, kann man die Testfälle validieren. Dafür ruft man Kontextmenü des Projektes oder der Testsuite den Menüpunkte Kindelemente validieren auf.



Nach dem Aufruf wird durch einen Fortschritts Dialog zunächst das Laden aller Kindelemente dargestellt. Diese werden danach darauf überprüft, ob alle Testschritte in den Testfällen durch Aktionen in der Bibliothek abgebildet werden können. Ist dies nicht der Fall, wird die entsprechende Zeile in einem Ergebnisfenster dargestellt. Die Zeilen werden dabei anhand ihres Testfalles oder der Testkomponente gruppiert. Sollten alle Testfälle lauffähig sein, bleibt der Bericht leer.

Testhistory Validation

Validation results for TestEditorTestsBroken with 7 broken Test elements.

- ▶ TestEditorTestsBroken.ProjektKonfigurationSuite.AlleProjekteLoeschenTestenObKeinsMehrDaDannDemoPr
- ▶ TestEditorTestsBroken.ProjektKonfigurationSuite.FehleingabenNeuesProjektWizard (8)
- ▼ TestEditorTestsBroken.ProjektKonfigurationSuite.FehleingabenRenameProjektWizard (2)
  - ✖ |ist der Button|wizard.rename.cancelButton|enabled|
  - ✖ |klicke auf|wizard.rename.cancelButton|
- ▶ TestEditorTestsBroken.ProjektKonfigurationSuite.ProjektEinstellungenTest (1)
- ▶ TestEditorTestsBroken.ProjektKonfigurationSuite.ProjektUmbenennen (2)
- ▶ TestEditorTestsBroken.ProjektKonfigurationSuite.TestHistorieVonTestFallOffenenProjektLoeschenHistorieNi

## Version des Testeditors und Version der Config.tpr

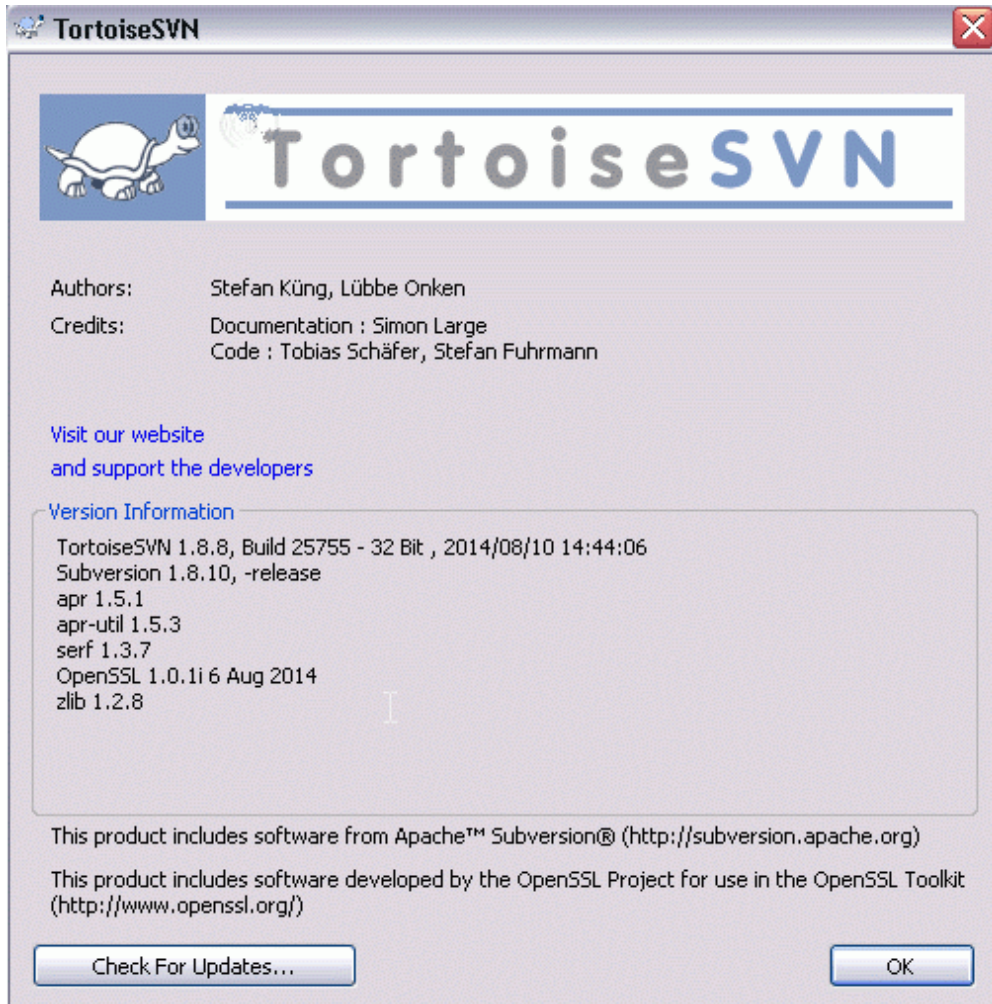
Version Testeditor	Version der config.tpr
1.1	?
1.1.1	?
1.2.0	?
1.5.2	1.1, 1.2
1.5.3	1.1, 1.2
1.5.4	1.1, 1.2
1.5.5	1.1, 1.2
1.6	1.1, 1.2

# Unterstützte SVN-Version

Man benötigt folgende SVN Version compatible Clients

Subversion 1.8.X

hier ein Beispiel für den Tortoise SVN Client





# Glossar

## Akzeptanztest

Mit Hilfe des Akzeptanztests oder auch Abnahmetests kann der Anwender überprüfen ob die gewünschten Funktionalitäten einer Software vorhanden sind. Dabei werden die Funktionalitäten anhand der Projektvorgaben überprüft. Der Anwender nutzt die Software und muss entsprechend die technischen Details der Umsetzung nicht kennen. Oft werden Alpha- oder Beta-Tests als Akzeptanztest genutzt.

## AUT

Application Under Test (auch SUT, System Under Test) meint die zu testende Software.

## Element-Liste

Die Element-Liste ist in der Datei `ElementList.conf` abgebildet und besteht aus Key-Value Paaren. Sie muss ebenfalls vom Projekt gepflegt werden. Der Key ist identisch mit dem "Locator" in der `ActionGroup.xml` und der Value ist jeweils der technische Schlüssel um ein Element auf dem AUT zu adressieren (also z.B. eine HTML ID oder ein X-Path Ausdruck).

## Fachsprache (DSL)

Eine Fachsprache, häufig auch als Domain Specific Language (DSL) bezeichnet, ist nichts anderes als eine Definition wie Testschritte sich generell zusammen setzen. So ist beispielsweise definiert ob ein Testschritt zur Erfassung eines Textes in einem Input-Feld "gebe in das Feld XYZ den Wert ABC ein" heißt, oder ob z.B. eine englische oder eine andere Formulierung gewählt werden soll. Diese Informationen werden in der Datei `TechnicalBindingTypeCollection.xml` hinterlegt. Änderungen sind in der R

## FitNesse

FitNesse <http://fitnesse.org/FitNesse.UserGuide> ist ein automatisiertes Test-Framework, mit dem Testfälle und Testergebnisse in einer Wiki-Struktur dargestellt werden. Der Test-Editor setzt auf diese Wiki-Struktur auf und gibt ihr eine Nutzeroberfläche, die auch technisch weniger versierten Nutzern ohne Vorkenntnisse nutzen können.

## Fixture

Fixtures verbinden die Wiki-Struktur von FitNesse mit der Application Under Test (AUT). Im Falle des Test-Editors handelt es sich dabei um Java-Klassen.

## Masken und Testschritte

Die nächste Konfigurationsdatei muss von einem Projekt entsprechend erweitert werden, sie beschreibt welche Testschritte auf den jeweiligen Masken der AUT überhaupt möglich sind. Wichtig ist zu unterscheiden, dass hier keine konkreten Testfälle gespeichert werden, sondern nur die Meta-Informationen zu einem Projekt, also welche Aktionen überhaupt möglich sind (z.B. Klicke auf Login, Gebe in Name einen Wert ein usw.). Diese Informationen sind in der `ActionGroup.xml` gespeichert. Sie verweist dabei

## Port

Ihre Aktivitäten im Internet werden durch sogenannte „Services“ organisiert, von denen jeder seine Kennnummer hat, die als „Portnummer“ (kurz: Port [http://de.wikipedia.org/wiki/Port\\_%28Protokoll%29](http://de.wikipedia.org/wiki/Port_%28Protokoll%29)) bezeichnet wird.

## RAP

Remote Application Platform [http://de.wikipedia.org/wiki/Remote\\_Application\\_Platform](http://de.wikipedia.org/wiki/Remote_Application_Platform) ist ein Eclipse-Plugin, zur Entwicklung von Web-2.0-Anwendungen.

## REST

Representational State Transfer [http://de.wikipedia.org/wiki/Representational\\_State\\_Transfer](http://de.wikipedia.org/wiki/Representational_State_Transfer) ist ein Programmierparadigma für Webanwendungen.

## SOAP

Simple Object Access Protocol <http://de.wikipedia.org/wiki/SOAP> mit dessen Hilfe Daten zwischen Systemen ausgetauscht werden können. Dazu gehören auch sogenannte Remote Procedure Calls, also der Aufruf einer Prozedur durch ein anderes System.

## Szenario

Szenarien sind eine Gruppe von Testschritten die zu einem oder mehreren Testfällen hinzugefügt werden können. Diese Testschritte sind so angeordnet das sie logisch sinnvoll sind. So wird z. B. das Login für mehrere Testfälle benötigt. Indem man das Login als Szenario vorbereitet muss für jeden Testfall nur noch das Szenario hinzugefügt und mit Login-Daten bestückt werden. Diese Definition entspricht auf der Szenariodefinition von FitNesse.

## Szenariosuite

Szenariosuiten können angelegt werden um Szenarien thematisch zu gruppieren.

## Test

Ein Test oder Testlauf die Ausführung eines Testfalls oder eine Testsuite

## Testfall

Ein Testfall ist ein logische Abfolge von Testschritten, die in der Ausführung eine Funktionalität des Systems prüfen.

## Testlog

Das Testlog stellt die Meldungen des Systems dar, die während der entsprechende Test durchgelaufen ist, aufgezeichnet wurden.

## Teststruktur

Eine Teststruktur beschreibt folgende Artefakte.

## Testsuite

Um Testfälle logisch gruppieren zu können, können diese in einer entsprechenden Testsuite abgelegt werden.

## Test-Treiber

Ein Test-Treiber ist eine Software, die eine Schnittstelle erzeugt, um Systeme zu steuern. Die Treiber müssen dabei abhängig von der jeweiligen Software gewählt werden.

## Widget

Widget ist ein Überbegriff für die Bedienelemente einer Oberfläche. Bedienelemente können Eingabefelder, Buttons, usw. sein.

## XML

Die Extensible Markup Language [http://de.wikipedia.org/wiki/Extensible\\_Markup\\_Language](http://de.wikipedia.org/wiki/Extensible_Markup_Language) ist eine Sprache zur Darstellung hierarchisch strukturierter Daten in Form von Textdateien.

## Akzeptanztest

Mit Hilfe des Akzeptanztests oder auch Abnahmetests kann der Anwender überprüfen ob die gewünschten Funktionalitäten einer Software vorhanden sind. Dabei werden die Funktionalitäten anhand der Projektvorgaben überprüft. Der Anwender nutzt die Software und muss entsprechend die technischen Details der Umsetzung nicht kennen. Oft werden Alpha- oder Beta-Tests als Akzeptanztest genutzt. Der Test-Editor unterstützt den Akzeptanztest, indem die Testfälle bereits zur Konzeptionierungsphase erfasst werden können. Dadurch können sinkt die Wahrscheinlichkeit das Sonderfälle im späteren Testing vergessen werden.

# AUT

Application Under Test (auch SUT, System Under Test) meint die zu testende Software.

## Element-Liste

Die Element-Liste ist in der Datei **ElementList.conf** abgebildet und besteht aus Key-Value Paaren. Sie muss ebenfalls vom Projekt gepflegt werden. Der Key ist identisch mit dem "Locator" in der ActionGroup.xml und der Value ist jeweils der technische Schlüssel um ein Element auf dem AUT zu adressieren (also z.B. eine HTML ID oder ein X-Path Ausdruck).

## Fachsprache (DSL)

Eine Fachsprache, häufig auch als Domain Specific Language (DSL) bezeichnet, ist nichts anderes als eine Definition wie Testschritte sich generell zusammen setzen. So ist beispielsweise definiert ob ein Testschritt zur Erfassung eines Textes in einem Input-Feld "gebe in das Feld XYZ den Wert ABC ein" heißt, oder ob z.B. eine englische oder eine andere Formulierung gewählt werden soll. Diese Informationen werden in der Datei **TechnicalBindingTypeCollection.xml** hinterlegt. Änderungen sind in der Regel nur dann vorzunehmen, wenn ein Projekt eine eigene Fachsprache entwickelt.

# FitNesse

[FitNesse](#) ist ein automatisiertes Test-Framework, mit dem Testfälle und Testergebnisse in einer Wiki-Struktur dargestellt werden. Der Test-Editor setzt auf diese Wiki-Struktur auf und gibt ihr eine Benutzeroberfläche, die auch technisch weniger versierten Nutzern ohne Vorkenntnisse nutzen können.

## Fixture

Fixtures verbinden die Wiki-Struktur von FitNesse mit der Application Under Test (AUT). Im Falle des Test-Editors handelt es sich dabei um Java-Klassen.



## Masken und Testschritte

Die nächste Konfigurationsdatei muss von einem Projekt entsprechend erweitert werden, sie beschreibt welche Testschritte auf den jeweiligen Masken der AUT überhaupt möglich sind. Wichtig ist zu unterscheiden, dass hier keine konkreten Testfälle gespeichert werden, sondern nur die Meta-Informationen zu einem Projekt, also welche Aktionen überhaupt möglich sind (z.B. Klicke auf **Login**, Gebe in **Name** einen **Wert** ein usw.). Diese Informationen sind in der **ActionGroup.xml** gespeichert. Sie verweist dabei auf die Fachsprache der TechnicalBinding.xml.

# Port

Ihre Aktivitäten im Internet werden durch sogenannte „Services“ organisiert, von denen jeder seine Kennnummer hat, die als „**Portnummer**“ (kurz: **Port**) bezeichnet wird.

# RAP

Remote Application Platform ist ein Eclipse-Plugin, zur Entwicklung von Web-2.0-Anwendungen.

# REST

Representational State Transfer ist ein Programmierparadigma für Webanwendungen.

# SOAP

Simple Object Access Protocol mit dessen Hilfe Daten zwischen Systemen ausgetauscht werden können. Dazu gehören auch sogenannte Remote Procedure Calls, also der Aufruf einer Prozedur durch ein anderes System.

## Szenario

Szenarien sind eine Gruppe von Testschritten die zu einem oder mehreren Testfällen hinzugefügt werden können. Diese Testschritte sind so angeordnet das sie logisch sinnvoll sind. So wird z. B. das Login für mehrere Testfälle benötigt. Indem man das Login als Szenario vorbereitet muss für jeden Testfall nur noch das Szenario hinzugefügt und mit Login-Daten bestückt werden. Diese Definition entspricht auf der Szenariodefinition von FitNesse.

## Szenariosuite

Szenariosuiten können angelegt werden um Szenarien thematisch zu gruppieren.

# Test

Ein Test oder Testlauf die Ausführung eines Testfalls oder eine Testsuite



# Testfall

Ein Testfall ist eine logische Abfolge von Testschritten, die in der Ausführung eine Funktionalität des Systems prüfen.

Jeder Testfall beschreibt:

- was (Funktion, Genauigkeit, usw.) zu prüfen ist,
- welche Ausgangssituation hierfür erforderlich ist,
- welche Eingaben (Daten, Signale, Zeitbedingungen, usw.) notwendig sind und
- welche Ergebnisse (Ausgaben, Reaktionen) zu erwarten sind.

## Testlog

Das Testlog stellt die Meldungen des Systems dar, die während der entsprechende Test durchgelaufen ist, aufgezeichnet wurden.

# Teststruktur

Eine Teststruktur beschreibt folgende Artefakte.

- Testfall
- Test-Suite
- Testkomponente

## Testsuite

Um Testfälle logisch gruppieren zu können, können diese in einer entsprechenden Testsuite abgelegt werden.

# Test-Treiber

Ein Test-Treiber ist eine Software, die eine Schnittstelle erzeugt, um Systeme zu steuern. Die Treiber müssen dabei abhängig von der jeweiligen Software gewählt werden.

Art der Anwendung	Treiber
Swing	FEST
Web	Selenium
SWT und RCP	SWT Bot
Datenbankzugriffe	DB Slim

# Widget

Widget ist ein Überbegriff für die Bedienelemente einer Oberfläche. Bedienelemente können Eingabefelder, Buttons, usw. sein.

# XML

Die [Extensible Markup Language](#) ist eine Sprache zur Darstellung hierarchisch strukturierter Daten in Form von Textdateien.

# Release Notes

Die folgenden Release-Notes dienen der Übersicht, welche Features in welchem *Test-Editor* Release umgesetzt wurden:

## Release 1.8.0 (Juli 2015)

<b>Teamfunktionalität</b>	<ul style="list-style-type: none"><li>• Abbrechen Projekt Laden über SVN (<a href="#">TE-25</a>)</li><li>• Ein/Ausgehende Änderungen anzeigen (<a href="#">TE-190</a>)</li><li>• Anzeige der Revisionsnummer nach Freigabe (<a href="#">TE-94</a>)</li></ul>
<b>Testfälle</b>	<ul style="list-style-type: none"><li>• Duplizieren von Testfällen (<a href="#">TE-70</a>)</li><li>• Metadaten verwalten (<a href="#">TE-122</a>)</li><li>• Suche nach Testfällen (<a href="#">TE-141</a>)</li></ul>
<b>Log/Statusanzeige</b>	<ul style="list-style-type: none"><li>• Hervorheben der Logeinträge in der Logview (<a href="#">TE-113</a>)</li><li>• Statusanzeige beim Beenden des Editors (<a href="#">TE-29</a>)</li></ul>
<b>Konfiguration</b>	<ul style="list-style-type: none"><li>• Wartezeiten über Variable "waits.after.teststep" in Millisekunden einstellbar (<a href="#">TE-114</a>)</li><li>• Automatisches Nachladen der Konfigurationsänderungen (<a href="#">TE-126</a>)</li></ul>
<b>Fehlerbehebung</b>	<ul style="list-style-type: none"><li>• Reiter verschieben führt zu Menüverdopplung (<a href="#">TE-55</a>)</li><li>• Diverse Fehler behoben (<a href="#">TE-103</a>, <a href="#">TE-47</a>)</li></ul>

## Release 1.7.1 (März 2015)

<b>Test-Editor</b>	<ul style="list-style-type: none"><li>• Szenarioparameter werden nicht mehr mehrfach eingefügt (<a href="#">TE-34</a>).</li><li>• Sonderzeichen "{" kann nun in Eingabefelder eingegeben werden (<a href="#">TE-31</a>).</li></ul>
<b>SVN-Funktionalität</b>	<ul style="list-style-type: none"><li>• Suiten, Testfälle, die unter SCM Kontrolle stehen, können umbenannt werden.</li><li>• bestehende Projekte mit dem selben Namen werden erneut geladen und dadurch überschrieben (<a href="#">TE-32</a>).</li><li>• Fehlerbehebung an Markierungen geänderter Ressourcen (<a href="#">TE-28</a>, <a href="#">TE-79</a>).</li><li>• Bei fehlerhaften Eingabedaten (Benutzername, URL, Passwort) bleibt der Wizarddialog zur weiteren Bearbeitung offen (<a href="#">TE-89</a>).</li></ul>
<b>Test-Log</b>	<ul style="list-style-type: none"><li>• Das Test-Log wird nun in einer View statt einem Dialog angezeigt (<a href="#">TE-88</a>).</li></ul>

## Release 1.7.0 (Februar 2015)

<b>Test-Explorer</b>	<ul style="list-style-type: none"><li>• Markierung geänderter Testfälle, Testsuiten usw. mit einem Sternchen im Test-Explorer. (<a href="#">TE-981</a>)</li><li>• Anzeige aller Änderungen als Liste. (<a href="#">TE-1539</a>)</li></ul>
<b>SVN-Funktionalität</b>	<ul style="list-style-type: none"><li>• Änderungen verwerfen (Revert-Funktion). (<a href="#">TE-1399</a>)</li><li>• Projekt trennen. (<a href="#">TE-983</a>)</li><li>• Projekte, Testfälle, Szenarien, die unter SCM Kontrolle steht können nicht mehr umbenannt werden.</li><li>• Versionsnummer der verwendeten SVN-Kit Version ist im Info-Dialog sichtbar. (<a href="#">TE-1465</a>)</li><li>• Kommentar beim Freigeben von Teststrukturen eingeben (<a href="#">TE-1426</a>)</li></ul>
<b>Testhistorie</b>	<ul style="list-style-type: none"><li>• Anzeige der Testhistorie innerhalb des Test-Editors. (<a href="#">TE-1552</a>)</li></ul>
<b>Performanceoptimierung</b>	<ul style="list-style-type: none"><li>• Änderungen an Teststrukturen nun über Dateioperation statt Rest-Aufrufen. (<a href="#">TE-1506</a>)</li></ul>
<b>Port</b>	<ul style="list-style-type: none"><li>• Der Fitness-Port wird nun automatisch generiert und kann händisch nicht mehr geändert werden. (<a href="#">TE-1503</a>)</li></ul>
<b>Fehlerbehebung</b>	<ul style="list-style-type: none"><li>• Fehler und Fehler-Handling im SVN-Bereich wurden verbessert. (<a href="#">TE-1435</a>, <a href="#">TE-1449</a>)</li></ul>
<b>Verbesserte Bedienung der Tabellen</b>	<ul style="list-style-type: none"><li>• Leere Spalten in Tabellen werden falsch dargestellt. (<a href="#">TE-1443</a>)</li></ul>



## Release 1.6.1 (Oktober 2014)

<b>Fehlerbehebung</b>	<ul style="list-style-type: none"><li>• Copy/Paste einer Tabelle innerhalb eines Szenarios (<a href="#">TE-1475</a>)</li><li>• Scenariotabellen in Szenarien werden fehlerhaft abgespeichert (<a href="#">TE-1477</a>)</li></ul>
-----------------------	--

## Release 1.6.0 (Oktober 2014)

<b>UI-Scanner</b>	<ul style="list-style-type: none"><li>• Webseiten können gescannt werden, um Vorlagen für die AllActionGroups und die TechnicalBindingTypeCollections zu erzeugen</li></ul>
<b>Test-Suite</b>	<ul style="list-style-type: none"><li>• Wenn einer Suite neue referenzierte Testfälle hinzugefügt werden, wird beim wiederholten Öffnen des Dialogs die alte Auswahl wiederhergestellt</li><li>• Wenn man die Liste der referenzierten Testfälle öffnet, wird die Anzahl der referenzierten Testfälle angezeigt</li></ul>
<b>SVN Kommentare</b>	<ul style="list-style-type: none"><li>• Beim Teilen von Projekten und freigeben von Änderungen kann jetzt ein Kommentar eingegeben werden</li></ul>
<b>Fehlerbehebung</b>	<ul style="list-style-type: none"><li>• Fehler und Fehler-Handling im SVN-Bereich wurden verbessert</li><li>• Bearbeiten der Testkonfiguration bereinigt</li><li>• Exception-Handling wurde verbessert</li></ul>

## Release 1.5.4 (Oktober 2014)

<b>Fehlerbehebung</b>	<ul style="list-style-type: none"><li>• Tabellen (Excel, CVN) Import mit leeren Zellen funktioniert (<a href="#">TE-1412</a>)</li><li>• Einsatz von Umgebungsvariablen in Szenarien möglich (<a href="#">TE-1425</a>)</li><li>• Umbenennen eines nicht geteilten Projektes (<a href="#">TE-1416</a>)</li><li>• SzenarioView wird korrekt aktualisiert (<a href="#">TE-1409</a>)</li><li>• Status in der Testhistorie stimmt mit dem Status des Test-Explorers überein (<a href="#">TE-1400</a>)</li><li>• Geöffnete Testobjekte werden nach Aktualisierung des Baumes neu geladen (<a href="#">TE-1407</a>, <a href="#">TE-1427</a>)</li><li>• Beim Fehlen von XSD-Schemata Dateien wird eine sprechende Fehlermeldung ausgegeben (<a href="#">TE-1311</a>)</li><li>• Leere Testsuite ist nun ausführbar (<a href="#">TE-1379</a>)</li><li>• Darstellung von Parametern im Test-Explorer abhängig von ihrem Inhalt (<a href="#">TE-1138</a>)</li></ul>
-----------------------	--

## Release 1.5.3 (Juli 2014)

<b>Testbestand prüfen</b>	<ul style="list-style-type: none"><li>• Testfallbeschreibung kann automatisch auf Konsistenz bezüglich der genutzten Bibliothek (TechnicalBindingTypeCollection.xml und AllActionGroups.xml) geprüft werden. Inkonsistenzen werden im Test-Editor als fehlerhaft dargestellt.</li></ul>
<b>Perspektive speichern</b>	<ul style="list-style-type: none"><li>• Die Anordnung der Bereiche (Test-Explorer, Beschreibung, Testschritt, usw.) des Test-Editors werden beim Beenden gespeichert. Diese können jederzeit auf den Auslieferungszustand des Test-Editors zurückgesetzt werden.</li></ul>
<b>Performance</b>	<ul style="list-style-type: none"><li>• Performanceverbesserung (Öffnen von größeren Testfällen sowie Öffnen von Suites mit einer großen Anzahl an Testfällen ist schneller)</li></ul>
<b>Fehlerbehebung</b>	<ul style="list-style-type: none"><li>• SVN- Handling Fehlermeldungen verbessert</li><li>• Wiki Parseranpassungen (Parameter werden besser unterstützt und Fehlerhandling wurde verbessert)</li></ul>

## Release 1.5.2 (Juni 2014)

<b>Verbesserte Bedienung der Tabellen</b>	<ul style="list-style-type: none"><li>• Die Tabellen für die Systemkonfiguration und die Massentests können einfacher bedient werden</li></ul>
<b>Fortschrittsanzeige bei Teamarbeit</b>	<ul style="list-style-type: none"><li>• Beim Freigeben/Aktualisieren von Testfällen/Szenarien/Suites wird der Fortschritt angezeigt</li></ul>
<b>Performance</b>	<ul style="list-style-type: none"><li>• Performanceverbesserung</li></ul>

<b>Fehlerbehebung</b>	<ul style="list-style-type: none"> <li>• Es wurden diverse Fehler behoben</li> </ul>
-----------------------	--

## Release 1.5.1 (April 2014)

<b>Test abbrechen</b>	<ul style="list-style-type: none"> <li>• Das Abbrechen eines laufenden Tests kann jetzt schneller durchgeführt werden, ohne langes Warten</li> </ul>
<b>Fortschrittsanzeige bei Teamarbeit</b>	<ul style="list-style-type: none"> <li>• Beim Freigeben/Aktualisieren von Testfällen/Szenarien/Suites wird der Fortschritt angezeigt</li> </ul>
<b>Performance</b>	<ul style="list-style-type: none"> <li>• Performanceverbesserung</li> </ul>
<b>Fehlerbehebung</b>	<ul style="list-style-type: none"> <li>• Es wurden diverse Fehler behoben</li> </ul>

## Release 1.5 (April 2014)

<b>TestSuite verwalten</b>	<ul style="list-style-type: none"> <li>• TestSuiten können Verweise auf Testfälle verwalten und so als Container für die Testausführung dienen.</li> </ul>
<b>Projekte verwalten</b>	<ul style="list-style-type: none"> <li>• Erweiterung der Unterstützung bei der Synchronisierung von Projektdateien über SVN.</li> </ul>
<b>Testscenarien verwalten</b>	<ul style="list-style-type: none"> <li>• Umbenennung von TestKomponenten zu TestSzenarien</li> <li>• Die TestSzenarien sind jetzt keine Suite mehr und daher nicht mehr als Test ausführbar</li> <li>• Es sind Untergruppierungen für die Szenarien unterhalb der TestSzenarien möglich.</li> </ul>

## Release 1.4 (Februar 2014)

<b>Test-Editor Einstellungen</b>	<ul style="list-style-type: none"> <li>▪ Globale Variablen (z.B. Browser-Pfad, Fixture-Pfad) können zentral gesetzt werden (absolute Pfade in einzelnen Konfigurationsseiten entfallen damit)</li> <li>▪ Projekt Konfiguration für die Demo-Projekte vereinheitlicht</li> </ul>
<b>Projekte verwalten</b>	<ul style="list-style-type: none"> <li>▪ Projekte können über SVN geteilt werden</li> <li>▪ Projekte können aus dem SVN geladen werden</li> <li>▪ Testobjekte eines Projekt (SVN) können aktualisiert/freigegeben werden</li> </ul>
<b>Testfälle editieren</b>	<ul style="list-style-type: none"> <li>▪ Auswahlboxen können jetzt editiert werden, wodurch ein direkter Filter der möglichen Ergebnisse gesetzt wird</li> <li>▪ In Szenarien können auch für Auswahllisten Platzhalter mit @ gesetzt werden</li> <li>▪ Bei Wechsel zwischen mehreren Testfällen bleibt die Auswahl (z.B. Testschritt) erhalten</li> </ul>
<b>Testhistorie löschen</b>	<ul style="list-style-type: none"> <li>▪ Testhistorie kann gelöscht werden</li> </ul>

## Release 1.3 (Dezember 2013)

<b>Testfälle editieren</b>	<ul style="list-style-type: none"> <li>▪ Performanceoptimierung beim Öffnen und Bearbeiten von Testfällen</li> </ul>
<b>Testhistorie anzeigen</b>	<ul style="list-style-type: none"> <li>▪ Anzeigen einer lokalen Historie der Test-Ausführung</li> </ul>

Folgende **Bugs** wurden behoben:

- Speichern vor Testausführung kann korrekt abgebrochen werden
- Beim Einfügen von Testschritten bleibt der Scrollbereich bestehen
- Copy und Paste Funktion in Eingabefeldern sichergestellt

## Release 1.2 (November 2013)

<b>Projekte verwalten</b>	<ul style="list-style-type: none"> <li>■ Projekte können über entsprechende Wizards angelegt, umbenannt und gelöscht werden</li> <li>■ Optimierung der Fehlermeldungen in allen Wizards (Anlage, Bearbeiten, Löschen)</li> </ul>
<b>Testfälle editieren</b>	<ul style="list-style-type: none"> <li>■ Massentestdaten (Tabelle mit Testdaten) werden optimiert dargestellt</li> </ul>
<b>Test ausführen</b>	<ul style="list-style-type: none"> <li>■ eine Suite (auch mit verschachtelten Sub-Suiten) kann als Test ausgeführt werden</li> <li>■ Optimierung des Live-Logs bei der Test-Ausführung</li> </ul>

## Release 1.1 (Oktober 2013)

<b>Szenarien editieren</b>	<ul style="list-style-type: none"> <li>■ Es können Szenarien erstellt werden, die in Testfällen benutzt werden können</li> <li>■ Szenarien können 0-n Parameter erwarten, Parameter werden über @... bei der Eingabe in Textfeldern festgelegt</li> </ul>
<b>Testfälle editieren</b>	<ul style="list-style-type: none"> <li>■ Ein Testfall kann ein bzw. mehrere Szenarien verwenden</li> <li>■ Erwartet das Szenario Parameter werden die Daten tabellarisch erfasst oder via Excel- bzw. CSV-Datei importiert</li> <li>■ Dadurch dass die Tabelle n-Zeilen beinhaltet kann, können verschiedene Testdurchläufe definiert werden (= Massentest)</li> </ul>
<b>Test-Bibliothek verwalten</b>	<ul style="list-style-type: none"> <li>■ Pro Projekt kann eingestellt werden, ob die standardisierte XML Bibliothek des Test-Editors oder eine eigene projektspezifische Impl. verwendet wird</li> <li>■ Die projektspezifische Impl. wird als eigenes Bundle im Zusammenhang mit dem Test-Editor verwendet (Plugin-Mechanismus)</li> <li>■ Die technische Locator ID (z.B. als Key zur Elementliste) kann direkt in der XML Konfiguration gepflegt werden, wodurch die Konfiguration vereinfacht wurde</li> </ul>
<b>Test ausführen</b>	<ul style="list-style-type: none"> <li>■ Während der Testausführung wird ein "Live-Log" angezeigt, der einzelne Testschritte dokumentiert</li> </ul>

## Release 1.0 (Juni 2013)

<b>Test-Editor starten</b>	<ul style="list-style-type: none"> <li>■ Der Test-Editor ist über eine ausführbare Datei startbar (Windows und Linux)</li> <li>■ Anzeige eines Splash-Screens mit Logo</li> <li>■ Starten der FitNesse Server: Pro Projekt wird ein FitNesse Server lokal gestartet</li> <li>■ Beenden der FitNesse Server: Beim Schließen des Test-Editors werden diese gestoppt</li> </ul>
<b>Projekte verwalten</b>	<ul style="list-style-type: none"> <li>■ Unterstützung mehrerer Projekte im Homeverzeichnis des Test-Editors</li> <li>■ Je Projekt gibt es eine Projekt Konfigurationsdatei (z.B. mit FitNesse Port etc.)</li> <li>■ Zentrale Projektkonfigurationen können über die UI eingestellt werden</li> </ul>
<b>Test-Explorer bedienen</b>	<ul style="list-style-type: none"> <li>■ Testfälle können geöffnet werden (im Hauptbereich wird der Testfall angezeigt)</li> <li>■ Testfälle und Suiten können angelegt und umbenannt werden</li> <li>■ Testfälle und Suiten können einzeln oder auch mehrere gleichzeitig gelöscht werden</li> <li>■ Ein Test kann zu einem Testfall gestartet werden</li> <li>■ Quellcode zu einer Suite oder Testfall kann angezeigt werden</li> </ul>
<b>Testfälle editieren</b>	<ul style="list-style-type: none"> <li>■ Beschreibungen können erzeugt, geändert und gelöscht werden</li> <li>■ Testschritte können erzeugt, geändert und gelöscht werden (Auswahl über Maske und Schritt)</li> <li>■ Bestehende Testschritte werden validiert und eine Fehlermeldung angezeigt, wenn sie ungültig sind</li> <li>■ Einzelne oder mehrere Zeilen können kopiert, ausgeschnitten und in einem anderen Testfall eingefügt werden</li> <li>■ markierten Zeilen können in einem Testfall verschoben werden</li> </ul>
<b>Test-Bibliothek verwalten</b>	<ul style="list-style-type: none"> <li>■ Projekte können eine eigene DSL (Fachsprache) verwenden (XML-Konfiguration)</li> <li>■ Die möglichen Testschritte zu einzelnen Masken (die sog. Test-Bibliothek) kann über XML je Projekt konfiguriert werden</li> </ul>
<b>Test ausführen</b>	<ul style="list-style-type: none"> <li>■ Tests können ausgeführt werden und es wird angezeigt, ob der Test erfolgreich/nicht erfolgreich war</li> <li>■ Testfalls erhalten ein grünes bzw. rotes Icon, nachdem ein Test ausgeführt wurde (bis zur ersten Ausführung ist es grau)</li> </ul>