ATOSj v1.0
Benutzerhandbuch

Autor: Nicos Tegos

Version: 26.01.2007

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

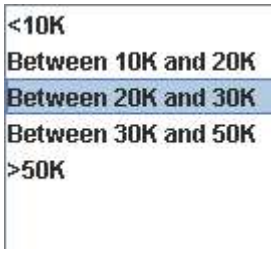
1 ATOSj – Allgemeines Konzept

Das regelmäßige Testen von oberflächenbasierten Systemen ist ein aufwändiger und zeitintensiver Vorgang. Eine Automatisierung dieser Tätigkeit spart Zeit und Kosten und minimiert Fehler. Das Testsystem ATOSj realisiert die Automatisierung des Oberflächentests für Java-Programme. Es unterstützt Programme deren graphische Oberflächen mit Hilfe der Bibliothek Swing oder SWT erstellt wurden. ATOSj ist in der Lage oberflächenbasierte Systeme so zu steuern als wenn ein Benutzer diese bedienen würde. Zusätzlich hat es die Fähigkeit testrelevante Bedingungen zu überprüfen und den gesamten Testvorgang zu protokollieren. Es kann eine dynamische Überdeckungsanalyse durchgeführt werden, um die Vollständigkeit der Testfälle zu überprüfen.

Außerdem unterstützt ATOS den Benutzer bei der Erstellung von Testsequenzen. Testsequenzen enthalten eine Folge von Kommandos, die zur Steuerung von testrelevanten Aktionen dienen. In Testpaketen können Abfolgen von Testsequenzen festgelegt werden.

Damit das Testobjekt gesteuert werden kann, ist es nötig den einzelnen graphischen Elementen eindeutige Bezeichner zuzuordnen. Diese Bezeichner werden in einer speziellen Datei, der so genannten URF-Datei (Uniform Resource Locator), verwaltet.

Tabelle 1.1 zeigt alle Arten von Components, die von ATOSj angesteuert werden können. Die Beispieldarstellungen enthalten keine SWT-Elemente, da diese ein betriebssystemspezifisches Aussehen haben.

Componenttyp	***Beschreibung***	***Graph. Darstellung (Swing)***
`COMPONENT`	Alle Components, die nicht zu den unten genannten zählen.	
`BUTTON`	Druckknopf	
`COMBOBOX`	Texteingabefeld kombiniert mit einer Auswahlliste	
`CHECKBOX`	Anwählen/Abwählen einer Option.	
`EDITBOX`	Texteingabefeld	
`LABEL`	Nicht änderbares Feld zur Textdarstellung.	
`LIST`	Auswahlliste	
`MENU`	Hierarchisch angeordnete Druckknöpfe	

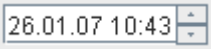
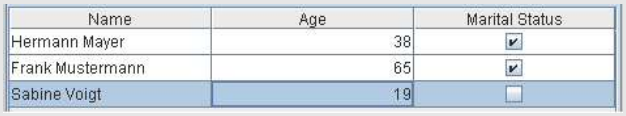
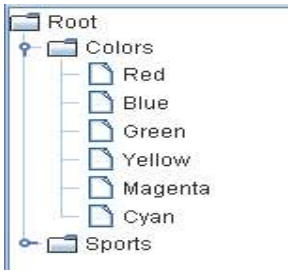
Componenttyp	*Beschreibung*	*Graph. Darstellung (Swing)*
RADIOBUTTON	Anwählen einer Option, innerhalb einer Gruppe darf nur ein Knopf zur Zeit angewählt sein.	
SPINNER	Eingabefeld mit der Möglichkeit zum In- und Dekrementieren.	
TABFOLDER	Karteireiter	
TABLE	Tabelle mit Möglichkeit zur Auswahl und zum Editieren von Einträgen.	
TREE	Hierarchisch angeordnete Einträge mit der Möglichkeit zur Auswahl.	
WINDOW	Alle Fenster und Dialoge.	

Tabelle 1.1 Componenttypen

1.1 Projekte

Wollen Sie ein Programm testen, steht als erster Schritt immer das Erstellen eines Projektes siehe Kapitel 2.2. In einem ATOSj Projekt werden alle testrelevanten Daten verwaltet. Technisch gesehen ist ein Projekt ein Verzeichnis mit Daten welches die folgende Struktur hat:

Verzeichnis	*Dateien*		*Beschreibung*
Projektverzeichnis	Projektdatei Backup	(.apf) (.bak)	Das Projektverzeichnis trägt den Projektnamen und enthält die Projektdatei (<Projektname>.apf) sowie deren Sicherheitskopie (<Projektname>.bak) und alle weiteren Verzeichnisse.
SEQ	Testsequenzen Testpakete	(.seq) (.pak)	Dieses Verzeichnis beinhaltet alle Dateien zu den Testsequenzen und Testpaketen.
URF	URF-Datei	(.urf)	Dieses Verzeichnis beinhaltet die URF-Datei (<Projektname>.urf) des Projektes. Diese enthält alle eindeutigen Bezeichner der Components des Testobjektes.

Verzeichnis	*Dateien*		*Beschreibung*
LOG	Reportdateien	(.pdf)	Dieses Verzeichnis beinhaltet alle Berichte, die bei der Ausführung von Testsequenzen oder Testpaketen generiert wurden. Diese können beispielsweise mit dem AcrobatReader betrachtet werden.
BIN	Testbegleitende Programme		Hier können externe Programme, die während des Tests benötigt werden, abgelegt werden. Diese können über das Kommando LAUNCH gestartet werden.
ENV	Umgebungsdateien		Hier können alle Dateien untergebracht werden, die benötigt werden, um das Testobjekt vor einem Testlauf in einen definierten Ausgangszustand zu bringen. Die Dateien können dann über das Kommando COPY in das gewünschte Verzeichnis kopiert werden.
REF	Referenzdateien		Hier können alle Dateien für eventuelle Dateivergleiche abgelegt werden. Ein Dateivergleich ist in ATOSj nicht implementiert und muss über ein externes Programm durchgeführt werden.
EXT	Konfiguration Bibliothek	custom.lst ext.jar	Hier können die Dateien zur Erweiterung der Unterstützung von Components abgelegt werden. Die genaue Erläuterung dieses Verfahrens erfolgt in Kapitel 1.6.
COV	XML-Datenbasis HTML-Report	coverage.xml coverage.html	Hier befindet sich die Datenbasis zur Überdeckungsanalyse und die Reportdatei, die die Ergebnisse visualisiert. Die genaue Erläuterung des Verfahrens erfolgt in Kapitel 1.5.

Tabelle 1.2 Ordnerstruktur

1.2 Testsequenzen

1.2.1 Kommandos

Eine Testsequenz besteht aus einer Folge von Kommandos, die in einer eigens für ATOSj entwickelten Skriptsprache – HTS (High Level Testscript) - abgefasst sind. Es gibt drei Hauptkategorien von Kommandos:

1. Aktionskommandos
2. Testkommandos
3. weitere Kommandos

Aktionskommandos

Aktionskommandos dienen zur Simulation von Nutzeraktionen auf der GUI. Sie werden durch das HTS-Kommando `ACTION` definiert. Abhängig vom anzusteuernden Component sind verschiedene semantische Aktionen möglich. So kann beispielsweise ein Textfeld editiert werden:

```
ACTION, "Kunde", EDITBOX, "name", EDIT, "Nicos"
```

Das Beispielkommando fügt den Text „Nicos“ in das Textfeld mit dem Namen „name“ im Fenster mit dem Titel „Kunde“ an. Lesen Sie Hinweise zur Benamung von Components im Kapitel 1.4.1.

Testkommandos

Testkommandos dienen zum Abgleich von Soll- und Istzuständen von Components. Sie werden durch das HTS-Kommandos `TEST` definiert. Schlägt ein Test fehl wird ein entsprechenden Fehler gemeldet. Abhängig vom zu testenden Component sind verschiedene Tests möglich. So kann beispielsweise für einen Radiobutton überprüft werden, ob dieser angewählt ist:

```
TEST, "Kunde", CHECKBOX, "male", CHECKSTATE, TRUE
```

Das Beispielkommando überprüft, ob der Radiobutton mit dem Namen „male“ im Fenster mit dem Titel „Kunde“ angewählt ist. Lesen Sie Hinweise zur Benamung von Components im Kapitel 1.4.1.

Weitere Kommandos

Es gibt noch einige weitere Kommandos, die für das Durchführen von Tests nützlich sind z.B. Kommandos zum Starten von externen Programmen oder zum Kopieren von Dateien. Eine vollständige Liste aller Kommandos und deren Bedeutung ist der HTS-Syntax zu entnehmen [HTS06].

1.2.2 Erstellen einer Testsequenz

Die Kommandos einer Testsequenz können per Hand eingegeben werden, da dieses Verfahren sehr zeitaufwändig ist, bietet ATOSj die Möglichkeit des Capturings von Kommandos. Beim Capturing werden alle Aktionen des Nutzers auf dem Testobjekt aufgezeichnet und in HTS-Kommandos umwandelt. Es können die Kommandos `ACTION` und `TEST` automatisch erstellt werden. Das genaue Vorgehen ist in Kapitel 2.6.1 beschrieben.

Allgemeine Hinweise zum Aufbau einer Testsequenz

Sie können durchaus mehrere `START` Kommandos in einer Testsequenz verwenden, beachten Sie dabei, dass alle folgenden Kommandos sich nur auf das zuletzt gestartete Testobjekt beziehen. Beachten Sie weiterhin, dass es guter Stil ist, das Testobjekt am Ende einer Testsequenz regulär zu beenden. Reguläres Beenden heisst, dass das Testobjekt durch eine geeignete Programmfunktion beendet wird, z.B. durch Schließen des Hauptfensters. Verwenden Sie das Kommando `CLEANUP` zur Nachbereitung einer Testsequenz.

1.2.3 Ausführen einer Testsequenz

Ist eine Testsequenz spezifiziert und fehlerfrei, kann sie zur Ausführung gebracht werden.

Eine Testsequenz ist fehlerfrei, wenn alle enthaltenen Kommandos der HTS-Syntax entsprechen und alle in den Kommandos verwendeten ComponentIDs auch in der URF-Datei enthalten sind.

Der Interpreter der Skriptsprache HTS führt alle Kommandos aus, Fehler werden dem Nutzer mitgeteilt. Die Ausführung kann in einer Datei dokumentiert werden. Dem Nutzer bleibt die Auswertung der Ergebnisse der Ausführung überlassen. Er muss überprüfen, ob gemeldete Fehler tatsächlich auf Fehler im Testobjekt zurückzuführen sind oder ob sie durch Fehler in der Testsequenz verursacht wurden. Das genaue Vorgehen zum Ausführen einer Testsequenz ist in Kapitel 2.7 beschrieben.

1.3 Testpakete

Testpakete beinhalten eine beliebige Abfolge von Testsequenzen, welche nacheinander ausgeführt werden. Ein Testpaket unterstützt die Modularisierung eines Testprojektes und ermöglicht Wiederverwendbarkeit von Testsequenzen. Testpakete sollen zur Gliederung eingesetzt werden. Ein Testpaket kann beispielsweise alle Testsequenzen zu einem bestimmten Anwendungsfall, wie er in den Entwicklungsdokumenten des Testobjektes beschrieben wird, enthalten.

1.3.1 Ausführen eines Testpakets

Ist eine Testpaket spezifiziert und fehlerfrei kann es zur Ausführung gebracht werden.

Eine Testpaket ist fehlerfrei, wenn alle enthaltenen Testsequenzen fehlerfrei sind.

Wie bei einer Testsequenz kann auch die Ausführung eines Testpakets in einer Datei dokumentiert werden. Das genaue Vorgehen zum Ausführen eines Testpakets ist in Kapitel 2.12 beschrieben.

1.4 URF-Datei

Die URF-Datei ist ein wichtiger Bestandteil eines ATOSj-Projektes. Sie beinhaltet eindeutige Bezeichner zu den Components des Testobjektes. Dieser Bezeichner wird ComponentID genannt und ist dreigliedrig:

1. Fenstertitel – Der Titel des Fensters, in dem sich das Component befindet oder das HTS-Schlüsselwort `MAIN`, welches ein Platzhalter für den Titel des Hauptfensters des Testobjektes ist.
2. Componenttyp – Der Typ des Components, die möglichen Typen sind in Tabelle 1.1 beschrieben.
3. Name – Der Name des Components, dieser wird über eine bestimmte Strategie ermittelt, siehe Tabelle 1.3.

Bestimmte HTS-Kommandos, wie z.B. `ACTION` oder `TEST`, beinhalten eine ComponentID, um die Components des Testobjekts zur Laufzeit zu referenzieren. Alle verwendeten ComponentIDs müssen in der URF-Datei enthalten sein, wenn nicht ist das Kommando ungültig. Dieses Vorgehen verhindert, dass versehentlich falsche ComponentIDs bei der Erstellung von Kommandos verwendet werden. Desweiteren bietet ATOSj die Möglichkeit Änderungen an vorhanden ComponentIDs vorzunehmen, die dann in allen Kommandos übernommen werden. Das genaue Vorgehen zum Erstellen, Ändern und Löschen von ComponentIDs erfahren Sie in Kapitel 2.13.

Die Referenzierung zur Laufzeit erfolgt, in dem zuerst nach einem Fenster mit dem angegebenen Titel gesucht wird. Ist ein solches Fenster gefunden, werden alle Components im Fenster untersucht, ob sie dem angegeben Typ entsprechen und den angegeben Namen tragen. Existieren mehrere Fenster mit dem gleichen Titel, werden alle nacheinander durchsucht. Kann das Component nicht gefunden werden meldet ATOSj einen Fehler.

1.4.1 Benamung von Components

Um bei der Testausführung ein Component anhand seiner ID zu finden, muss ATOSj die Namen der Components des Testobjektes ermitteln. Dies geschieht zur Laufzeit des Testobjektes nach einer genau definierten Strategie. Diese soll möglichst eindeutige und plakative Namen für ein Component generieren. Des Weiteren soll sie auch die Arbeit mit Testobjekten ermöglichen, deren Quelltext nicht zur Verfügung steht. Das verwendete Vorgehensmuster ist in der nachfolgenden Tabelle beschrieben. Die Möglichkeiten der Benamung werden der Reihe nach durchgegangen bis ein nicht leerer Name gefunden wird.

Componenttyp	***SWT***	***SWING***

`COMPONENT`
`EDITBOX`
`COMBOBOX`
`LIST`
`TABLE`
`TREE`
`TABFOLDER`
	1. Der Name des Components. Festgelegt durch den Programmierer und abrufbar über die Funktion `getData(String key)` aus der Klasse `org.eclipse.swt.widgets.Widget`, wobei key den Wert "ATOSJ_COMPONENT_NAME_KEY" erhält.
	2. Die Grenzen des Components relativ zu seinem Elterncomponent. Die Grenzen werden definiert durch die linke obere Ecke des Components sowie dessen Höhe und Breite in Pixeln.
	3. Der Rückgabewert der Funktion `toString()` des Components.
		1. Der Name des Components. Festgelegt durch den Programmierer und abrufbar über die Funktion `getName()` aus der Klasse `java.awt.Component`.
		2. Der Text eines, diesem Component zugewiesenen, Labels, abrufbar über die Funktion `getClientProperty(Object property)` aus der Klasse `javax.swing.JComponent`, wobei `property` den Wert "labeledBy" erhält.
		3. Die Grenzen des Components relativ zu seinem Fenster. Die Grenzen werden definiert durch die linke obere Ecke des Components sowie dessen Höhe und Breite in Pixeln.

`LABEL`
`RADIOBUTTON`
`CHECKBOX`
`BUTTON`
	1. Der Name des Components. Festgelegt durch den Programmierer und abrufbar über die Funktion `getData(String key)` aus der Klasse `org.eclipse.swt.widgets.Widget`, wobei key den Wert "ATOSJ_COMPONENT_NAME_KEY" erhält.
	2. Der Text des Components, wie er zur Darstellung gesetzt wurde.
	3. Die Grenzen des Components relativ zu seinem Elterncomponent. Die Grenzen werden definiert durch die linke obere Ecke des Components sowie dessen Höhe und Breite in Pixeln.
	4. Der Rückgabewert der Funktion `toString()` des Components.
		1. Der Name des Components. Festgelegt durch den Programmierer und abrufbar über die Funktion `getName()` aus der Klasse `java.awt.Component`.
		2. Der Text des Components, wie er zur Darstellung gesetzt wurde.
		3. Der Text eines, diesem Component zugewiesenen, Labels, abrufbar über die Funktion `getClientProperty(Object property)` aus der Klasse `javax.swing.JComponent`, wobei `property` den Wert "labeledBy" erhält.
		4. Die Grenzen des Components relativ zu seinem Fenster. Die Grenzen werden definiert durch die linke obere Ecke des Components sowie dessen Höhe und Breite in Pixeln.

Componenttyp	*SWT*	*SWING*
`MENU`	1. Der Name des Menüs. Festgelegt durch den Programmierer und abrufbar über die Funktion `getData(String key)` aus der Klasse `org.eclipse.swt.widgets.Widget`, wobei key den Wert "ATOSJ_COMPONENT_NAME_KEY" erhält. 2. Der Rückgabewert der Funktion toString() des Menüs.	1. Der Name des Menüs. Festgelegt durch den Programmierer und abrufbar über die Funktion `getName()` aus der Klasse `java.awt.Component`. 2. Der Text des Menüs, wie er zur Darstellung gesetzt wurde.
`WINDOW`	1. Der Name des Components. Festgelegt durch den Programmierer und abrufbar über die Funktion `getData(String key)` aus der Klasse `org.eclipse.swt.widgets.Widget`, wobei key den Wert "ATOSJ_COMPONENT_NAME_KEY" erhält. 2. Der Titel des Fensters.	1. Der Titel des Fensters.

Tabelle 1.3 Benamungsstrategie

ATOSj versucht zuerst immer einen vom Programmierer gesetzten Namen zu finden. Steht Ihnen der Quelltext des Testobjektes zur Verfügung, sollten Sie von dieser Möglichkeit Gebrauch machen und die Components selbst benamen.

Manuelle Benamung - SWT

Die Benamung für SWT Components führen Sie mit der Funktion `setData(Object key, Object value)` aus der Klasse `org.eclipse.widgets.Widget` durch, wobei `key` den Wert "ATOSJ_COMPONENT_NAME_KEY" erhält und `value` den von Ihnen vergebenen Namen. Beispiel:

```
...
Button addBtn = new Button(parent, SWT.PUSH);
addBtn.setText("Add");
addBtn.setData("ATOSJ_COMPONENT_NAME_KEY", "add");
...
```

Im Beispiel wird dem Knopf `addBtn` der Name „add“ zugewiesen.

Manuelle Benamung - Swing

Die Benamung für Swing Components führen Sie mit der Funktion `setName(String name)` aus der Klasse `java.awt.Component` durch, wobei `name` den von Ihnen vergebenen Namen erhält. Beispiel:

```
...
JButton addBtn = new JButton("Add");
addBtn.setName("add");
...
```

Im Beispiel wird dem Knopf `addBtn` der Name „add“ zugewiesen.

Achten Sie darauf, dass in Verbindung mit dem Fenstertitel und dem Componenttyp eindeutige IDs entstehen. Sollten zwei Components dieselbe ID haben, wird eines der beiden willkürlich gewählt.

1.5 Überdeckungsanalyse

ATOSj ermöglicht dem Tester die Vollständigkeit seiner Testfälle mit einer Überdeckungsanalyse zu überprüfen. Für den Oberflächentest wurden 3 neue Überdeckungsmaße definiert, die in ihrer Granularität dem Testgebiet angemessen sind. Ähnlich der Zweigüberdeckung aus dem Bereich des White-Box-Tests wurde für den Oberflächentest die Methodenüberdeckung definiert. Sie gibt den Prozentsatz aller Methoden einer Klasse an, die während eines oder mehrerer Testläufe mind. einmal ausgeführt wurden. Eine Klasse gilt als vollständig überdeckt, wenn 100% ihrer Methoden ausgeführt wurden. Hierarchisch darauf aufbauend wurde ein Maß zur Paket- und zur Systemüberdeckung definiert. Die einzelnen Maßzahlen werden wie folgt berechnet:

$$Methodenüberdeckung = \frac{Anzahl\ durchlaufene\ Funktionen\ pro\ Klasse}{Anzahl\ Funktionen\ pro\ Klasse\ insgesamt} \cdot 100\,\%$$

$$Paketüberdeckung = \frac{Anzahl\ überdeckte\ Klassen}{Anzahl\ Klassen\ insgesamt} \cdot 100\,\%$$

$$Systemüberdeckung = \frac{Anzahl\ überdeckte\ Pakete}{Anzahl\ Pakete\ insgesamt} \cdot 100\,\%$$

Um die einzelnen Maßzahlen berechnen zu können wird im Ordner `COV` eine XML-Datenbank angelegt, die Aufrufzähler zu bestimmten Methoden des Testobjekts enthält. Es werden alle Methoden vermerkt, die in den Klassen eines oder mehrere Pakete des Testobjekts enthalten sind. Die zu untersuchende Pakete werden vom Tester angegeben. Die Aufrufzähler werden nach jedem Testlauf aktualisiert. Danach können die Maßzahlen in einer übersichtlich formatierten HTML-Datei betrachtet werden. Das genaue Vorgehen wird in Kapitel 2.14 beschrieben.

1.6 Erweiterung von ATOSj (für Fortgeschrittene)

Sollten Ihnen, die von ATOSj unterstützten Standardelemente, siehe Tabelle 1.1, nicht genügen, können Sie ATOSj auch um Custom-Components erweitern. Ein Custom-Component ist ein Component, dass für einen ganz bestimmten Zweck erstellt wurde, der durch die Standard-Components nicht erfüllt wird. Als Beispiel soll uns ein fiktives Component zur graphischen Eingabe eines Datums dienen, genannt Kalender-Component implementiert in der Klasse `CalendarComponent`. Im Folgenden wird erläutert, welche Schritte der Tester/Programmierer zu unternehmen hat, um das Kalender-Component in ATOSj einzubinden. Ein ausgiebiges Studium der API Dokumentation von ATOSj ist erforderlich.

1. Erstellen einer Wrapperklasse für die eigentliche GUI-Klasse
2. Bekanntgabe des neuen Components an ATOSj
3. Verwendung in Kommandos

1.6.1 Erstellen einer Wrapperklasse

Im ersten Schritt, müssen Sie zur Ihrem Custom-Component eine Wrapperklasse erstellen. Dazu müssen Sie ein neues Java Projekt erstellen, welches die Bibliotheken des Testobjektes und die Bibliothek atosj.jar verwendet, setzen Sie den Classpath entsprechend.

- Erstellen Sie eine neue Klasse, in unserem Beispiel soll diese `CalendarWrapper` heißen.

> *Beachten Sie, dass ihr Custom-Component direkt von einer der Klassen aus der jeweiligen GUI-Bibliothek abgeleitet sein muss, beispielsweise javax.swing.JComponent.*

  - Swing – Leiten Sie die Klasse direkt von `atosj.component.swing.SwingComponent` oder von einer Subklasse von `atosj.swing.SwingComponent` ab.
  - SWT – Leiten Sie die Klasse direkt von `atosj.component.swt.SWTComponent` oder von einer Subklasse von `atosj.swt.SWTComponent` ab.
- Reimplentieren Sie die Funktion `setProperty(String property, String value)` aus der Basisklasse von `CalendarWrapper`. Diese hat die Aufgabe durch Simulation von Nutzereingaben bestimmte Werte zu setzen.
  - Definieren Sie gültige Eigenschaften (engl. property), in unserem Beispiel ist das die Eigenschaft „date“. Für alle ungültigen Eigenschaften muss eine `atosj.exception.ComponentException` geworfen werden.
  - Definieren Sie gültige Werte (engl. value) für jede Eigenschaft, in unserem Beispiel sind gültige Werte für die Eigenschaft „date“ Zeichenketten im Format DD.MM.JJJJ. Für alle ungültigen Werte muss eine `atosj.exception.ComponentException` geworfen werden.
  - Implementieren Sie das Setzen eines Wertes. Das Setzen eines Wertes soll möglichst nur durch die Simulation von Nutzereingaben erreicht werden, also durch Tastendruck oder Mausklick. Dazu gibt es in der Basisklasse von `CalendarWrapper` die Funktionen `pressKey` und `click`. Diese sollten möglichst verwendet werden. Es steht frei für Standard-Components innerhalb des Custom-Components die vordefinierten Wrapperklassen zu nutzen. So kann beispielsweise die Klasse `atosj.component.swing.SwingTextComponent` genutzt werden, um den Text für eine Instanz der Klasse `javax.swing.JTextField` zu setzen.
- Reimplentieren Sie die Funktion `getProperty(String property)` aus der Basisklasse von CalendarWrapper. Diese hat die Aufgabe durch aktuelle Werte/Zustände des Kalender-Components zu ermitteln.
  - Definieren Sie gültige Eigenschaften (engl. property), in unserem Beispiel ist das die Eigenschaft „date“. Für alle ungültigen Eigenschaften muss eine `atosj.exception.ComponentException` geworfen werden.
  - Implementieren Sie das Auslesen eines Wertes. In unserem Fall soll das aktuell gesetzte Datum „date“ ausgelesen werden. Geben Sie den ausgelesenen Wert als `java.lang.String` zurück, beispielsweise „24.12.06“.
- Erstellen Sie eine weitere Klasse `CalendarWrapperCommandCreator`. Dies kann eine private innere Klasse der Klasse CalendarWrapper sein. Sie soll das Aufzeichnen von Nutzeraktionen auf dem Kalender-Component ermöglichen.
  - Leiten Sie die Klasse `CalendarWrapperCommandCreator` von der CommandCreator

Klasse, der Basisklasse von `CalendarWrapper` ab.

- Reimplementieren Sie die Funktion `getTestCommands()`. Diese liefert als Rückgabewert einen Vector, welcher alle `TEST`-Kommandos für Kalender-Component enthält. Das Kalender-Component kann beispielsweise auf Sichtbarkeit überprüft werden oder es kann ein Sollwertvergleich des aktuellen Datums durchgeführt werden. Nutzen Sie zur Generierung der Testkommandos die Funkion `createTestCommand(String state, Vector stateParameters, Vector subitemPath)`.
- Überwachen Sie alle vom Nutzer durchgeführten Änderungen am Kalender-Component und generieren Sie hierzu ein `ACTION`-Kommando. Beispielsweise könnten Änderungen des Datums im Kalender-Component ATOSj bekanntgegeben werden. Nutzen Sie zum Erstellen und Bekanntgeben des `ACTION`-Kommandos an ATOSj die Funktion `publishActionCommand(String actionType, Vector actionParameters, Vector subitemPath)`.

- Reimplementieren Sie die Funktion `initCommandCreator(IHTSCommandReceiver receiver)`. Diese hat die Aufgabe den von Ihnen erstellten `CalendarWrapperCommandCreator` zu initialisieren. Liefern Sie also eine neue Instanz von `CalendarWrapperCommandCreator` als Rückgabewert.

Damit haben Sie die Implementation der Wrapperklasse beendet. Eine vollständige Implementation eines Beispiels für eine Wrapperklasse finden sie in den Quellen von ATOSj im Paket `atosj.sampleapp.custom`, betrachten sie dort die Klasse `DateWidgetWrapper`.

1.6.2 Bekanntgabe des neuen Components an ATOSj

Nachdem Sie die Wrapperklasse für das Custom-Component erstellt haben, müssen Sie diese nun ATOSj bekanntgeben, damit Sie in Kommandos verwendet werden kann. Hierzu erstellen Sie im Ordner `EXT` ihres ATOSj Projektes die Datei custom.lst.

Fügen Sie für jede Wrapperklasse, die Sie erstellt haben, eine Zeile im folgenden Format in die Datei ein:

`COMPONENT, <TYPENAME>, <WRAPPERKLASSE>, <CUSTOMKLASSE>`

`<TYPENAME>`	Gibt den Typnamen für das Custom-Component, dieser wird dann in den HTS Kommandos verwendet. Der Typname ist eine Abstraktion, welche innerhalb der verwendeten GUI-Bibliothek eindeutig sein muss.
`<WRAPPERKLASSE>`	Ist der vollständig qualifizierte Name der Wrapperklasse für die Klasse des Custom-Components.
`<CUSTOMKLASSE>`	Ist der vollständig qualifizierte Name der Klasse des Custom-Components, diese muss von der Basisklasse der GUI-Bibliothek abgeleitet sein.

Für unser Kalender-Component muss die Zeile also lauten:

`COMPONENT, CALENDAR, CalendarComponentWrapper, CalendarComponent`

Nachdem Sie die Datei custom.lst erstellt haben, fügen sie alle Wrapperklassen einer jar-Datei mit dem Namen ext.jar hinzu und legen Sie diese im `EXT` Verzeichnis ihres ATOSj Projektes ab.

1.6.3 Verwendung des neuen Components

Das Custom-Component kann, nachdem die Schritte 1 und 2 ausgeführt wurden, nun auch in den Kommandos von ATOSj verwendet werden. Es folgen zwei Beispiele zum Setzen und Abtesten von Eigenschaften des Kalender-Components:

```
ACTION, "Kunde", CALENDAR, "geb.dat.", PROPERTY, "date",
"24.07.1981"
TEST, "Kunde", CALENDAR, "geb.dat.", PROPERTY, "date",
"24.07.1981"
```

1.6.4 Datenflussdiagramm

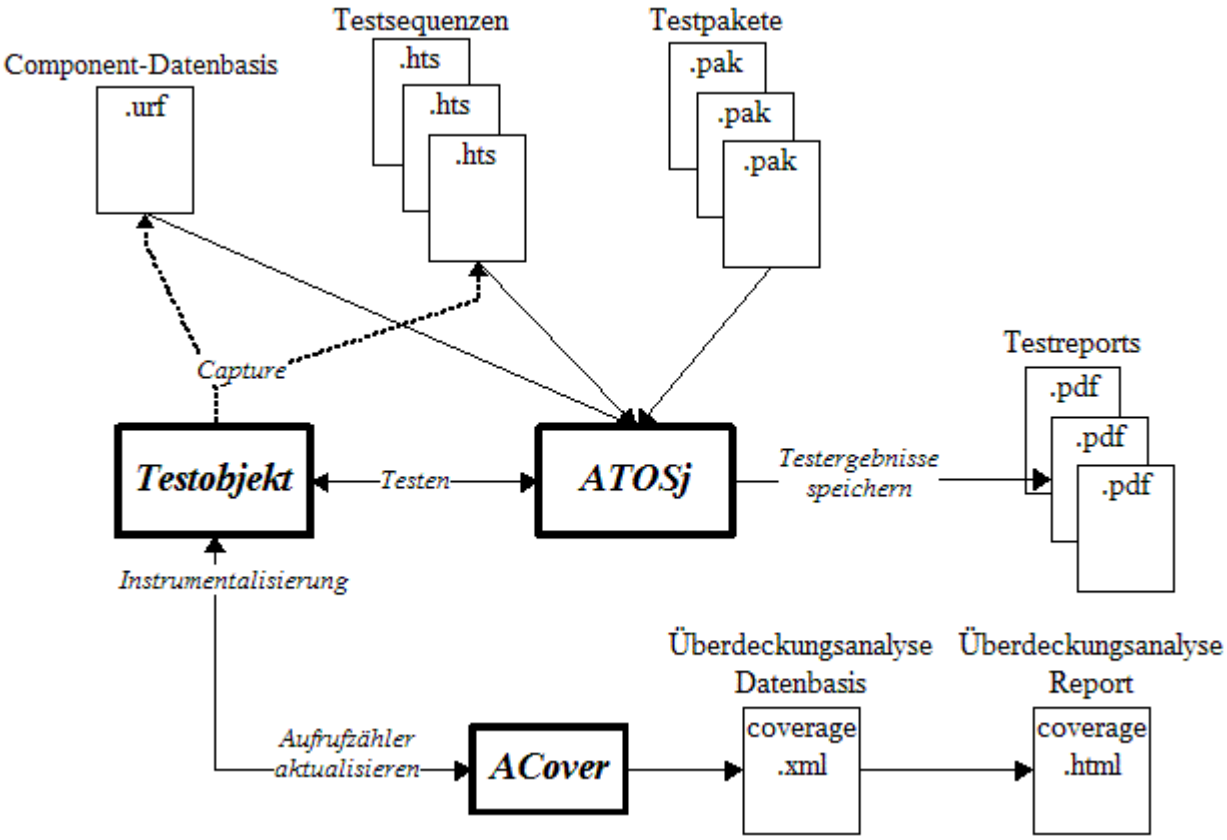


Abbildung 1.1Testkomponenten und Dateien

2 Umgang mit der Benutzeroberfläche

Dieses Kapitel stellt alle wichtigen Funktionen der Nutzeroberfläche von ATOSj dar. Abbildung 2.1 zeigt das Hauptfenster von ATOSj in der Projektansicht. Auf der linken Seite befindet sich der Navigator. Er enthält alle eine Übersicht über alle Testsequenzen, Testpakete und die URF-Datei des aktuellen Projektes. Auf der rechten Seite befinden sich die Editoren, der aktuell bearbeiteten Ressourcen.

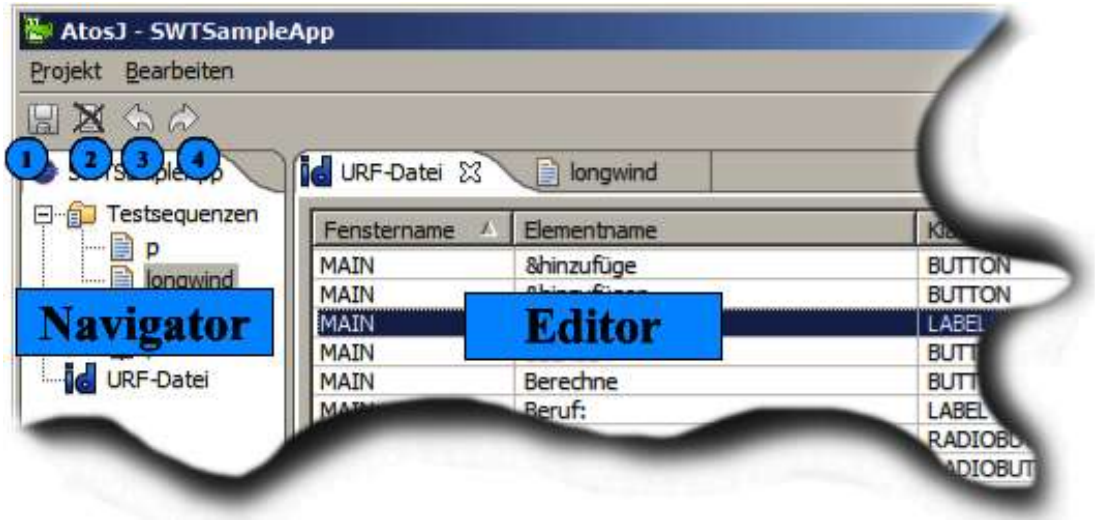


Abbildung 2.1 ATOSj Hauptfenster

Abhängig vom aktuellen Status einer Ressource werden im Navigator und auch in den Editoren verschiedene Symbole angezeigt, die in der nachfolgenden Tabelle näher beschrieben werden sollen.

Symbol	***Auftreten***	***Bedeutung***
*	Bei allen Ressourcen	Die Ressource wurde geändert und ist noch gespeichert.
	Im Navigator bei Testsequenzen oder Testpaketen	• Warnung die Testsequenz enthält kein `START` Kommando. • Warnung das Testpaket enthält Testsequenzen mit Warnungen.
	Im Navigator bei Testsequenzen oder Testpaketen	• Die Testsequenz ist fehlerhaft. • Ein Testpaket enthält fehlerhafte Testsequenzen.
	Im Paketeditor	• Die Testsequenz enthält Warnungen. Für eine Beschreibung führen Sie einen Doppelklick auf die Spalte mit dem Symbol aus.
	Im Testsequenz- oder Paketeditor	• Das Kommando ist fehlerhaft auf Grund inkorrekter Syntax oder weil es eine ComponentID referenziert, die nicht in der URF-Datei enthalten ist. Um eine Fehlerbeschreibung zu erhalten, führen Sie einen Doppelklick auf die Spalte mit dem Symbol aus. • Die Testsequenz ist fehlerhaft. Um eine Fehlerbeschreibung zu erhalten, führen Sie einen Doppelklick auf die Spalte mit dem Symbol aus.

2.1 Allgemeine Editorfunktionen

Dieser Abschnitt beschreibt Funktionen, die für alle Editoren gleichermaßen ausgeführt werden können.

2.1.1 Editor öffnen

Das Öffnen eines Editors funktioniert für alle Ressourcen, d.h. für Testsequenzen, Testpakete und die URF-Datei, nach dem gleichen Prinzip. Soll der Editor geöffnet werden, klicken Sie die Ressource doppelt im *Navigator* an.

2.1.2 Änderungen Rückgängig machen

Sie können an allen Ressourcen (Testsequenzen, Testpakete, URF-Datei) Änderungen vornehmen. Sollten Sie feststellen, dass Sie einen Fehler gemacht haben, bietet Ihnen ATOSj die Möglichkeit die 5 letzten Änderungen rückgängig zu machen.

- Vorbedingung ist, dass Sie vorher Änderungen am aktuellen Editor vorgenommen haben.
- 3 Möglichkeiten:
  1. Klicken Sie auf den Knopf *Rückgängig machen*  in der Knopfleiste des Hauptfensters.
  2. Wählen den Punkt *Rückgängig machen* im Menü *Bearbeiten* des Hauptfensters.
  3. Drücken Sie die Tasten Strg und 'Z' gleichzeitig.
- Ihre letzte Änderung im aktuellen Editor wird rückgängig gemacht und der vorhergehende Status wiederhergestellt.

2.1.3 Änderungen wiederherstellen

Sie können an allen Ressourcen (Testsequenzen, Testpakete, URF-Datei) Änderungen vornehmen. Diese können rückgängig gemacht werden, siehe Kapitel 2.1.2. Dual dazu können Sie rückgängig gemachte Änderungen wiederherstellen.

- Vorbedingung ist, dass Sie vorher Änderungen am aktuellen Editor rückgängig gemacht haben.
- 3 Möglichkeiten:
  1. Klicken Sie auf den Knopf *Wiederherstellen*  in der Knopfleiste des Hauptfensters.
  2. Wählen den Punkt *Wiederherstellen* im Menü *Bearbeiten* des Hauptfensters.
  3. Drücken Sie die Tasten Strg und 'Y' gleichzeitig.
- Ihre letzte Änderung wird rückgängig gemacht und der vorhergehende Status wiederhergestellt.

2.1.4 Änderungen speichern

Sie können an allen Ressourcen (Testsequenzen, Testpakete, URF-Datei) Änderungen vornehmen. Diese Änderungen können Sie explizit festschreiben.

- Vorbedingung ist, dass Sie vorher Änderungen am aktuellen Editor vorgenommen haben.
- 3 Möglichkeiten:
  1. Klicken Sie auf den Knopf Änderungen *Speichern*  in der Knopfleiste des Hauptfensters.

2. Wählen den Punkt *Speichern* im Menü *Bearbeiten* des Hauptfensters.
3. Drücken Sie die Tasten Strg und 'S' gleichzeitig.

- Ihre letzte Änderung am aktuellen Editor wird rückgängig gemacht und der vorhergehende Status wiederhergestellt.

2.1.5 Änderungen verwerfen

Sie können an allen Ressourcen (Testsequenzen, Testpakete, URF-Datei) Änderungen vornehmen. Sollten Sie feststellen, dass Ihre Änderungen fehlerhaft waren, bietet Ihnen ATOSj die Möglichkeit die Änderungen zu verwerfen, d.h. den letzten gespeicherten Zustand wiederherzustellen.

- Vorbedingung ist, dass Sie vorher Änderungen am aktuellen Editor vorgenommen haben.
- 3 Möglichkeiten:

1. Klicken Sie auf den Knopf *Verwerfen*  in der Knopfleiste des Hauptfensters.
2. Wählen den Punkt *Verwerfen* im Menü *Bearbeiten* des Hauptfensters.
3. Drücken Sie die Tasten Strg und 'W' gleichzeitig.

- Ihre letzte Änderung wird rückgängig gemacht und der vorhergehende Status wiederhergestellt.

2.2 Projekt erstellen

Dieser Abschnitt zeigt wie Sie ein neues Testprojekt anlegen können.

- 2 Möglichkeiten:
  1. Im Menü *Datei* den Punkt *Neu...* wählen.
  2. Auf dem Startbildschirm den Knopf *Projekt erstellen* wählen.
- Der Dialog *Projekt erstellen* erscheint.

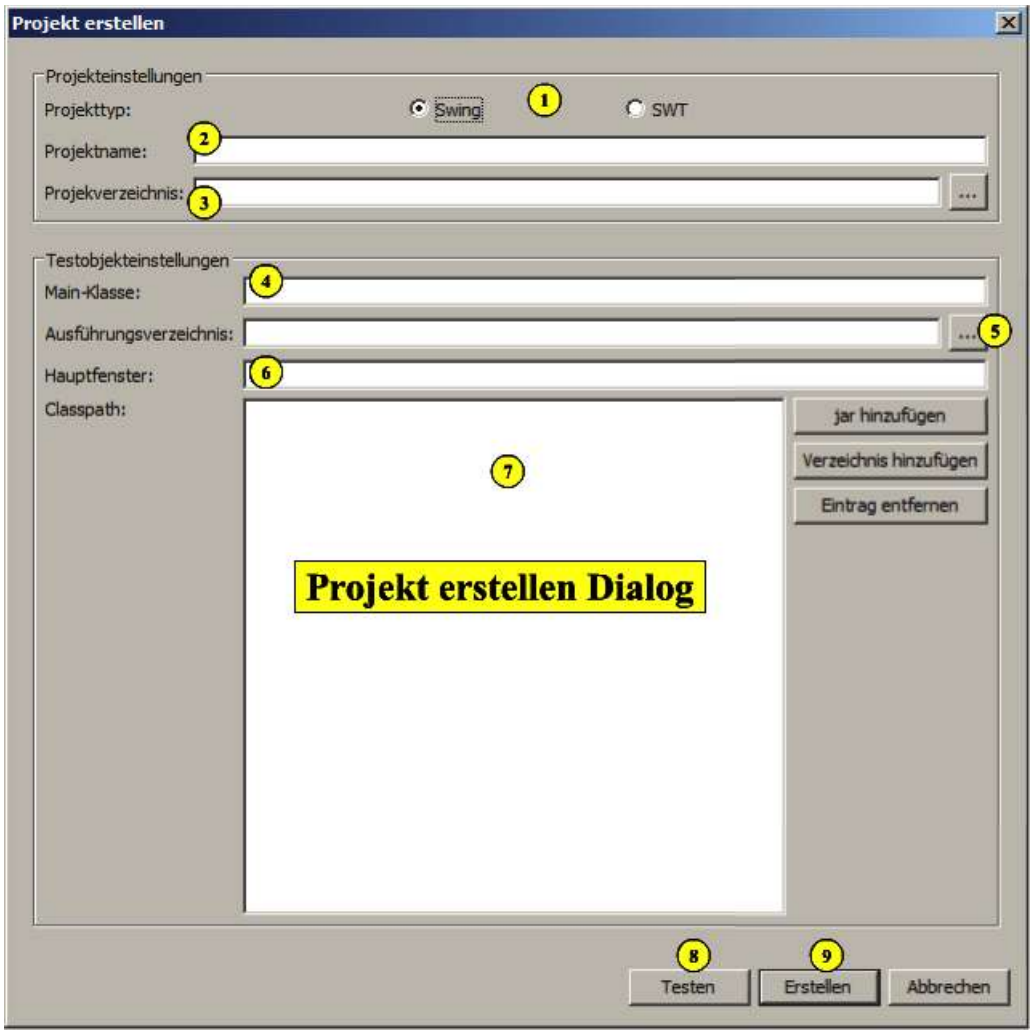


Abbildung 2.2 Projekt erstellen Dialog

- Wählen Sie den Projekttyp (1), 2 Möglichkeiten:
  4. Swing für Testobjekte, deren Oberfläche mit der Bibliothek Swing erstellt wurde.
  5. SWT für Testobjekte, deren Oberfläche mit der Bibliothek SWT erstellt wurde.
- Geben Sie den Namen (2) des neuen Projektes an. Dieser ist gleichzeitig auch der Name des Projektverzeichnisses und der Projektdatei, die im Projektverzeichnis angelegt wird.
- Wählen Sie den Ort (Verzeichnis) (3) für das Projekt aus. Das Projektverzeichnis wird dann in dem angegebenen Verzeichnis erstellt.

- Geben Sie die Main-Klasse (4) des Testobjekts an. Hier muss der vollqualifiziert Name der Klasse angegeben werden, die die main-Funktion zum Starten des Testobjektes enthält.
- Geben Sie das Verzeichnis (5) an, in dem das Testobjekt ausgeführt werden soll. Das Testobjekt wird während aller Tests in dem hier angegebenen Verzeichnis ausgeführt. Achten Sie darauf, dass alle eventuell verwendeten nativen Bibliotheken, z.B. Dateien mit der Endung „dll“ in Windows Systemen, die vom Testobjekt zur Laufzeit benötigt werden entweder im Ausführungsverzeichnis oder im Suchpfad des Betriebssystems für Laufzeitbibliotheken liegen.
- Geben Sie den Titel des Hauptfensters (6) des Testobjekts an. Der Titel des Hauptfensters wird von ATOSj benötigt, um den Start des Testobjektes zu verifizieren. Erscheint das Hauptfenster nicht binnen 20 Sekunden nach dem Start des Testobjektes wird ein Fehler gemeldet.
- Geben Sie den Klassenpfad (7) für das Testobjekt an. Hier müssen alle Verzeichnisse und Bibliotheken (jar-Dateien) angegeben werden, die für das Starten des Testobjektes benötigt werden. Werden hier Einträge vergessen führt dies mit hoher Wahrscheinlichkeit zur Ausnahme `java.lang.ClassNotFoundException` zur Laufzeit des Testobjektes.
- Drücken Sie den Knopf Testen (8), um das Testobjekt probeweise starten. Schlägt der Start fehl überprüfen Sie die Angaben zum Testobjekt (Main-Klasse, Hauptfenster, Ausführungsverzeichnis, Klassenpfad).
- Drücken Sie den Knopf *Erstellen* (9).
- Das Projekt wird erstellt und die Projektansicht öffnet sich.

2.3 Testsequenz erstellen

Dieser Abschnitt zeigt wie Sie einem Projekt eine neue Testsequenz hinzufügen können.

- Im *Navigator* mit der rechten Maustaste auf den Ordner *Testsequenzen* klicken. Es erscheint ein Kontextmenü, wählen Sie daraus den Punkt *neue Testsequenz*.
- Es erscheint der Dialog *Testsequenz erstellen.*
  - Geben Sie den Namen der Testsequenz ein und drücken Sie *OK*.

    Der Name der Testsequenz muss ein gültiger Dateiname sein und darf noch nicht von einer anderen Testsequenz verwendet werden. Bei der Arbeit auf unterschiedlichen Betriebssystemen ist darauf zu achten, dass die Namen der Testsequenzen portabel zwischen den verschiedenen Betriebsystemen sind.

- Die Testsequenz wird erstellt. Sie erscheint als Eintrag im *Navigator* und der Editor der Testsequenz wird geöffnet. Die Testsequenz ist noch leer.

2.4 Testsequenz löschen

Dieser Abschnitt zeigt wie Sie eine Testsequenz aus einem Projekt entfernen können.

- Im *Navigator* mit der rechten Maustaste auf die zu löschende Testsequenz klicken. Es erscheint ein Kontextmenü, wählen Sie daraus den Punkt *löschen*.
- Es erscheint der Dialog *Testsequenz löschen.*

  - Drücken Sie *Ja*.
- Die Testsequenz wird gelöscht. Die Testsequenz wird auch aus allen Paketen gelöscht, in denen sie verwendet wird.

> *Beachten Sie das die Testsequenz aus allen Paketen, in denen sie verwendet wird, entfernt wird. Überprüfen Sie, ob die Testpakete hiernach noch den gewünschten Zweck erfüllen.*
>
> *Beachten Sie weiterhin, dass das Löschen einer Testsequenz nicht rückgängig gemacht werden kann.*

2.5 Testsequenz umbenennen

Dieser Abschnitt zeigt, wie Sie den Namen einer Testsequenz ändern können.

- Im *Navigator* mit der rechten Maustaste auf die gewünschte Testsequenz klicken. Es erscheint ein Kontextmenü, wählen Sie daraus den Punkt *umbenennen*.
- Es erscheint der Dialog *Testsequenz umbenennen.*
  - Geben Sie den neuen Namen der Testsequenz ein und drücken Sie *OK*. Der Name darf noch nicht von einer anderen Testsequenz verwendet sein.
- Die Testsequenz wird umbenannt.

2.6 Testsequenz bearbeiten

Dieser Abschnitt zeigt, wie Sie eine bereits vorhandene Testsequenz mit Hilfe des *Testsequenzeditors* ändern können.

- Führen Sie einen Doppelklick auf die gewünschte Testsequenz im *Navigator* aus.
- Der *Testsequenzeditor* öffnet sich.

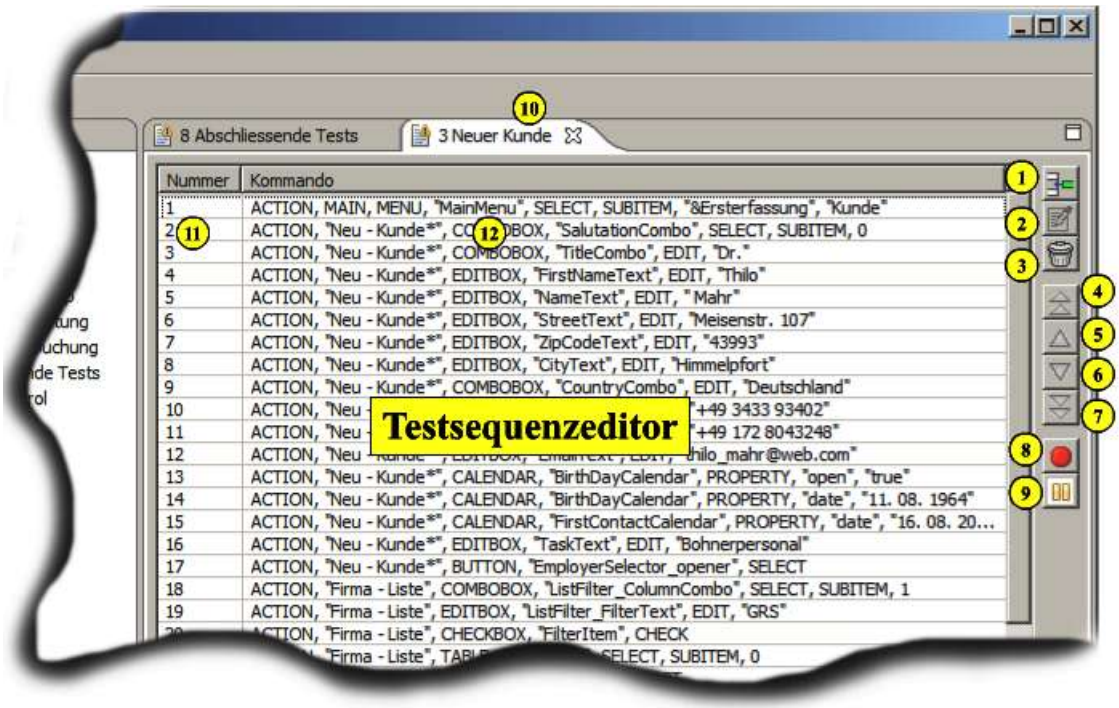


Abbildung 2.3 Testsequenzeditor

2.6.1 Kommando erstellen (manuell und automatisch)

Dieser Abschnitt zeigt, wie Sie einer Testsequenz ein neues Kommando hinzufügen können.

Manuell für Anfänger

Ein Kommando wird über einen graphischen Dialog eingegeben, der sehr intuitiv ist und nur geringe Kenntnisse der verwendeten Skriptsprache voraussetzt [HTS06].

- Es bieten sich 3 Möglichkeiten:
  1. Klicken Sie auf den Knopf *Kommando hinzufügen* (1).
  2. Rechtsklick auf die Kommandotabelle (12) und wählen Sie aus dem Kontextmenü den Punkt *Kommando hinzufügen*.
  3. Klicken sie auf die Kommandotabelle (12) und drücken Sie die Tasten Strg und 'N' gleichzeitig.
- Der Dialog *Kommando hinzufügen* erscheint.
  - Geben Sie die Parameter für das Kommando ein.
  - Wählen Sie die Einfügeposition des Kommandos aus.
  - Drücken Sie auf den Knopf *hinzufügen*.
- Das neue Kommando erscheint an der angegebenen Position (11) in der Kommandotabelle.

Manuell für Fortgeschrittene

Im Gegensatz zur Erstellung eines Kommandos in der Anfängervariante wird ein Kommando nicht über graphische Elemente sondern in einer rein textuellen Notation spezifiziert. Darum sollten Sie gut mit der Syntax der Skriptsprache HTS [HTS06] vertraut sein, wenn Sie diesen Weg zur Erstellung eines Kommandos wählen.

- Es bieten sich 2 Möglichkeiten:
  1. Rechtsklick auf die Kommandotabelle und wählen Sie aus dem Kontextmenü den Punkt *Kommando hinzufügen (adv.)*.
  2. Klicken sie auf die Kommandotabelle und drücken Sie die Tasten Alt und 'N' gleichzeitig.
- Der Dialog *Kommando hinzufügen* erscheint.
  - Geben Sie das Kommando in seiner vollständigen textuellen Notation ein.
  - Wählen Sie die Einfügeposition des Kommandos aus.
  - Drücken Sie auf den Knopf *hinzufügen*.
- Das neue Kommando erscheint an der angegebenen Position in der Kommandotabelle.

Automatisch per Capture

Sie können nicht nur manuell Kommandos eingeben sondern auch, um das Erstellen von Testsequenzen zu beschleunigen, direkt Ihre Aktionen auf dem Testobjekt in HTS-Kommandos umwandeln lassen.

- Drücken Sie den Knopf *Aufzeichnung starten* (8).
- Das Testobjekt wird gestartet.
- Zu allen Aktionen, die Sie auf der Oberfläche des Testobjektes ausführen, wird, sofern existent, ein entsprechendes `ACTION` Kommando am Ende der Testsequenz eingefügt.
- Wollen Sie für bestimmte Aktionen die Aufzeichnung verhindern drücken Sie den Knopf *Pause* (9). Drücken Sie den Knopf nochmals, um die Aufzeichnung wieder zu aktivieren.
- Es können nicht nur Aktionen sondern auch `TEST` Kommandos automatisch generiert werden.
  - Drücken Sie über einem Component des Testobjektes die Taste Strg und die rechte Maustaste.
  - Der Dialog *Test-Kommandos erstellen* erscheint.
  - Wählen Sie aus der Liste alle Zustände des Components, für die Sie einen Sollwertvergleich durchführen wollen.
  - Drücken Sie *OK*.
- Die `TEST` Kommandos werden am Ende der Testsequenz eingefügt.
- Beenden Sie das Testobjekt oder Drücken sie nochmals den Knopf *Aufzeichnung starten* (8).

Allgemeine Hinweise

Es ist dringend anzuraten die generierte Testsequenz nach dem Capture-Vorgang zu überprüfen.

- Wird bei einer SWT-Anwendung ein Fenster programmatisch geschlossen, z.B. durch das Drücken eines Knopfes, werden zwei Kommandos generiert. Das Kommando welches explizit die Aktionsart `CLOSE` enthält, muss gelöscht werden. Dies betrifft auch das Kalenderfenster in der Seminarorganisation (der Name des Fensters lautet Shell{}).
- Nach dem Editieren eines Textfeldes sollte die Eingabe explizit mit Enter oder Tab abgeschlossen werden, sonst kann es passieren, dass die Reihenfolge der generierten Kommandos vertauscht wird.
- Kalender in der Seminarorganisation: Bei Änderung der Jahreszahl mit den Knöpfen des Spinners wird für jede Änderung ein Kommando generiert. Hier sollte evtl. nur eine Änderung vorgenommen und das Kommando nachträglich per Hand editiert werden.

2.6.2 Kommando editieren

Dieser Abschnitt zeigt wie sie ein bereits vorhandenes Kommando ändern können.

Für Anfänger

Das Kommando wird über denselben graphischen Dialog editiert, der auch zum Erstellen eines Kommandos verwendet wird. Dieser ist sehr intuitiv ist und setzt nur geringe Kenntnisse der verwendeten Skriptsprache [HTS06] voraus.

- Wählen Sie genau ein Kommando aus der Tabelle aus. Das Kommando muss gültig sein!
- 3 Möglichkeiten:

  3. Klicken Sie auf den Knopf *Kommando bearbeiten* (2).

4. Rechtsklick auf die Kommandotabelle (12) und wählen Sie aus dem Kontextmenü den Punkt *Kommando bearbeiten*.
5. Drücken Sie die Tasten Strg und 'E' gleichzeitig.

- Der Dialog *Kommando bearbeiten* erscheint.
  - Geben Sie Ihre Änderungen für das Kommando ein.
  - Drücken Sie auf den Knopf *hinzufügen*.
- Das Kommando wird entsprechend verändert.

Für Fortgeschrittene

Im Gegensatz zum Editieren eines Kommandos in der Anfängervariante, wird hier ein Kommando in seiner rein textuellen Notation bearbeitet. Darum sollten Sie gut mit der Syntax der Skriptsprache HTS [HTS06] vertraut sein, wenn Sie diesen Weg zum Editieren eines Kommandos wählen.

- Wählen sie genau einen Eintrag aus der Kommandotabelle aus.
- 2 Möglichkeiten:
  1. Rechtsklick auf die Kommandotabelle (12) und wählen Sie aus dem Kontextmenü der Kommandotabelle den Punkt *Kommando bearbeiten (adv.)*.
  2. Drücken Sie die Tasten Alt und 'E' gleichzeitig.
- Der Dialog *Kommando bearbeiten* erscheint.
  - Geben Sie Ihre Änderungen für das Kommando in textueller Notation ein.
  - Drücken Sie auf den Knopf *hinzufügen*.
- Das Kommando wird entsprechend verändert.

2.6.3 Kommando(s) entfernen

Dieser Abschnitt zeigt, wie Sie bereits vorhandene Kommandos aus einer Testsequenz löschen können.

- Wählen Sie einen oder mehrere Einträge aus der Kommandotabelle.
- 3 Möglichkeiten:
  1. Klicken Sie auf den Knopf *Kommandos entfernen* (3).
  2. Rechtsklick auf die Kommandotabelle (12) und wählen Sie aus dem Kontextmenü der Kommandotabelle den Punkt *Kommandos entfernen*.
  3. Drücken Sie die Taste Entf.
- Die ausgewählten Kommandos werden entfernt.

2.6.4 Kommando(s) deaktivieren

Dieser Abschnitt zeigt wie Sie Kommandos deaktivieren können. Deaktivierte Kommandos werden bei der Testdurchführung nicht ausgeführt. Will man ein Kommando nur kurzzeitig nicht zur

Ausführung bringen, so kann man es deaktivieren, anstatt es zu löschen.

- Wählen Sie einen oder mehrere Einträge aus der Kommandotabelle.
- 2 Möglichkeiten:
  1. Rechtsklick auf die Kommandotabelle (12) und wählen Sie aus dem Kontextmenü der Kommandotabelle den Punkt *Kommandos deaktivieren*.
  2. Drücken Sie die Tasten Strg und 'D' gleichzeitig.
- Die ausgewählten Kommandos werden deaktiviert.

2.6.5 Kommando(s) aktivieren

Dieser Abschnitt zeigt wie Sie deaktivierte Kommandos wieder aktivieren können.

- Wählen Sie einen oder mehrere Einträge aus der Kommandotabelle.
- 2 Möglichkeiten:
  1. Rechtsklick auf die Kommandotabelle (12) und wählen Sie aus dem Kontextmenü der Kommandotabelle den Punkt *Kommandos aktivieren*.
  2. Drücken Sie die Tasten Strg und 'A' gleichzeitig.
- Die ausgewählten Kommandos werden aktiviert.

2.6.6 Kommando verschieben

Dieser Abschnitt zeigt, wie man die Position eines Kommandos innerhalb einer Testsequenz verändern kann.

- Wählen Sie einen Eintrag aus der Kommandotabelle (12).
- Sie haben die Möglichkeit ein Kommando:
  - um eine Position nach oben (5)
  - um eine Position nach unten (6)
  - an den Anfang der Testsequenz (4)
  - das Ende der Testsequenz zu verschieben (7)
- 2 Möglichkeiten:
  1. Rechtsklick auf die Kommandotabelle (12) und wählen Sie den entsprechenden Menüpunkt aus dem Kontextmenü aus.
  2. Drücken Sie den entsprechenden Knopf in der Knopfleiste rechts von der Kommandotabelle.
- Das ausgewählte Kommando wird verschoben.

2.7 Testsequenz ausführen

Dieser Abschnitt zeigt, wie Sie eine Testsequenz ausführen könne. Die Testsequenz muss fehlerfrei sein.

- Im *Navigator* mit der rechten Maustaste auf die gewünschte Testsequenz klicken. Es erscheint ein Kontextmenü, wählen Sie daraus den Punkt *starten*.
- Es erscheint der Dialog *Testsequenz ausführen*.
  - Setzen die das Häkchen bei der Option *Bei einem Fehler anhalten*, so wird die Testsequenz beim ersten Fehler, der auftritt beendet. Diese Option ist besonders dann sinnvoll, wenn das fehlschlagen eines Kommandos Folgefehler produzieren kann.
  - Setzen Sie das Häkchen bei der Option *Protokolldatei anlegen*, so wird nach der Beendigung der Testsequenz eine Protokolldatei im Ordner `LOG` angelegt. Die Protokolldatei hat das Format <Testsequenzname> DD.MM.JJJJ HH.MM.SS.pdf
  - Geben im Feld *Verzögerung in ms* eine Zeitdauer in Millisekunden an. Diese Dauer wird nach der Ausführung eines jeden Kommandos gewartet. Diese Option ist sinnvoll, um den Testlauf genau mitverfolgen zu können.
  - Drücken Sie den Knopf *Start*, um den Testvorgang zu starten.
  - Nach der Beendigung der Testsequenz können Sie den Knopf *Report* drücken, um sich den Testbericht anzeigen zu lassen. Der Knopf ist nur aktiviert, wenn sie die Option *Protokolldatei anlegen* angewählt haben.
  - Schließen Sie den Dialog *Testsequenz starten*, in dem Sie den Knopf *Abbrechen* drücken.
- Sie sind wieder in der Projektansicht von ATOSj.

2.8 Testpaket erstellen

Dieser Abschnitt zeigt wie Sie einem Projekt ein neues Testpaket hinzufügen können. Ein Testpaket kann mehrere Testsequenzen enthalten.

- Im *Navigator* mit der rechten Maustaste auf den Ordner *Testpakete* klicken. Es erscheint ein Kontextmenü, wählen Sie daraus den Punkt *neues Testpaket*.
- Es erscheint der Dialog *Testpaket erstellen*.
  - Geben Sie den Namen des Testpakets ein und drücken Sie *OK*.

    Der Name des Testpakets muss ein gültiger Dateiname sein und darf noch nicht von einem anderen Testpaket verwendet werden. Bei der Arbeit auf unterschiedlichen Betriebssystemen ist darauf zu achten, dass die Namen der Testpakete portabel zwischen den verschiedenen Betriebsystemen sind.
- Das Testpaket wird erstellt. Es erscheint als Eintrag im Navigator und der entsprechende Editor wird geöffnet. Das Testpaket ist noch leer.

2.9 Testpaket löschen

Dieser Abschnitt zeigt wie Sie ein Testpaket aus einem Projekt entfernen können.

- Im *Navigator* mit der rechten Maustaste auf das zu löschende Testpaket klicken. Es erscheint ein Kontextmenü, wählen Sie daraus den Punkt *löschen*.
- Es erscheint der Dialog *Testpaket löschen*.
  - Drücken Sie *Ja*.
- Das Testpaket wird gelöscht.

Beachten Sie, dass das Löschen eines Testpakets nicht rückgängig gemacht werden kann.

2.10 Testpaket umbenennen

Dieser Abschnitt zeigt, wie Sie den Namen eines Testpakets ändern können.

- Im *Navigator* mit der rechten Maustaste auf das gewünschte Testpaket klicken. Es erscheint ein Kontextmenü, wählen Sie daraus den Punkt *umbenennen*.
- Es erscheint der Dialog *Testpaket umbenennen*.
  - Geben Sie den neuen Namen des Testpakets ein und drücken Sie *OK*. Der Name darf noch nicht von einem anderen Testpaket verwendet sein.
- Das Testpaket wird umbenannt.

2.11 Testpaket bearbeiten

Dieser Abschnitt zeigt, wie Sie ein bereits vorhandenes Testpaket mit Hilfe des *Paketeditors* ändern können.

- Führen Sie einen Doppelklick auf das gewünschte Testpaket im *Navigator* aus.
- Der *Paketeditor* öffnet sich.

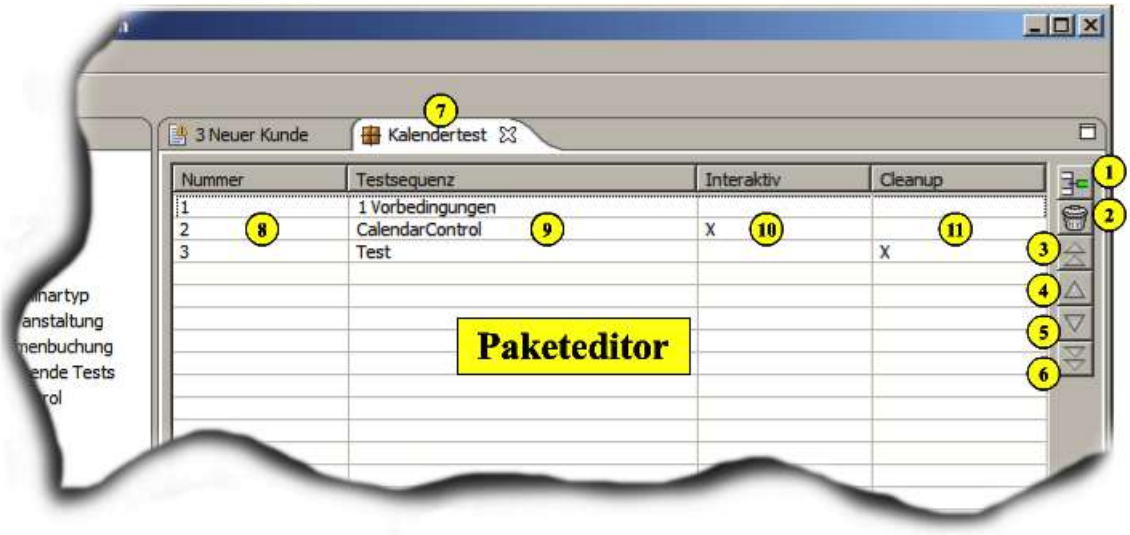


Abbildung 2.4 Paketeditor

2.11.1 Testsequenz(en) hinzufügen

Dieser Abschnitt zeigt, wie Sie einem Testpaket ein Testsequenz hinzufügen können.

- 3 Möglichkeiten:
  1. Klicken Sie auf den Knopf *Testsequenzen hinzufügen* (1).
  2. Rechtsklick auf die Tabelle (9) und wählen Sie aus dem Kontextmenü den Punkt *Testsequenzen hinzufügen.*
  3. Klicken sie auf die Tabelle (9) und drücken Sie die Tasten Strg und 'N' gleichzeitig.
- Der Dialog *Testsequenzen hinzufügen* erscheint.
  - Setzen Sie ein Häkchen bei allen Testsequenzen, die Sie dem Testpaket hinzufügen möchten. Es ist möglich eine Testsequenz mehrfach zu einem Testpaket hinzuzufügen.
  - Wählen Sie die Einfügeposition der Testsequenz aus.
  - Drücken Sie auf den Knopf *hinzufügen*.
- Die neuen Testsequenzen erscheinen an der angegebenen Position (8) in der Tabelle.

2.11.2 Testsequenz editieren

Sie haben die Möglichkeit aus dem *Paketeditors* direkt den *Tesequenzeditor* zu öffnen. Führen Sie hierzu einen Doppelklick auf die Spalte *Testsequenz* (9) der gewünschten Testsequenz aus.

2.11.3 Testsequenz(en) entfernen

Dieser Abschnitt zeigt wie Sie eine oder mehrere Testsequenzen aus einem Paket löschen können.

- Wählen Sie einen oder mehrere Einträge aus der Tabelle der Testsequenzen.
- 3 Möglichkeiten:
  1. Klicken Sie auf den Knopf *Testsequenzen entfernen* (2).
  2. Rechtsklick auf die Tabelle (9) und wählen Sie aus dem Kontextmenü den Punkt *Testsequenzen entfernen*.
  3. Drücken Sie die Taste Entf.
- Die ausgewählten Testsequenzen werden entfernt.

2.11.4 Testsequenz verschieben

Dieser Abschnitt zeigt, wie man die Position einer Testsequenz innerhalb eines Testpakets verändern kann.

- Wählen Sie einen Eintrag aus der Tabelle der Testsequenzen.
- Sie haben die Möglichkeit eine Testsequenz:
  - um eine Position nach oben (4)
  - um eine Position nach unten (5)
  - an den Anfang des Testpakets (3)
  - an das Ende des Testpakets (6) zu verschieben
- 2 Möglichkeiten:
  1. Rechtsklick auf die Tabelle (9) und wählen Sie den entsprechenden Menüpunkt aus dem Kontextmenü aus.
  2. Drücken Sie den entsprechenden Knopf, in der Knopfleiste rechts von der Kommandotabelle.
- Die ausgewählte Testsequenz wird verschoben.

2.12 Testpaket ausführen

Dieser Abschnitt zeigt, wie Sie ein Testpaket ausführen können. Das Testpaket muss fehlerfrei sein.

- Im *Navigator* mit der rechten Maustaste auf das gewünschte Testpaket klicken. Es erscheint ein Kontextmenü, wählen Sie daraus den Punkt *starten*.
- Es erscheint der Dialog *Testpaket starten*.
  - Setzen die das Häkchen bei der Option *Testsequenz bei einem Fehler anhalten*, so wird die aktuell ausgeführte Testsequenz beim ersten Fehler, der auftritt beendet. Diese Option ist besonders dann sinnvoll, wenn das Fehlschlagen eines Kommandos Folgefehler produzieren kann.
  - Setzen Sie das Häkchen bei der Option *Protokolldatei anlegen*, so wird nach der Beendigung des Testpakets eine Protokolldatei im Ordner `LOG` angelegt. Die Protokolldatei hat das Format

<Testpaketname> DD.MM.JJJJ HH.MM.SS.pdf

- Geben im Feld *Verzögerung in ms* eine Zeitdauer in Millisekunden an. Diese Dauer wird nach der Ausführung eines jeden Kommandos gewartet. Diese Option ist sinnvoll, um den Testlauf genau mitverfolgen zu können.
- Setzen Sie das Häkchen bei der Option *Testpaket bei einer fehlerhaften Testsequenz beenden*, so wird die Ausführung des gesamten Testpaketes nach der fehlerhaften Ausführung einer Testsequenz beendet.
- Drücken Sie den Knopf *Start*, um den Testvorgang zu starten.
- Nach der Beendigung des Testpakets können Sie den Knopf *Report* drücken, um sich den Testbericht anzeigen zu lassen. Der Knopf ist nur aktiviert, wenn sie die Option *Protokolldatei anlegen* angewählt haben.
- Schließen Sie den Dialog *Testpaket starten*, in dem Sie den Knopf *Abbrechen* drücken.

- Sie sind wieder in der Projektansicht von ATOSj.

2.13 URF-Datei bearbeiten

Dieser Abschnitt zeigt, wie Sie die URF-Datei mit Hilfe des *URF-Editors* ändern können.

- Führen Sie einen Doppelklick auf die URF-Datei im *Navigator* aus.
- Der *URF-Editor* öffnet sich.

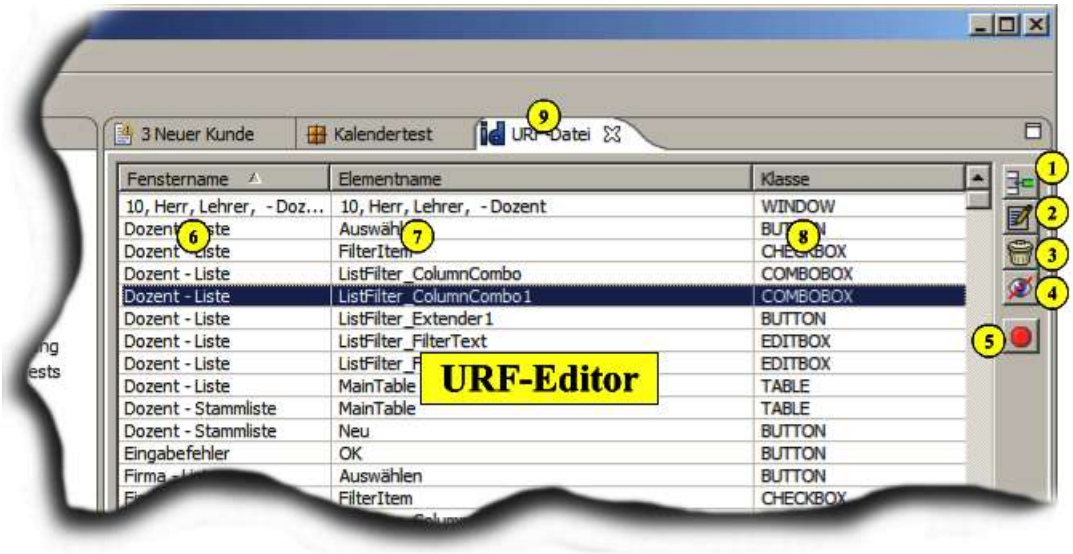


Abbildung 2.5 URF-Editor

2.13.1 Neue ComponentIDs hinzufügen

Dieser Abschnitt zeigt, wie Sie der URF-Datei neue ComponentIDs hinzufügen können.

Manuell

- Es bieten sich 3 Möglichkeiten:

1. Klicken Sie auf den Knopf *ID hinzufügen* (1).

2. Rechtsklick auf die ID-Tabelle (7) und wählen Sie aus dem Kontextmenü den Punkt *ID hinzufügen*.
3. Klicken sie auf die ID-Tabelle und (7) drücken Sie die Tasten Strg und 'N' gleichzeitig.

- Der Dialog *ID erstellen* erscheint.
  - Geben Sie den Fenstertitel ein.
  - Geben Sie den Elementnamen ein.
  - Wählen Sie den Elementtyp aus der Liste aus.
  - Drücken Sie den Knopf *OK*.

  Beachten Sie, dass bei Components vom Typ `WINDOW` der Fenstertitel und der Elementname gleich lauten müssen.

- Die neue ID wird der URF-Datei hinzugefügt, wenn sie noch nicht enthalten ist.

Automatisch per Aufzeichnung

ComponentIDs können der URF-Datei nicht nur manuell sondern auch automatisch hinzugefügt werden. Hierzu wird das Testobjekt gestartet und beim Öffnen eines Fensters werden zu allen enthaltenen Components die entsprechenden IDs generiert.

- 2 Möglichkeiten:
  1. Drücken Sie den Knopf *Aufzeichnung starten* (5).
  2. Rechtsklick auf die ID-Tabelle (7) und wählen Sie den Menüpunkt *Aufzeichnung starten*.
- Das Testobjekt wird gestartet.
- Öffnen Sie alle Fenster zu deren Components neue IDs generiert werden sollen.
- Die neuen IDs werden der URF-Datei hinzugefügt, vorhandene IDs bleiben erhalten.

  Beachten Sie, dass nur zu den Components IDs generiert werden, die auch tatsächlich direkt nach dem Öffnen eines Fensters erstellt waren. Components die im weiteren Verlauf dynamisch erzeugt werden, werden nicht berücksichtigt.

- Schließen Sie das Testobjekt oder Drücken Sie den Knopf *Aufzeichnung beenden* (5) bzw. wählen Sie den entsprechenden Menüpunkt aus dem Kontextmenü der ID-Tabelle.

2.13.2 ComponentID editieren

Dieser Abschnitt beschreibt, wie Sie eine vorhanden ComponentID bearbeiten können. Die Änderung wird bei allen Kommandos übernommen, die diese ID verwenden.

- Wählen Sie genau eine ID aus der Tabelle (7) aus.
- 3 Möglichkeiten:
  3. Klicken Sie auf den Knopf *ID bearbeiten* (2).

4. Rechtsklick auf die ID-Tabelle (7) und wählen Sie aus dem Kontextmenü den Punkt *ID bearbeiten*.
5. Drücken Sie die Tasten Strg und 'E' gleichzeitig.

- Der Dialog *ID bearbeiten* erscheint.
  - Geben Sie Ihre Änderungen für die ID ein.
  - Drücken Sie auf den Knopf *OK*.
- Die ID wird geändert, die URF-Datei wird gespeichert und die Änderung wird bei allen Kommandos übernommen, die diese ID verwenden.

2.13.3 ComponentID(s) entfernen

Dieser Abschnitt zeigt wie Sie eine oder mehrere IDs aus URF-Datei löschen können.

- Wählen Sie einen oder mehrere Einträge aus der Tabelle (7) aus.
- 3 Möglichkeiten:
  1. Klicken Sie auf den Knopf *IDs entfernen* (3).
  2. Rechtsklick auf die Tabelle (7) und wählen Sie aus dem Kontextmenü den Punkt *IDs entfernen*.
  3. Drücken Sie die Taste Entf.
- Die ausgewählten IDs werden entfernt.

> *Beachten Sie, dass das Entfernen von Ids, die noch von Kommandos verwendet werden, diese Kommandos ungültig macht.*

2.14 Überdeckungsanalyse

Dieser Abschnitt zeigt, wie sie eine Überdeckungsanalyse durchführen können. Das Ergebnis der Überdeckungsanalyse wird in einer HTML-Datei visualisiert.

- Im Menü *Projekt* den Punkt *Überdeckungsanalyse* wählen.
- Der Dialog *Überdeckungsanalyse* öffnet sich:

Abbildung 2.6 Dialog Überdeckungsanalyse

- Spezifizieren Sie die Java-Pakete des Testobjekts, die einer Überdeckungsanalyse unterzogen werden sollen (2). Es genügt nur einen Teil des Paketnamens anzugeben. Alle Pakete, die mit diesem Namensteil beginnen, werden automatisch in die Untersuchung einbezogen. Der Namensteil muss mindestens ein Zeichen lang sein. Nach der Eingabe erscheint er in der Liste (1).
- Wollen Sie einen Namensteil wieder aus der Liste löschen, wählen Sie den entsprechenden Eintrag aus und drücken Sie den Knopf *ausgewählte Pakete entfernen* (3).
- Wählen Sie die Option *Überdeckungsanalyse bei jedem Testlauf durchführen* (4), um alle folgenden Testläufe in die Überdeckungsanalyse einzubeziehen. Wenn diese Option nicht aktiviert ist, wird keine Überdeckungsanalyse durchgeführt.
- Drücken Sie den Knopf *Generieren* (5), um die Datenbasis für die Überdeckungsanalyse der angegebenen Pakete zu erstellen. Die Datenbasis befindet sich danach im Verzeichnis `COV` des Projektes unter dem Namen "coverage.xml".

> *Beachten Sie, dass das Erstellen der Datenbasis die vorhergende Datenbasis überschreibt. Alle Aufrufzähler werden auf 0 gesetzt. Die Überdeckungsanalyse kann erst erfolgen, wenn Sie eine Datenbasis erstellt haben.*

- Drücken Sie den Knopf *Beenden* (7), um den Dialog zu Schließen und führen Sie nun eine Testsequenz oder ein Testpaket aus. Danach öffnen Sie erneut den Dialog *Überdeckungsanalyse.*
- Drücken Sie den Knopf *Report* (6), um einen HTML-Report zu erstellen. Er enthält die Maße zur Methoden- und Paketüberdeckung und gibt zu jeder Methode die Anzahl ihrer Aufrufe an. Der HTML-Report befindet sich danach im Verzeichnis `COV` des Projektes unter dem Namen

"coverage.html".

Beachten Sie, dass der HTML-Report nur erstellt wird, wenn auch eine Datenbasis existiert. Beim Erstellen wird der vorhergehende Report überschrieben.

2.14.1 Allgemeine Hinweise

Haben Sie die Datenbasis erstellt, können Sie Testsequenzen und Testpakete ausführen und danach den Report zur Überdeckungsanalyse generieren. Die Datenbasis wird immer dann aktualisiert, wenn das Testobjekt während eines Testlaufs regulär beendet wird. Das heißt das Testobjekt muss durch eine Nutzeraktion beendet werden, die z.B. zum Aufruf der Funktion `System.exit` führt. Beachten Sie, dass das Beenden des Testobjekt-Prozesses durch das HTS-Kommando `CLEANUP` ein reguläres Ende und damit auch die Aktualisierung der Datenbasis verhindert.

Das Aktualisieren der Datenbasis benötigt Zeit. Wollen Sie also die Überdeckungsanalyse verwenden, fügen Sie nach dem Kommando, welches das Testobjekt regulär beenden soll, eine angemessene Wartezeit ein (z.B. `WAIT, 5000`).

Beachten Sie beim Auswerten der Überdeckungsanalyse, dass auch eine 100-prozentige Überdeckung kein Garant dafür ist, dass tatsächlich alle funktionalen Anforderungen getestet wurden. Die Beziehung oder Aufrufreihenfolge bestimmter Programmfunktionen wird bei der Methodenüberdeckung nicht berücksichtigt.

Beachten Sie außerdem, dass bei der Überdeckungsanalyse das Testobjekt zur Laufzeit instrumentiert wird, d.h. in jede Methode der ausgewählten Pakete wird ein Aufruf für das Inkrementieren eines Zähler eingefügt. Dazu wird ein spezieller Klassenlader verwendet, verwendet ihr Programm auch einen selbstdefinierten Klassenlader kann es zu Problemen kommen.

3 Dateiformate

3.1 URF-Datei

Die URF-Datei besteht aus einzelnen Zeilen, die durch '\n' voneinander getrennt sind. Jede Zeile hat das folgenden Format:

```
<Elementname>, <HTS-Typ>, <Fenster>
```

`<Elementname>`	Der Name des Element in "-Zeichen. Der Name entspricht der Benamungsstrategie in Tabelle 1.3.
`<HTS-Typ>`	Der Typ des Components. Die gültigen Typen können Tabelle 1.1 entnommen werden.
`<Fenster>`	Der initiale Titel des Fensters in dem sich das Component befindet. Für das Hauptfenster dient das HTS-Schlüsselwort `MAIN` als Platzhalter für den tatsächlichen Titel. Alle anderen Fenstertitel müssen in "-Zeichen eingeschlossen sein.

Beispiel:

```
"OptionPane.body", COMPONENT, "Doppelclick"
"First Name:", EDITBOX, "MAIN"
```

3.2 APF-Datei

Die APF-Datei speichert alle projektrelevanten Parameter. Projektparameter werden zeilenweise definiert, alle Zeilen sind durch '\n' voneinander getrennt. Die Reihenfolge der Zeilen ist beliebig. Folgende Morpheme werden vereinbart:

`<String>` `String` ist eine Zeichenkette in "-Zeichen. Als Maskierungszeichen wird das Backslash (\) verwendet.

Projektname:

```
PROJECTNAME, <Projektname>
```

`<Projektname>` Der Name des Projektes.

Projekttyp:

```
PROJECTTYPE, <Projekttyp>
```

`<Projekttyp>` Der Typ des, derzeit sind zwei Typen gültig – AWT und SWT.

Ausführungspfad:

```
WORKPATH, <Pfad>
```

`<Pfad>` Der absolute Pfad, des Ordners in dem das Testobjekt ausgeführt werden soll.

Main-Klasse:

```
APPLICATION, <Main-Klasse>
```

`<Main-Klasse>` Der vollqualifizierte Name der Main-Klasse zum Start des Testobjekts.

Fenstername:

```
MAINWINDOW, <Fenstername>
```

`<Fenstername>` Der Name des Hauptfensters des Testobjekts.

Klassenpfad:

`CLASSPATH (, <Pfad>)`

`<Pfad>` Der absolute Pfad einer Bibliothek (jar) oder eines Verzeichnisses, das Java-Klassen für die Ausführung des Testobjekts beinhaltet.

Pakete für die Überdeckungsanalyse:

`COVERPATH (, <Paketname>)`

`<Paketname>` Namensteil eines Pakets des Testobjektes, das in die Überdeckungsanalyse einbezogen werden soll.

Überdeckungsanalyse durchführen:

`COVER, TRUE|FALSE`

`TRUE|FALSE` Ist der Parameter `TRUE`, wird die Überdeckungsanalyse bei der Ausführung einer Testsequenz oder eines Testpakets durchgeführt, sonst nicht.

Geöffnete Ressourcen:

`OPEN (, <Dateiname>)`

`<Dateiname>` Name einer Datei aus dem ATOSj-Projekt, deren Editor beim letzten Beenden des Programms geöffnet war.

Aktive Ressource:

`ACTIVE, <Dateiname>`

`<Dateiname>` Name einer Datei aus dem ATOSj-Projekt, deren Editor beim letzten Beenden des Programms geöffnet UND gerade aktiv war.

Beispiel:

```
PROJECTNAME, "Mein Projekt"
PROJECTTYPE, "AWT"
WORKPATH, "E:\\JavaMyFiles\\workspace\\AtosJ"
APPLICATION, "sampleapp.SampleApp"
MAINWINDOW, "Simple Widgets"
CLASSPATH, "E:\\JavaMyFiles\\workspace\\AtosJ"
COVERPATH, "sampleapp"
COVER, TRUE
OPEN, "Test1.hts", "Test2.hts", "All.pak"
ACTIVE, "Test1.hts"
```

4 Glossar

Component	Allgemeine Bezeichnung für ein Element der graphischen Benutzeroberfläche, wie z.B. ein Druckknopf oder ein Fenster. Bezeichnung in Anlehnung an die Basisklasse aller graphischen Elemente in AWT/Swing `java.awt.Component`.
GUI	Engl. Graphical User Interface, die graphische Benutzeroberfläche eines Programms.
Testobjekt	Das Programm, welches einem Test mit ATOSj unterzogen werden soll.
Wrapperklasse	Eine Java Klasse, die ein Objekt einer anderen Klasse beinhaltet und dessen Funktionalität über eine abgewandelte Schnittstelle teilweise oder auch vollständig zur Verfügung stellt.

5 Literaturverzeichnis

HTS06: Nicos Tegos, Die HTS-Spezifikation, 2006