Eine Anmerkung zur HTS-Sprachspezifikation:

Das WHILE-Kommando wurde hier leider vergessen. Ausreichende Hinweise dazu findet man jedoch auf den S. 3 (whilestruktur) und S. 7 oben (vergleichskörper). Außerdem leitet einen der Kommandoeditor.

Uli Sacklowski, 2009

Die HTS-Spezifikation

Autor: Nicos Tegos

05. Januar 2007

Dieses Dokument enthält die überarbeitete Syntax und Semantik der Skriptsprache HTS. Die Originalversion wurde von Jens Hanisch und Johann Letzel spezifiziert und in dieser Version erweitert und abgeändert. Alle neuen und geänderten Kommandos sind gekennzeichnet. Die überabeitete Version dient ausschließlich zur Spezifikation von Testsequenzen in *ATOSj*. In *ATOS* wird weiterhin die Originalversion verwendet.

Inhaltsverzeichnis

1 Die HTS-Spezifikation

1.1 Die Syntax

Hier wird die Grammatik der geänderten Scriptsprache in EBNF (erweiterte Backus-Naur-Form) vorgestellt. Jede Anweisung wird mit einem Zeilenumbruch (\n) abgeschlossen. Folgende Morpheme werden vereinbart und besonders gekennzeichnet:

`<zahl>`	ist eine beliebige nichtnegative natürliche Zahl
`<+zahl+>`	ist eine beliebige positive natürliche Zahl
`<.float.>`	ist eine beliebige Fließkommazahl, als Dezimalzeichen wird ein Punkt verwendet
`$string$`	ist eine beliebige Zeichenkette in Anführungszeichen, als Maskierungszeichen wird das Backslash verwendet
`$+string+$`	ist eine Zeichenkette ohne Leer- und Anführungszeichen

```
skript              =  {anweisung}
                       CLEANUP
                       {anweisung}
anweisung           =  aktionsschritt | auswertungsschritt |
                       interaktion | dateioperation | loopstruktur |
                       whilestruktur | kommentar | deakt_kommando \n
aktionsschritt      =  warten | aktion
auswertungsschritt  =  lesen | vergleich | testen | fenstersichtbarkeit
interaktion         =  frage | nachricht
dateioperation      =  kopieren | löschen | exitenz | starten
```

Unbedingte Schleife

```
loopstruktur  =  LOOP, <+Wiederholungsanzahl+>
                 (anweisung)
                 ENDLOOP
```

Bedingte Schleife *neu*

```
whilestruktur  =  WHILE, vergleichskörper
                  (anweisung)
                  ENDWHILE
```

Nicht ausführbare Anweisungen

```
kommentar       =  COMMENT, $text$
deakt_kommando  =  DISABLE, anweisung
```

Warten für eine Zeitspanne

```
warten  =  WAIT, <+Millisekunden+>
```

Abfragen der Components *geändert*

```
lesen          =  READ, $Fenstertitel$ | MAIN, typ_lesen, $Variablenname$
typ_lesen      =  component_lesen | custom_lesen | editbox_lesen | label_lesen |
                  radiobutton_lesen | checkbox_lesen | button_lesen | combobox_-
                  lesen | liste_lesen | tabelle_lesen | baum_lesen | menü_lesen |
                  kartei_lesen | fenster_lesen

component_-    =  COMPONENT, $Name$, ENABLESTATE |
lesen             COMPONENT, $Name$, FOCUSSTATE |
                  COMPONENT, $Name$, VISIBLESTATE
custom_lesen   =  $+Custom+$, $Name$, PROPERTY, $Eigenschaft$
editbox_-      =  EDITBOX, $Name$, TEXT |
lesen             EDITBOX, $Name$, NUM |
                  EDITBOX, $Name$, ENABLESTATE |
                  EDITBOX, $Name$, FOCUSSTATE |
                  EDITBOX, $Name$, VISIBLESTATE
label_lesen    =  LABEL, $Name$, TEXT |
                  LABEL, $Name$, NUM |
                  LABEL, $Name$, ENABLESTATE |
                  LABEL, $Name$, FOCUSSTATE |
                  LABEL, $Name$, VISIBLESTATE
radiobutton_-  =  RADIOBUTTON, $Name$, CHECKSTATE |
lesen             RADIOBUTTON, $Name$, TEXT |
                  RADIOBUTTON, $Name$, NUM |
                  RADIOBUTTON, $Name$, ENABLESTATE |
                  RADIOBUTTON, $Name$, FOCUSSTATE |
                  RADIOBUTTON, $Name$, VISIBLESTATE
checkbox_-     =  CHECKBOX, $Name$, CHECKSTATE |
lesen             CHECKBOX, $Name$, TEXT |
                  CHECKBOX, $Name$, NUM |
                  CHECKBOX, $Name$, ENABLESTATE |
                  CHECKBOX, $Name$, FOCUSSTATE |
                  CHECKBOX, $Name$, VISIBLESTATE
button_lesen   =  BUTTON, $Name$, TEXT |
                  BUTTON, $Name$, NUM |
                  BUTTON, $Name$, ENABLESTATE |
                  BUTTON, $Name$, FOCUSSTATE |
                  BUTTON, $Name$, VISIBLESTATE
combobox_-     =  COMBOBOX, $Name$, ITEMCOUNT |
lesen             COMBOBOX, $Name$, TEXT |
                  COMBOBOX, $Name$, NUM |
                  COMBOBOX, $Name$, ENABLESTATE |
                  COMBOBOX, $Name$, FOCUSSTATE |
                  COMBOBOX, $Name$, VISIBLESTATE
```

```
liste_lesen    =  LIST, $Name$, ITEMCOUNT |
                  LIST, $Name$, TEXT, SUBITEM, <+Listenindex+> |
                  LIST, $Name$, ENABLESTATE |
                  LIST, $Name$, FOCUSSTATE |
                  LIST, $Name$, VISIBLESTATE
tabelle_-      =  TABLE, $Name$, TEXT, SUBITEM, <+Zeilenindex+>, <+Spaltenindex> |
lesen             TABLE, $Name$, NUM, SUBITEM, <+Zeilenindex+>, <+Spaltenindex> |
                  TABLE, $Name$, CHECKSTATE, SUBITEM, <+Zeilenindex+>,
                  <+Spaltenindex> |
                  TABLE, $Name$, ITEMCOUNT |
                  TABLE, $Name$, ENABLESTATE |
                  TABLE, $Name$, FOCUSSTATE |
                  TABLE, $Name$, VISIBLESTATE
baum_lesen     =  TREE, $Name$, ENABLESTATE |
                  TREE, $Name$, FOCUSSTATE |
                  TREE, $Name$, VISIBLESTATE
menü_lesen     =  MENU, $Name$, ENABLESTATE, SUBITEM {,$Menüpunkt$} |
                  MENU, $Name$, CHECKSTATE, SUBITEM {,$Menüpunkt$} |
                  MENU, $Name$, ENABLESTATE |
                  MENU, $Name$, FOCUSSTATE |
                  MENU, $Name$, VISIBLESTATE
kartei_lesen   =  TABFOLDER, $Name$, TEXT, SUBITEM, <+Tabindex+> |
                  TABFOLDER, $Name$, ITEMCOUNT |
                  TABFOLDER, $Name$, ENABLESTATE |
                  TABFOLDER, $Name$, FOCUSSTATE |
                  TABFOLDER, $Name$, VISIBLESTATE
fenster_-      =  WINDOW, $Name$, ENABLESTATE |
lesen             WINDOW, $Name$, FOCUSSTATE |
                  WINDOW, $Name$, VISIBLESTATE
```

Abfragen und Sollwertvergleich der Components *geändert*

```
testen         =  TEST, $Fenstertitel$ | MAIN, typ_testen
typ_testen     =  component_testen | custom_testen | editbox_testen | label_testen |
                  radiobutton_testen | checkbox_testen | button_testen | combobox_-
                  testen | liste_testen | tabelle_testen | baum_testen | menü_testen
                  | kartei_testen | fenster_testen

component_-    =  COMPONENT, $Name$, ENABLESTATE, bool_wert |
testen            COMPONENT, $Name$, FOCUSSTATE, bool_wert |
                  COMPONENT, $Name$, VISIBLESTATE, bool_wert
custom_-       =  $+Custom+$, $Name$, PROPERTY, $Eigenschaft$, $Sollwert$
testen
editbox_-      =  EDITBOX, $Name$, TEXT, $Sollwert$ |
testen            EDITBOX, $Name$, NUM <.Sollwert.> [,vergleichsmodus] |
                  EDITBOX, $Name$, ENABLESTATE, bool_wert |
                  EDITBOX, $Name$, FOCUSSTATE, bool_wert |
                  EDITBOX, $Name$, VISIBLESTATE, bool_wert
label_testen   =  LABEL, $Name$, TEXT, $Sollwert$ |
                  LABEL, $Name$, NUM, <.Sollwert.> [,vergleichsmodus] |
                  LABEL, $Name$, ENABLESTATE, bool_wert |
                  LABEL, $Name$, FOCUSSTATE, bool_wert |
                  LABEL, $Name$, VISIBLESTATE, bool_wert
```

```
radiobutton_-  =  RADIOBUTTON, $Name$, CHECKSTATE, bool_wert |
testen            RADIOBUTTON, $Name$, TEXT, $Sollwert$ |
                  RADIOBUTTON, $Name$, NUM, <.Sollwert.> [,vergleichsmodus] |
                  RADIOBUTTON, $Name$, ENABLESTATE, bool_wert |
                  RADIOBUTTON, $Name$, FOCUSSTATE, bool_wert |
                  RADIOBUTTON, $Name$, VISIBLESTATE, bool_wert
checkbox_-     =  CHECKBOX, $Name$, CHECKSTATE, bool_wert |
testen            CHECKBOX, $Name$, TEXT, $Sollwert$ |
                  CHECKBOX, $Name$, NUM, <.Sollwert.> [,vergleichsmodus] |
                  CHECKBOX, $Name$, ENABLESTATE, bool_wert |
                  CHECKBOX, $Name$, FOCUSSTATE, bool_wert |
                  CHECKBOX, $Name$, VISIBLESTATE, bool_wert
button_-       =  BUTTON, $Name$, TEXT, $Sollwert$ |
testen            BUTTON, $Name$, NUM, <.Sollwert.> [,vergleichsmodus] |
                  BUTTON, $Name$, ENABLESTATE, bool_wert |
                  BUTTON, $Name$, FOCUSSTATE, bool_wert |
                  BUTTON, $Name$, VISIBLESTATE, bool_wert
combobox_-     =  COMBOBOX, $Name$, ITEMCOUNT, <+Sollwert+> [,vergleichsmodus2] |
testen            COMBOBOX, $Name$, TEXT, $Sollwert$ |
                  COMBOBOX, $Name$, NUM, <.Sollwert.> [,vergleichsmodus] |
                  COMBOBOX, $Name$, ENABLESTATE, bool_wert |
                  COMBOBOX, $Name$, FOCUSSTATE, bool_wert |
                  COMBOBOX, $Name$, VISIBLESTATE, bool_wert
liste_testen   =  LIST, $Name$, ITEMCOUNT, <+Sollwert+> [,vergleichsmodus2] |
                  LIST, $Name$, TEXT, $Sollwert$, SUBITEM, <+Listenindex+> |
                  LIST, $Name$, NUM, <.Sollwert.> [,vergleichsmodus], SUBITEM,
                  <+Listenindex+> |
                  LIST, $Name$, ENABLESTATE, bool_wert |
                  LIST, $Name$, FOCUSSTATE, bool_wert
tabelle_-      =  TABLE, $Name$, ITEMCOUNT, <+Sollwert+> [,vergleichsmodus2] |
testen            TABLE, $Name$, TEXT, $Sollwert$, SUBITEM, <+Zeilenindex+>,
                  <+Spaltenindex+> |
                  TABLE, $Name$, NUM, <.Sollwert.> [,vergleichsmodus], SUBITEM,
                  <+Zeilenindex+>, <+Spaltenindex+> |
                  TABLE, $Name$, CHECKSTATE, bool_wert, SUBITEM, <+Zeilenindex+>,
                  <+Spaltenindex> |
                  TABLE, $Name$, ENABLESTATE, bool_wert |
                  TABLE, $Name$, FOCUSSTATE, bool_wert |
                  TABLE, $Name$, VISIBLESTATE, bool_wert
baum_testen    =  TREE, $Name$, ENABLESTATE, bool_wert |
                  TREE, $Name$, FOCUSSTATE, bool_wert |
                  TREE, $Name$, VISIBLESTATE, bool_wert
menü_testen    =  MENU, $Name$, CHECKSTATE, bool_wert, SUBITEM {,$Menüpunkt$}] |
                  MENU, $Name$, ENABLESTATE, bool_wert [, SUBITEM {,$Menüpunkt$}] |
                  MENU, $Name$, FOCUSSTATE, bool_wert |
                  MENU, $Name$, VISIBLESTATE, bool_wert
kartei_-       =  TABFOLDER, $Name$, ITEMCOUNT, <+Sollwert+> [,vergleichsmodus2] |
testen            TABFOLDER, $Name$, TEXT, $Sollwert$, SUBITEM, <+Tabindex+> |
                  TABFOLDER, $Name$, ENABLESTATE, bool_wert |
                  TABFOLDER, $Name$, FOCUSSTATE, bool_wert
fenster_-      =  WINDOW, $Name$, ENABLESTATE, bool_wert |
testen            WINDOW, $Name$, FOCUSSTATE, bool_wert |
                  WINDOW, $Name$, VISIBLESTATE, bool_wert
```

Vergleich von Ist- und Sollwerten *geändert*

```
vergleich                =  COMPARE, $Variable1$, vergleichskörper
vergleichskörper         =  vergleich_mit_variable | vergleich_mit_wert
vergleich_mit_variable   =  vergleich_var_textuell | vergleich_var_numerisch |
                            vergleich_var_bool
vergleich_mit_wert       =  vergleich_wert_textuell | vergleich_wert_numerisch |
                            vergleich_wert_bool

vergleich_var_textuell   =  STR, VAR, $Variable2$
vergleich_var_numerisch  =  NUM, VAR, $Variable2$ [,vergleichsmodus]
vergleich_var_bool       =  BOOL, VAR, $Variable2$

vergleich_wert_textuell  =  STR, VAL, $Wert$
vergleich_wert_numerisch =  NUM, VAL, <.Wert.> [,vergleichsmodus]
vergleich_wert_bool      =  BOOL, VAL, bool_wert
bool_wert                =  TRUE | FALSE

vergleichsmodus          =  EQ | GRT | GEQ | LEQ | LSS | NEQ | toleranz
toleranz                 =  TOL, <.Toleranzwert.>
vergleichsmodus2         =  EQ | GRT | GEQ | LEQ | LSS | NEQ
```

Aktionen auf Components *geändert*

```
aktion         =  ACTION, $Fenstertitel$ | MAIN, aktions_art
aktions_art    =  component_aktion | custom_aktion | editbox_aktion | label_aktion |
                  radiobutton_aktion | checkbox_aktion | button_aktion | combobox_-
                  aktion | listen_aktion | tabellen_aktion | baum_aktion | menü_-
                  aktion | kartei_aktion | fenster_aktion

component_-    =  COMPONENT, $Name$, tasten_druck |
aktion            COMPONENT, $Name$, maus_klick
custom_-       =  $+Custom+$, $Name$, PROPERTY, $Eigenschaft$, $Wert$
aktion
editbox_-      =  EDITBOX, $Name$, tasten_druck |
aktion            EDITBOX, $Name$, maus_klick |
                  EDITBOX, $Name$, EDIT, $Text$
label_aktion   =  LABEL, $Name$, tasten_druck |
                  LABEL, $Name$, maus_klick
radiobutton_-  =  RADIOBUTTON, $Name$, tasten_druck |
aktion            RADIOBUTTON, $Name$, maus_klick |
                  RADIOBUTTON, $Name$, CHECK |
                  RADIOBUTTON, $Name$, SELECT
checkbox_-     =  CHECKBOX, $Name$, tasten_druck |
aktion            CHECKBOX, $Name$, maus_klick |
                  CHECKBOX, $Name$, CHECK
                  CHECKBOX, $Name$, UNCHECK
                  CHECKBOX, $Name$, SELECT
```

```
button_-       =   BUTTON, $Name$, tasten_druck |
aktion             BUTTON, $Name$, maus_klick
                   BUTTON, $Name$, SELECT
combobox_-     =   COMBOBOX, $Name$, tasten_druck |
aktion             COMBOBOX, $Name$, maus_klick |
                   COMBOBOX, $Name$, SELECT, SUBITEM, <+Listenindex+> |
                   COMBOBOX, $Name$, EDIT, $Text$
listen_-       =   LIST, $Name$, tasten_druck |
aktion             LIST, $Name$, maus_klick |
                   LIST, $Name$, SELECT, SUBITEM, <+Listenindex+> |
                   LIST, $Name$, ADDSELECT, SUBITEM, <+Listenindex+> |
                   LIST, $Name$, DESELECT, SUBITEM, <+Listenindex+>
tabellen_-     =   TABLE, $Name$, tasten_druck |
aktion             TABLE, $Name$, maus_klick |
                   TABLE, $Name$, SELECT, SUBITEM, <+Zeilenindex+> |
                   TABLE, $Name$, ADDSELECT, SUBITEM, <+Zeilenindex+> |
                   TABLE, $Name$, DESELECT, SUBITEM, <+Zeilenindex+> |
                   TABLE, $Name$, EDIT, $Text$, SUBITEM, <+Zeilenindex+>,
                   <+Spaltenindex+> |
                   TABLE, $Name$, CHECK, SUBITEM, <+Zeilenindex+>, <+Spaltenindex+> |
                   TABLE, $Name$, UNCHECK, SUBITEM, <+Zeilenindex+>, <+Spaltenindex+>
baum_aktion    =   TREE, $Name$, tasten_druck |
                   TREE, $Name$, maus_klick |
                   TREE, $Name$, SELECT, SUBITEM {,$Knoten$} |
                   TREE, $Name$, ADDSELECT, SUBITEM, {$Knoten$} |
                   TREE, $Name$, DESELECT, SUBITEM {,$Knoten$} |
                   TREE, $Name$, EDIT, $Text$, SUBITEM, {$Knoten$}
menü_aktion    =   MENU, $Name$, tasten_druck |
                   MENU, $Name$, maus_klick |
                   MENU, $Name$, SELECT, SUBITEM {,$Menüpunkt$} |
                   MENU, $Name$, CHECK, SUBITEM {,$Menüpunkt$} |
                   MENU, $Name$, UNCHECK, SUBITEM {,$Menüpunkt$}
kartei_akion   =   TABFOLDER, $Name$, tasten_druck |
                   TABFOLDER, $Name$, maus_klick |
                   TABFOLDER, $Name$, SELECT, <+Tabindex+>
fenster_-      =   WINDOW, $Name$, tasten_druck |
aktion             WINDOW, $Name$, maus_klick |
                   WINDOW, $Name$, CLOSE |
                   WINDOW, $Name$, MAXIMIZE |
                   WINDOW, $Name$, MINIMIZE |
                   WINDOW, $Name$, NORMALIZE

tasten_druck   =   PRESSKEY, $Zeichen$ [,ALT] [,CTRL] [,SHIFT]
maus_klick     =   CLICK, LEFT | RIGHT, <+Klickanzahl+> [,ALT] [,CTRL] [,SHIFT]
```

Fenster vorhanden/nicht vorhanden

```
fenstersichtbarkeit = WINDOWEXISTS, $Fenstertitel$, YES|NO
```

Nachrichten/Fragen an Nutzer

```
nachricht  =  MESSAGE, $Text$
frage      =  QUESTION, $Text$, YES|NO
```

Datei kopieren

```
kopieren  =  COPY, dir, $Quelldatei$, dir, $Zieldatei$ [,FORCE] [,IFSRCEXISTS]
dir       =  ABS | TGT | REF | BIN | ENV | LOG
```

Existenz Datei/Verzeichnis

```
existenz              =  existenz_datei | existenz_verzeichnis
existenz_datei        =  EXISTS, dir, $Dateiname$
existenz_verzeichnis  =  EXISTS, dir, $Verzeichnisname$, DIRECTORY
```

Datei loeschen

```
loeschen = DELETE, dir, $Dateiname$ [,FORCE] [,IFEXISTS]
```

Starten einer Anwendung

```
starten           =  testobjekt_start | extern_start
testobjekt_start  =  START, $Parameter$
```

Starten eines externen Programmes

```
extern_start           =  start_warten_auf_ende | start_warten_auf_zeit |
                          start_ohne_warten
start_warten_auf_ende  =  LAUNCH, dir, $Programmname$, $Parameter$, FOREVER,
                          <Returncode>, [, $Logfile$]
start_warten_auf_zeit  =  LAUNCH, dir, $Programmname$, $Parameter$,
                          TIME, <+Terminierungsdauer+>, <Returncode>, [,
                          $Logfile$]
start_ohne_warten      =  LAUNCH, dir, $Programmname$, $Parameter$, NOWAIT
```

1.2 Die Semantik der HTS-Kommandos

1.2.1 ACTION

Aufruf:
ACTION, <FENSTER>, <COMPONENTTYP>, <COMPONENTNAME>, <AKTION>

Parameter:

<FENSTER>: Der initiale Titel des Fensters in dem sich das Component befindet, welches manipuliert werden soll. Für das Hauptfenster dient das Schlüsselwort MAIN als Platzhalter für den tatsächlichen Titel.

<COMPONENTTYP>: Ist der Typ eines Components. Dieser ist eine unabhängige Abstraktion basierend auf der Funktionalität des Components. Folgende Typen sind möglich: COMPONENT, EDITBOX, LABEL, RADIOBUTTON, CHECKBOX, BUTTON, COMBOBOX, LIST, TABLE, TREE, MENU, TABFOLDER und WINDOW. Zu jedem HTS Typ existiert ein entsprechender Java-Typ aus der jeweiligen GUI-Bibliothek. Die folgende Tabelle zeigt die Zuordnung von HTS Typen zu den Klassen der graphischen Objekte, getrennt nach den, von ATOSj unterstützten Bibliotheken SWT und Swing.

Componenttyp	**SWT**	**Swing**
COMPONENT	Widget	Component
EDITBOX	Text	JTextComponent
COMBOBOX	Combo	JComboBox
LIST	List	JList
TABLE	Table	JTable
TREE	Tree	JTree
TABFOLDER	TabFolder	JTabbedPane
LABEL	Label	JLabel
RADIOBUTTON	Button Stil: SWT.RADIO ToolItem Stil: SWT.RADIO	JRadioButton
CHECKBOX	Button Stil: SWT.CHECK oder SWT.TOGGLE ToolItem Stil: SWT.CHECK	JToggleButton
BUTTON	Button Stil: SWT.PUSH oder SWT.ARROW	AbstractButton
MENU	Menu	JMenu
WINDOW	Shell	java.awt.Frame java.awt.Dialog

<COMPONENTNAME>: Der Name des Components. Dieser bildet zusammen mit <FENSTER> und <COMPONENTTYP> einen eindeutigen Identifikator. Um bei der Testausführung ein Component anhand seiner ID zu finden, muss ATOSj die Namen der Compenents 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 `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 `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.

`MENU`	1. Der Name des Menüs. Festgelegt durch den Programmierer und abrufbar über die Funktion `getData(String key)` aus der Klasse `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 Titel des Fensters.	1. Der Titel des Fensters.

`<AKTION>`: Folgende Aktionen können auf einem Component, in Abhängigkeit von `<COMPONENTTYP>`, durchgeführt werden. Es ist zu beachten, dass alle Aktionsarten für den Typ `COMPONENT` auch für alle anderen Typen gültig sind.

COMPONENT	PRESSKEY, $Zeichen$ [,ALT] [,CTRL] [,SHIFT]	Setzt den Eingabefokus auf das Component und simuliert einen Tastendruck mit dem angebenen Zeichen und optional mit den Tasten Alt (ALT), Steuerung (CTRL) und Umschalt (SHIFT).
COMPONENT	CLICK, LEFT | RIGHT, <+Klickanzahl+> [,ALT] [,CTRL] [,SHIFT]	Simuliert einen Mausklick entweder mit der linken Taste (LEFT) oder mit der rechten Taste (RIGHT). Es werden <+Klickanzahl+> Klicks ausgeführt. Optional kann gleichzeitig das Drücken der Steuertasten Alt (ALT), Steuerung (CTRL) und Umschalt (SHIFT) simuliert werden.
$+Custom+$	PROPERTY, $Eigenschaft$, $Wert$	Setzt die Eigenschaft eines Custom-Components auf den angegegebenen Wert. Der Typname $+Custom+$ wird in der Datei custom.lst festgelegt.
EDITBOX	EDIT, $Text$	Setzt den Eingabefokus auf das Textfeld und simuliert die Eingabe von $Text$
CHECKBOX	CHECK	Wählt die Checkbox an, sofern sie vorher nicht schon angewählt war.
CHECKBOX	UNCHECK	Wählt die Checkbox ab, sofern sie nicht schon vorher abgewählt war.
CHECKBOX	SELECT	Simuliert einen einfachen Mausklick mit der linken Taste, ohne Rücksicht auf den vorhergehenden Status. Dies führt zu einem Umschalten auf den jeweils komplementären Anwahlzustand.
RADIOBUTTON	CHECK	Wählt den Radiobutton an, sofern er vorher nicht schon angewählt war.
RADIOBUTTON	SELECT	Simuliert einen einfachen Mausklick mit der linken Taste, ohne Rücksicht auf den vorhergehenden Status.
BUTTON	SELECT	Simuliert einen einfachen Mausklick mit der linken Taste.
COMBOBOX	SELECT, SUBITEM, <+Listenindex+>	Wählt den Eintrag an Position <+Listenindex+> aus der Liste der Combobox aus.
COMBOBOX	EDIT, $Text$	Setzt den Eingabefokus auf die Combobox und simuliert die Eingabe von $Text$.
LIST	SELECT, SUBITEM, <+Listenindex+>	Setzt die Auswahl der Liste auf die Zeile an Position <+Listenindex+>.
LIST	ADDSELECT, SUBITEM, <+Listenindex+>	Fügt der Auswahl der Liste, die Zeile an Position <+Listenindex+> hinzu, sofern noch nicht in der bisherigen Auswahl enthalten.

LIST	DESELECT, SUBITEM, <+Listenindex+>	Entfernt die Zeile an Position <+Listenindex+> aus der aktuellen Auswahl Liste, sofern enthalten.
TABLE	SELECT, SUBITEM, <+Zeilenindex+>	Setzt die Auswahl der Tabelle auf das Element an Position <+Zeilenindex+>.
TABLE	ADDSELECT, SUBITEM, <+Zeilenindex+>	Fügt der Auswahl der Tabelle das Element an Position <+Zeilenindex+> hinzu, sofern noch nicht in der bisherigen Auswahl enthalten.
TABLE	DESELECT, SUBITEM, <+Zeilenindex+>	Entfernt das Element an Position <+Zeilenindex+> aus der aktuellen Auswahl Tabelle, sofern enthalten.
TABLE	EDIT, $Text$, SUBITEM, <+Zeilenindex+>, <+Spaltenindex+>	Aktiviert den Zelleneditor für die Zelle an Position <+Zeilenindex+> und <+Spaltenindex+> und simuliert die Eingabe von $Text$.
TABLE	CHECK, SUBITEM, <+Zeilenindex+>, <+Spaltenindex+>	Setzt den Status eines Zelleneditors vom Typ CHECKBOX an Position <+Zeilenindex+> und <+Spaltenindex+> auf angewählt, sofern sie nicht schon vorher angewählt war.
TABLE	UNCHECK, SUBITEM, <+Zeilenindex+>, <+Spaltenindex+>	Setzt den Status eines Zelleneditors vom Typ CHECKBOX an Position <+Zeilenindex+> und <+Spaltenindex+> auf abgewählt, sofern sie nicht schon vorher abgewählt war.
TREE	SELECT, SUBITEM {,$Knoten$}	Setzt die Auswahl des Baumes auf den angegebenen Knoten. Der Pfad beinhaltet die Beschriftung aller Elternknoten und des auszuwählenden Knotens, beginnend beim Wurzelknoten.
TREE	ADDSELECT, SUBITEM {,$Knoten$}	Fügt den angegebenen Knoten der Auswahl des Baumes hinzu, sofern noch nicht enthalten. Der Pfad beinhaltet die Beschriftung aller Elternknoten und des hinzuzufügenden Knotens, beginnend beim Wurzelknoten.
TREE	DESELECT, SUBITEM {,$Knoten$}	Entfernt den angegebenen Knoten aus der Auswahl des Baumes, sofern enthalten. Der Pfad beinhaltet die Beschriftung aller Elternknoten und des abzuwählenden Knotens, beginnend beim Wurzelknoten.
TREE	EDIT, $Text$, SUBITEM {,$Knoten$}	Aktiviert den Zelleneditor für den Knoten mit dem angegebenen Pfad und simuliert die Eingabe von $Text$. Der Pfad beinhaltet die Beschriftung aller Elternknoten und des hinzuzufügenden Knotens, beginnend beim Wurzelknoten.

MENU	SELECT, SUBITEM {,$Menüpunkt$}	Klickt den Menüpunkt mit dem angegeben Pfad an. Der Pfad beinhaltet die Beschriftung aller übergeordneten Menüs und des zu klickenden Menüpunktes selbst, beginnend beim Wurzelmenü.
MENU	CHECK, SUBITEM {,$Menüpunkt$}	Aktiviert den Menüpunkt mit dem angegeben Pfad. Der Pfad beinhaltet die Beschriftung aller übergeordneten Menüs und des zu aktivierenden Menüpunktes selbst, beginnend beim Wurzelmenü.
MENU	UNCHECK, SUBITEM {,$Menüpunkt$}	Deaktiviert den Menüpunkt mit dem angegeben Pfad. Der Pfad beinhaltet die Beschriftung aller übergeordneten Menüs und des zu deaktivierenden Menüpunktes selbst, beginnend beim Wurzelmenü.
TABFOLDER	SELECT, <+Tabindex+>	Wählt den Karteireiter an Position <+Tabindex+> aus.
WINDOW	CLOSE	Schließt das Fenster.
WINDOW	MAXIMIZE	Maximiert die Größe des Fensters.
WINDOW	MINIMIZE	Minimiert die Größe des Fensters.
WINDOW	NORMALIZE	Setzt die Größe des Fensters auf die ursprüngliche Größe.

Beschreibung:
Simuliert Nutzeraktionen auf den angegebenen Components.

Beispiel:
`ACTION, ‘‘Dozent‘‘, TABFOLDER, ‘‘Tab‘‘, SELECT, SUBITEM, ‘‘Seminare‘‘`
Im Fenster mit dem Titel „Dozent“ wird aus der Kartei mit dem Namen „Tab“ der Karteireiter mit der Beschriftung „Seminare“ ausgewählt.

1.2.2 CLEANUP

Aufruf: `CLEANUP`

Parameter: keine

Beschreibung:
Falls während des Testvorgangs ein Fehler auftritt, wird direkt zu diesem Kommando gesprungen, sofern es vorhanden ist und der Ausführungsmodus das Anhalten bei einem Fehler fordert. Die mit `START` aufgerufene Anwendung wird ggf. geschlossen. Alle folgenden Kommandos bis zum Ende der Datei dienen der Wiederherstellung des Zustandes vor Beginn der Ausführung des aktuellen Testskriptes. Temporär angelegte Dateien sollten

hier gelöscht und originale Konfigurationsdateien wieder hergestellt werden, um das Testobjekt in seinen Ausgangszustand zu versetzen. Fehler beim Ausführen von Kommandos nach dem `CLEANUP`-Kommando werden ignoriert.

1.2.3 COMMENT

Aufruf: `COMMENT, <Kommentar>`

Parameter:

`<Kommentar>`: Der verbale Kommentar.

Beschreibung:
Realisiert einen verbalen Kommentar in einer Testsequenz. Dieses Kommando wird bei der Testausführung ignoriert.

Beispiel: `COMMENT, ‘‘Das ist ein Kommentar‘‘`

1.2.4 COMPARE

Aufruf: `COMPARE, <ISTWERT>, <TYP>, <VAR/VAL>, <SOLLWERT> [,<MODUS>]`

Parameter:

`<ISTWERT>`: Ist ein, in „-Zeichen eingeschlossener, Bezeichner für eine Variable, die den zu überprüfenden Istwert (Zahl, Zeichenkette oder boolescher Wert) enthält.

`<TYP>`: Gibt den Datentyp für den Vergleich an. Bei einem numerischen Vergleich von Soll- und Istwert steht an dieser Stelle `NUM`. Für einen Vergleich von Zeichenketten wird stattdessen `STR` verwendet. Das Schlüsselwort `BOOL` steht für einen booleschen Vergleich.

`<VAR/VAL>`: Gibt die Art des Sollwertes an. `VAR` steht für eine Variable und `VAL` für einen festen Wert.

`<SOLLWERT>`: Der Sollwert ist in Abhängigkeit vom Parameter `VAR/VAL` entweder der Bezeichner für Variable oder ein Wert entsprechend des Vergleichstyps, der im Parameter `TYP` angegeben wurde. Der Bezeichner für eine Variable muss eine, in Anführungszeichen eingeschlossene, Zeichenkette sein. Dabei wird das Backslash (\) als Maskierungszeichen verwendet. Ein numerischer Wert wird als reelle Zahl

angegeben. Als Dezimalzeichen dient der Punkt. Eine Zeichenkette wird in Anführungzeichen eingeschlossen. Dabei wird das Backslash (\) als Maskierungszeichen verwendet. Ein boolescher Wert wird entweder mit dem Schlüsselwort `TRUE` für wahr oder `FALSE` für falsch angegeben.

`<MODUS>`: Dieser optionale Parameter bestimmt bei einem numerischen Vergleich den Vergleichsoperator. Folgende Möglichkeiten werden unterstützt:

`EQ`	Istwert und Sollwert müssen gleich sein
`GRT`	Istwert muss größer als der Sollwert sein
`GEQ`	Istwert muss größer-gleich dem Sollwert sein
`LSS`	Istwert muss kleiner als der Sollwert sein
`LEQ`	Istwert muss kleiner-gleich dem Sollwert sein
`NEQ`	Istwert und Sollwert müssen unterschiedlich sein
`TOL, <Tol.Wert>`	Istwert darf maximal um ± `<Tol.Wert>` vom Sollwert abweichen

Beschreibung: Vergleicht Istwerte aus Variablen entweder direkt mit Sollwerten oder mit Werten aus anderen Variablen. Der Vergleich kann zeichenweise, numerisch oder boolesch sein. Dieses Kommando arbeitet eng mit `READ` zusammen, welches das Abspeichern von Werten in Variablen realisiert. Diese können dann zu einem späteren Zeitpunkt unter Verwendung von `COMPARE` mit Sollwerten verglichen werden.

Beispiel:
`COMPARE, ‘‘weiblich‘‘, BOOL, VAL, TRUE`
Es wird geprüft, ob die Variable mit dem Namen „weiblich“ mit dem booleschen Wert „wahr“ belegt ist.
`COMPARE, ‘‘var1‘‘, BOOL, VAR, ‘‘var2‘‘`
Es wird geprüft, ob die booleschen Variablen „var1“ und „var2“ den gleichen Wert besitzen.

1.2.5 COPY

Aufruf: `COPY, <QUELLVERZEICHNIS>, <QUELLDATEI>, <ZIELVERZEICHNIS>, <ZIELDATEI> [,FORCE] [,IFSRCEXISTS]`

Parameter:

`<QUELLVERZEICHNIS>, <ZIELVERZEICHNIS>`: Einige Pfade zu Verzeichnissen sind ATOSj zur Ausführungszeit bekannt und können mit diesen Parametern angesprochen werden. Das ermöglicht die Portabilität von ATOSj-Projekten (bzw. Testskripten), da

sie weitestgehend unabhängig von absoluten Pfadangaben sind. Mögliche Werte sind:

ABS	in <QUELLDATEI> bzw. <ZIELDATEI> steht ein absoluter Pfad
TGT	Ausführungsverzeichnis des Testobjekts
REF	Verzeichnis mit Solldateien (\REF)
BIN	Binary-Verzeichnis des Projektes (\BIN)
ENV	Verzeichnis der Umgebungsdateien des Projektes (\ENV)
LOG	Verzeichnis zur Aufnahme der Logdateien (\LOG)

<QUELLDATEI>, <ZIELDATEI>: Ist ein in "-Zeichen eingeschlossener Dateiname.

FORCE: Diese Direktive ist optional und notwendig, wenn die Zieldatei bereits existiert und versteckt oder schreibgeschützt ist. Ohne FORCE würde das Kopieren in diesem Falle scheitern und einen Fehler verursachen.

IFSRCEXISTS: Dieser optionale Parameter bewirkt, dass nur dann eine Kopieraktion ausgeführt wird, wenn die Quelldatei vorhanden ist. Dadurch werden Fehler unterbunden, wenn die Quelldatei nicht existiert.

Beschreibung: Kopieren von Dateien aus Verzeichnissen eines ATOSj-Projektes oder aus Verzeichnissen mit absoluten Pfaden.

Beispiel:
`COPY, REF, ''t1kunden.dat'', TGT, ''kunden.dat'', FORCE`
Kopiert die Datei „t1kunden.dat" aus dem Unterverzeichnis „REF" des Projektverzeichnisses in das Ausführungverzeichnis des Testobjekts. Die Kopie erhält den Namen „kunden.dat". Existiert die Zieldatei bereits, wird sie überschrieben.

1.2.6 DELETE

Aufruf: `DELETE, <VERZEICHNIS>, <DATEI> [,FORCE] [,IFEXISTS]`

Parameter:

<VERZEICHNIS>: Mögliche Werte sind: ABS, TGT, REF, BIN, ENV und LOG (siehe COPY)

<DATEI>: Ist ein in "-Zeichen eingeschlossener Dateiname.

FORCE: Diese Direktive ist optional und notwendig, wenn die Datei versteckt oder schreibgeschützt ist. Ohne FORCE würde das Löschen in diesem Falle scheitern und

einen Fehler verursachen.

`IFEXISTS`: Diese Direktive ist optional und unterbindet einen Fehler, wenn die zu löschende Datei nicht existiert.

Beschreibung: Löschen einer Datei.

Beispiel:
`DELET, TGT, ‘‘kunden.dat‘‘, IFEXISTS`
Löscht die Datei „kunden.dat“ aus dem Ausführungsverzeichnis des Testobjektes. Existiert die Datei nicht, so wird dies ignoriert.

1.2.7 DISABLE

Aufruf: `DISABLE, <HTS-Kommando>`

Parameter:

`<HTS-Kommando>`: Ein beliebiges anderes HTS-Kommando, das beim Testvorgang nicht ausgeführt werden soll.

Beschreibung: Dient zur Deaktivierung eines HTS-Kommandos. Wird in ATOSj benötigt, um Kommandos zu Testzwecken zu deaktivieren. Deaktivierte Kommandos sollten nicht teil einer vollendeten Testsequenz sein. Das deaktivierte Kommando wird beim Testvorgang nicht ausgeführt.

Beispiel:
`DISABLE, WAIT, 500`
Das `WAIT` Kommando wird nicht ausgeführt.

1.2.8 ENDLOOP

Aufruf: `ENDLOOP`

Parameter: keine

Beschreibung: Bildet das Ende eines `LOOP`-Schleifenblocks.

Beispiel: siehe Kommando `LOOP`

1.2.9 ENDWHILE

Aufruf: `ENDWHILE`

Parameter: keine

Beschreibung: Bildet das Ende eines `WHILE`-Schleifenblocks.

Beispiel: siehe Kommando `WHILE`

1.2.10 EXISTS

Aufruf: `EXISTS, <VERZEICHNIS>, <ZIEL> [,DIRECTORY]`

Parameter:

`<Verzeichnis>`: MöglicheWerte sind: `ABS, TGT, REF, BIN, ENV` und `LOG` (siehe `COPY`)

`<ZIEL>`: Ist ein in "-Zeichen eingeschlossener Datei- oder Verzeichnisname.

`DIRECTORY`: Dieser optionale Parameter zeigt an, dass es sich bei `<ZIEL>` um ein Verzeichnis und nicht um eine Datei handelt.

Beschreibung: Überprüft die Existens einer Datei oder eines Verzeichnisses.

Beispiel:
`EXISTS, ABS, ''C:\\DATA'', ''kunden.dat''`
Überprüft ob im angegebenen Verzeichnis die Datei „kunden.dat" existiert.

1.2.11 LAUNCH

1. Warten auf Terminierung

Aufruf: `LAUNCH, <VERZEICHNIS>, <PROGRAMMNAME>, <PARAMETER>, FOREVER, <RETURNCODE>, [,<LOGFILE>]`

Parameter:

`<Verzeichnis>`: MöglicheWerte sind: `ABS, TGT, REF, BIN, ENV` und `LOG` (siehe `COPY`)

<PROGRAMMNAME>: Ist ein in "-Zeichen eingeschlossener Dateiname.

PARAMETER: Sind in "-Zeichen eingeschlossene Parameter zur Übergabe an das zu startende Programm.

<RETURNCODE>: Der Rückkehrcode des aufgerufenen Programmes wird mit <RETURNCODE> verglichen. Bei Nichtübereinstimmung wird wird ein Fehler gemeldet.

<LOGFILE>: Ist ein in "-Zeichen eingeschlossener Dateiname. Die Ausgabedatei (Protokolldatei o.ä.) des aufgerufenen Programmes wird durch Angabe ihres Namens in <LOGFILE> aus dem Ausführungsverzeichnis des Programms in das LOG-Verzeichnis des ATOSj-Projekts verschoben.

Beschreibung: Startet ein externes Programm für testbegleitende Aufgaben (Dateivergleicher, Grafikviewer etc.), wartet auf dessen Terminierung und schreibt gegebenenfalls ein Logfile in das LOG-Verzeichnis des ATOSj-Projektes.

Beispiel:
`LAUNCH, ABS, ‘‘mysql‘‘, ‘‘-e\‘‘Drop database husemorg;\‘‘‘‘, FOREVER, 0`
Startet das Programm „mysql“ mit den angegebenen Parametern und wartet unbegrenzte Zeit auf dessen Beendigung. Überprüft ob der Rückgabewert des Programmes dem Wert 0 entspricht.

2. Warten auf Zeit

Aufruf: `LAUNCH, <VERZEICHNIS>, <PROGRAMMNAME>, <PARAMETER>, TIME, <TERMINIERUNGSDAUER>, <RETURNCODE>, [,<LOGFILE>]`

Parameter:

<Terminierungsdauer>: Dauer in Millisekunden, nach der das aufgerufene Programm beendet sein muss. Ist das Programm nach Ablauf dieser Dauer nicht fertig, wird ein Fehler gemeldet.

Alle anderen Parameter wie bei 1.

Beschreibung: Startet ein externes Programm für testbegleitende Aufgaben (Dateivergleicher, Grafikviewer etc.), wartet auf dessen Terminierung und schreibt gegebenenfalls ein Logfile in das LOG-Verzeichnis des ATOSj-Projektes.

Beispiel:
`LAUNCH, ABS, ‘‘mysql‘‘, ‘‘-e\‘‘Drop database husemorg;\‘‘‘‘, TIME, 2000, 1`
Startet das Programm „mysql“ mit den angegebenen Parametern und wartet 2 Sekunden

auf dessen Beendigung. Überprüft ob der Rückgabewert des Programmes dem Wert 1 entspricht.

3. Nicht auf Terminierung warten

Aufruf: `LAUNCH, <VERZEICHNIS>, <PROGRAMMNAME>, <PARAMETER>, NOWAIT`

Parameter: Parameter wie bei 1.

Beschreibung: Startet ein externes Programm für testbegleitende Aufgaben (Dateivergleicher, Grafikviewer etc.), und wartet nicht auf dessen Terminierung.

Beispiel:
```
LAUNCH, ABS, ‘‘mysql‘‘, ‘‘-e\‘‘Drop database husemorg;\‘‘‘‘, NOWAIT
```
Startet das Programm „mysql“ mit den angegebenen Parametern. Es wird sofort mit der Ausführung der Testsequenz fortgefahren.

1.2.12 LOOP

Aufruf: `LOOP, <ZYKLEN>`

Parameter:

`<ZYKLEN>`: Gibt die Anzahl der Schleifendurchläufe an.

Beschreibung: Bildet den Anfang eines Schleifenblocks. Alle Kommandos zwischen `LOOP` und `ENDLOOP` werden bei jedem Zyklus wiederholt. Beliebig tiefe Verschachtelungen der Schleifenblöcke sind zulässig.

Beispiel:
```
LOOP, 10
ACTION, ‘‘Mitglieder‘‘, BUTTON, ‘‘add‘‘, SELECT
ENDLOOP
```
Führt 10 Mal hintereinander das `ACTION` Kommando aus.

1.2.13 MESSAGE

Aufruf: `MESSAGE, <NACHRICHT>`

Parameter:

`<NACHRICHT>`: Eine in „-Zeichen eingeschlossene Nachricht, die in einem Dialog angezeigt wird.

Beschreibung: Zeigt einen Dialog mit einer frei definierbaren Nachricht an den Tester. Das Testobjekt kann während der Existenz dieses Dialogs manipuliert werden. Nach dem Drücken des Buttons "OK" wird mit der Testausführung fortgefahren.

Beispiel:
`MESSAGE, ‘‘Zeichnen Sie einen Kreis!‘‘`
Öffnet ein Fenster in ATOSj mit der angegebenen Aufforderung an den Tester.

1.2.14 QUESTION

Aufruf: `QUESTION, <FRAGE>, <ANTWORT>`

Parameter:

`<FRAGE>`: Eine in „-Zeichen eingeschlossene Frage, die in einem Dialog angezeigt wird.

`<ANTWORT>`: Die erwartete Antwort auf die gestellte Frage. Mögliche Werte sind `YES` für eine Antwort mit „Ja“ und `NO` für eine Antwort mit „Nein“.

Beschreibung: Zeigt einen Dialog mit einer frei definierbaren Ja/Nein-Frage. Erst nach Beantwortung der Frage, durch Anklicken des „Ja“- oder „Nein“-Buttons, wird mit der Testdurchführung fortgefahren. Stimmt die Antwort des Testers nicht mit dem in `<ANTWORT>` definierten Wert überein, wird ein Fehler gemeldet.

Beispiel:
`QUESTION, ‘‘Sehen Sie in der Statusanzeige einen grünen Punkt?‘‘, YES`
Öffnet einen Ja/Nein-Dialog in ATOSj mit der angegebenen Frage. Die erwartete Antwort ist „Ja“.

1.2.15 READ

Aufruf: `READ, <FENSTER>, <COMPONENTTYP>, <COMPONENTNAME>, <ZUSTAND>, <VARIABLE>`

Parameter:

`<FENSTER>`: Der initiale Titel des Fensters in dem sich das Component befindet, dessen Zustand ermittelt werden soll. Für das Hauptfenster dient das Schlüsselwort MAIN

als Platzhalter für den tatsächlichen Titel.

<COMPONENTTYP>: Ist der Typ eines Components. Dieser ist eine unabhängige Abstraktion basierend auf der Funktionalität des Components. Folgende Typen sind möglich: COMPONENT, EDITBOX, LABEL, RADIOBUTTON, CHECKBOX, BUTTON, COMBOBOX, LIST, TABLE, TREE, MENU, TABFOLDER und WINDOW.

<COMPONENTNAME>: Der Name des Components. Dieser bildet zusammen mit <FENSTER> und <COMPONENTTYP> einen eindeutigen Identifikator, über den das Component zur Laufzeit ermittelt wird. Der Name wird nach einer Strategie ermittelt, die möglichst gute Klarnamen für das Component liefert, siehe ACTION.

<VARIABLE>: Der Bezeichner für die Variable, die den ausgelesenen Zustandswert speichern soll.

<ZUSTAND>: Folgenden Zustände können, in Abhängigkeit von <COMPONENTTYP>, ausgelesen werden. Es ist zu beachten, dass alle Zustände des Typs COMPONENT auch für alle anderen Typen gültig sind.

COMPONENT	ENABLESTATE	Ermittelt, ob das Component aktiviert ist, d.h. Eingaben verarbeitet. Das Ergebnis ist ein boolescher Wert, mit TRUE für aktiviert und FALSE für deaktiviert.
COMPONENT	FOCUSSTATE	Ermittelt, ob das Component den Eingabefokus besitzt. Das Ergebnis ist ein boolescher Wert, mit TRUE für fokussiert und FALSE für nicht fokussiert.
COMPONENT	VISIBLESTATE	Ermittelt, ob das Component sichtbar ist. Das Ergebnis ist ein boolescher Wert, mit TRUE für sichtbar und FALSE für unsichtbar.
$+Custom+$	PROPERTY, $Eigenschaft$	Liest eine Eigenschaft eines Custom-Components aus. Ergebnis ist eine Zeichenkette. Der Typname $+Custom+$ wird in der Datei custom.lst festgelegt.
EDITBOX	TEXT	Liest den Text eines Textfeldes. Das Ergebnis ist eine Zeichenkette.
EDITBOX	NUM	Liest den Text eines Textfeldes und interpretiert ihn als reelle Zahl. Für die Dezimalstelle wird ein Punkt erwartet.
LABEL	TEXT	analog zu EDITBOX
LABEL	NUM	analog zu EDITBOX
RADIOBUTTON	CHECKSTATE	Ermittelt, ob der Radiobutton angewählt ist. Das Ergebnis ist ein boolescher Wert, mit TRUE für angewählt und FALSE für nicht angewählt.
RADIOBUTTON	TEXT	analog zu EDITBOX
RADIOBUTTON	NUM	analog zu EDITBOX

CHECKBOX	CHECKSTATE	Ermittelt, ob die Checkbox angewählt ist. Das Ergebnis ist ein boolescher Wert, mit TRUE für angewählt und FALSE für nicht angewählt.
CHECKBOX	TEXT	analog zu EDITBOX
CHECKBOX	NUM	analog zu EDITBOX
BUTTON	TEXT	analog zu EDITBOX
BUTTON	NUM	analog zu EDITBOX
COMBOBOX	ITEMCOUNT	Ermittelt die Anzahl der Einträge in der Liste der Combobox. Ergebnis ist ein ganzzahliger Wert.
COMBOBOX	TEXT	Liest den Text aus dem Textfeld der Combobox. Ergebnis ist eine Zeichenkette.
COMBOBOX	NUM	Liest den Text aus dem Textfeld der Combobox und interpretiert ihn als reelle Zahl. Für die Dezimalstelle wird ein Punkt erwartet.
LIST	ITEMCOUNT	Ermittelt die Anzahl der Einträge in der Liste. Ergebnis ist ein ganzzahliger Wert.
LIST	TEXT, SUBITEM, <+Listenindex+>	Liest den Text des Eintrages in der Zeile <+Listenindex+>.
TABLE	ITEMCOUNT	Ermittelt die Anzahl der Zeilen in der Tabelle. Ergebnis ist ein ganzzahliger Wert.
TABLE	TEXT, SUBITEM, <+Zeilenindex+>, <+Spaltenindex+>	Liest den Text der Tabellenzelle an Position <+Zeilenindex+>, <+Spaltenindex+>.
TABLE	NUM, SUBITEM, <+Zeilenindex+>, <+Spaltenindex+>	Liest den Text der Tabellenzelle an Position <+Zeilenindex+>, <+Spaltenindex+> und interpretiert ihn als reelle Zahl. Für die Dezimalstelle wird ein Punkt erwartet.
TABLE	CHECKSTATE, SUBITEM, <+Zeilenindex+>, <+Spaltenindex+>	Ermittelt, ob die Checkbox in der Tabellenzelle an Position <+Zeilenindex+>, <+Spaltenindex+> angewählt ist. Das Ergebnis ist ein boolescher Wert, mit TRUE für angewählt und FALSE für nicht angewählt.
MENU	ENABLESTATE, SUBITEM {,$Menüpunkt$}	Ermittelt, ob der Menüpunkt mit dem angegebenen Pfad aktiviert ist. Der Pfad beinhaltet die Beschriftung aller übergeordneten Menüs und des gesuchten Menüpunktes selbst, beginnend beim Wurzelmenü.
MENU	CHECKSTATE, SUBITEM {,$Menüpunkt$}	Ermittelt, ob der Menüpunkt mit dem angegebenen Pfad angewählt ist. Der Pfad beinhaltet die Beschriftung aller übergeordneten Menüs und des gesuchten Menüpunktes selbst, beginnend beim Wurzelmenü.
TABFOLDER	ITEMCOUNT	Ermittelt die Anzahl an Karteireitern.

TABFOLDER	TEXT, SUBITEM, <+Tabindex+>	Liest den Text des Karteireiters an Position <+Tabindex+>.
WINDOW	siehe COMPONENT	

Beschreibung: Abfragen und Zwischenspeichern von Zuständen und Werten eines Components in einer Variable. Der Wert der Variable kann zu einem späteren Zeitpunkt, unter Verwendung des Kommandos COMPARE, mit einer anderen Variable oder einem festen Wert verglichen werden.

Beispiel:
READ, ‘‘Dozentliste‘‘, TABLE, ‘‘Dozenten‘‘, TEXT, ‘‘name‘‘, SUBITEM, 0, 0
Liest den Text der Tabellenzelle in Zeile 0, Spalte 0 der Tabelle mit dem Namen „Dozenten“ im Fenster „Dozentenliste“ aus und speichert den ermittelten Wert in der Variable „name“.

1.2.16 START

Aufruf: START, <PARAMETER>

Parameter:

<PARAMETER>: Eine Zeichenkette mit Parametern, die dem Testobjekt beim Start übergeben werden.

Beschreibung: Startet das Testobjekt und wartet auf das Erscheinen des Hauptfensters. Der Titel des Hauptfensters, die Main-Klasse und der Klassenpfad für das Testobjekt werden in den Projektparametern von ATOSj definiert. Hat sich das Hauptfenster nach Ablauf einer festgelegten Wartezeit nicht geöffnet oder kann das Programm nicht gestartet werden, wird ein Fehler gemeldet.

Beispiel:
START, ‘‘-exclusive kunden.dat‘‘
Startet das Testobjekt und übergibt die angegebenen Programmparameter.

1.2.17 TEST

Aufruf: TEST, <FENSTER>, <COMPONENTTYP>, <COMPONENTNAME>, <SOLLWERTVERGLEICH>

Parameter:

<FENSTER>: Der initiale Titel des Fensters in dem sich das Component befindet, dessen Zustand ermittelt werden soll. Für das Hauptfenster dient das Schlüsselwort MAIN als Platzhalter für den tatsächlichen Titel.

<COMPONENTTYP>: Ist der Typ eines Components. Dieser ist eine unabhängige Abstraktion basierend auf der Funktionalität des Components. Folgende Typen sind möglich: COMPONENT, EDITBOX, LABEL, RADIOBUTTON, CHECKBOX, BUTTON, COMBOBOX, LIST, TABLE, TREE, MENU, TABFOLDER und WINDOW.

<COMPONENTNAME>: Der Name des Components. Dieser bildet zusammen mit <FENSTER> und <COMPONENTTYP> einen eindeutigen Identifikator, über den das Component zur Laufzeit ermittelt wird. Der Name wird nach einer Strategie ermittelt, die möglichst gute Klarnamen für das Component liefert, siehe ACTION.

<SOLLWERTVERGLEICH>: Zu jedem Typ gibt es eine Menge von Zuständen, die mit Sollwerten verglichen werden können. Folgende Zustandsprüfungen können, in Abhängigkeit von <COMPONENTTYP>, vorgenommen werden. Es ist zu beachten, dass alle Zustandsprüfungen für den Typ COMPONENT auch für alle anderen Typen gelten.

COMPONENT	ENABLESTATE, TRUE	Überprüft, ob das Component aktiviert ist.
COMPONENT	ENABLESTATE, FALSE	Überprüft, ob das Component deaktiviert (ausgegraut) ist.
COMPONENT	FOCUSSTATE, TRUE	Überprüft, ob das Component den Eingabefokus hat.
COMPONENT	FOCUSSTATE, FALSE	Überprüft, ob das Component nicht den Eingabefokus hat.
COMPONENT	VISIBLESTATE, TRUE	Überprüft, ob das Component sichtbar ist.
COMPONENT	VISIBLESTATE, FALSE	Überprüft, ob das Component unsichtbar ist.
$+Custom+$	PROPERTY, $Eigenschaft$, $Sollwert$	Überprüft, ob der Eigenschaftswert des Custom-Components dem Sollwert entspricht. Es wird nur auf Gleichheit geprüft. Der Typname $+Custom+$ wird in der Datei custom.lst festgelegt.
EDITBOX	TEXT, $Text$	Überprüft, ob der Text in der Editbox mit $Text$ übereinstimmt.
EDITBOX	NUM, <.Wert.> [,<MODUS>]	Liest den Text einer Editbox interpretiert ihn als reelle Zahl und vergleicht ihn mit <.Wert.> entsprechend des angegeben <MODUS>.
LABEL	TEXT, $Text$	analog zu EDITBOX
LABEL	NUM, <.Wert.> [,<MODUS>]	analog zu EDITBOX
RADIOBUTTON	CHECKSTATE, TRUE	Überprüft, ob der Radiobutton angewählt ist.
RADIOBUTTON	CHECKSTATE, FALSE	Überprüft, ob der Radiobutton abgewählt ist.
RADIOBUTTON	TEXT, $Text$	analog zu EDITBOX

RADIOBUTTON	NUM, <.Wert.> [,<MODUS>]	analog zu EDITBOX
CHECKBOX	CHECKSTATE, TRUE	Überprüft, ob die Checkbox angewählt ist.
CHECKBOX	CHECKSTATE, FALSE	Überprüft, ob die Checkbox abgewählt ist.
CHECKBOX	TEXT, $Text$	analog zu EDITBOX
CHECKBOX	NUM, <.Wert.> [,<MODUS>]	analog zu EDITBOX
BUTTON	TEXT, $Text$	analog zu EDITBOX
BUTTON	NUM, <.Wert.> [,<MODUS>]	analog zu EDITBOX
COMBOBOX	ITEMCOUNT, <+Wert+>, [,<MODUS>]	Vergleicht die Anzahl der Einträge in der Liste der Combobox mit <+Wert+> entsprechend des angegebenen <MODUS>.
COMBOBOX	TEXT, $Text$	analog zu EDITBOX
COMBOBOX	NUM, <.Wert.> [,<MODUS>]	analog zu EDITBOX
LIST	ITEMCOUNT, <+Wert+> [,<MODUS>]	Vergleicht die Anzahl der Einträge in der Liste mit <+Wert+> entsprechend des angegebenen <MODUS>.
LIST	TEXT, $Text$, SUBITEM, <+Listenindex+>	Überprüft, ob der Text des Eintrages in Zeile <+Listenindex+> mit $Text$ übereinstimmt.
TABLE	ITEMCOUNT, <+Wert+> [,<MODUS>]	Vergleicht die Anzahl der Zeilen in der Tabelle mit <+Wert+> entsprechend des angegebenen <MODUS>.
TABLE	TEXT, $Text$, SUBITEM, <+Zeilenindex+>, <+Spaltenindex>	Überprüft, ob der Text der Tabellenzelle an Position <+Zeilenindex+>, <+Spaltenindex+> mit $Text$ übereinstimmt.
TABLE	NUM, <.Wert.> [,<MODUS>], SUBITEM, <+Zeilenindex+>, <+Spaltenindex>	Liest den Text der Tabellenzelle an Position <+Zeilenindex+>, <+Spaltenindex+>, interpretiert ihn als reelle Zahl und vergleicht ihn mit <.Wert.> entsprechend des angegeben <MODUS>.
TABLE	CHECKSTATE, TRUE, SUBITEM, <+Zeilenindex+>, <+Spaltenindex>	Überprüft, ob die Checkbox in der Tabellenzelle an Position <+Zeilenindex+>, <+Spaltenindex+> angewählt ist.
TABLE	CHECKSTATE, FALSE, SUBITEM, <+Zeilenindex+>, <+Spaltenindex>	Überprüft, ob die Checkbox in der Tabellenzelle an Position <+Zeilenindex+>, <+Spaltenindex+> abgewählt ist.
MENU	ENABLESTATE, TRUE, SUBITEM {,$Menüpunkt$}	Überprüft, ob der Menüpunkt mit dem angegebenen Pfad aktiviert ist. Der Pfad beinhaltet die Beschriftung aller übergeordneten Menüs und des gesuchten Menüpunktes selbst, beginnend beim Wurzelmenü.
MENU	ENABLESTATE, FALSE, SUBITEM {,$Menüpunkt$}	Überprüft, ob der Menüpunkt mit dem angegebenen Pfad deaktiviert ist. Der Pfad beinhaltet die Beschriftung aller übergeordneten Menüs und des gesuchten Menüpunktes selbst, beginnend beim Wurzelmenü.

MENU	CHECKSTATE, TRUE, SUBITEM {,$Menüpunkt$}	Überprüft, ob der Menüpunkt mit dem angegebenen Pfad angewählt ist (mit Häkchen). Der Pfad beinhaltet die Beschriftung aller übergeordneten Menüs und des gesuchten Menüpunktes selbst, beginnend beim Wurzelmenü.
MENU	CHECKSTATE, FALSE, SUBITEM {,$Menüpunkt$}	Überprüft, ob der Menüpunkt mit dem angegebenen Pfad abgewählt ist (ohne Häkchen). Der Pfad beinhaltet die Beschriftung aller übergeordneten Menüs und des gesuchten Menüpunktes selbst, beginnend beim Wurzelmenü.
TABFOLDER	ITEMCOUNT, <+Wert+> [,<MODUS>]	Vergleicht die Anzahl der Karteireiter mit <+Wert+> entsprechend des angegebenen <MODUS>.
TABFOLDER	TEXT, $Text$, SUBITEM, <+Tabindex+>	Überprüft, ob der Text des Karteireiters an Position <+Tabindex+> mit $Text$ übereinstimmt.
WINDOW	siehe COMPONENT	

Beschreibung: Fragt Zustände von Components ab und vergleicht sie mit den angegebenen Sollwerten. Stimmen der ausgelesene Wert eines Zustands und der Sollwert nicht überein, wird ein Fehler gemeldet.

Beispiel:
`TEST, ‘‘Mitglieder‘‘, BUTTON, ‘‘add‘‘, ENABLESTATE, TRUE`
Überprüft, ob der Knopf mit dem Namen „add“ im Fenster „Mitglieder“ aktiviert ist.

1.2.18 WAIT

Aufruf: `WAIT, <TIMEOUT>`

Parameter:

`<TIMEOUT>`: Gibt die Dauer in Millisekunden an, die gewartet werden soll.

Beschreibung: Wartet die in `<TIMEOUT>` angegebene Zeit bevor mit der Ausführung des Testskriptes fortgefahren wird.

Beispiel:
`WAIT, 1500`
Hält die Ausführung der Testsequenz wird für 1,5 Sekunden an.

1.2.19 WINDOWEXISTS

Aufruf: `WINDOWEXISTS, <FENSTER>, <ABFRAGE>`

Parameter:

`<FENSTER>`: Der initiale Titel des Fensters, dessen Existenz übepüft werden soll. Für das Hauptfenster dient das Schlüsselwort MAIN als Platzhalter für den tatsächlichen Titel.

`<ABFRAGE>`: Soll überprüft werden, ob ein Fenster momentan existiert bzw. sichtbar ist, muss YES verwendet werden, andernfalls NO.

Beschreibung: Abfrage der Existenz bzw. Sichtbarkeit eines Fensters. Entspricht der ermittelte Wert nicht der `<ABFRAGE>` dann wird ein Fehler gemeldet.

Beispiel:
`WINDOWEXISTS, ‘‘Datensatz speichern‘‘, NO`
Überprüft, ob das Fenster mit dem Namen „Datensatz speichern“ nicht existiert.