

# OpenOffice.org TestTool

Version 1.16

---

## Introduction to Automated GUI Testing

Sun Microsystems, Inc.  
901 San Antonio Road  
Palo Alto, CA 94303  
U.S.A. 650-960-1300

June 2003

# Table of Contents

<b>1 The OpenOffice.org TestTool.....</b>	<b>7</b>
1.1 About the TestTool.....	7
1.2 Location of the TestTool.....	7
1.3 Installing the TestTool.....	8
1.4 Setting up the TestTool.....	8
1.5 Adding the TestTool Library to an OpenOffice.org Installation.....	9
1.6 Starting a TestTool Script.....	10
1.7 TestTool Editor.....	11
1.8 Result File.....	12
1.9 Starting a Test Script From the Command Line.....	13
1.10 Testing a Non-Product Version .....	14
<b>2 The TestTool Environment.....</b>	<b>15</b>
2.1 Introduction.....	15
2.2 global-Module.....	17
2.3 CVS information.....	17
<b>3 Declaration of OpenOffice.org for the TestTool.....</b>	<b>18</b>
3.1 Introduction.....	18
3.2 Types of Declaration.....	19
3.3 Determining the SlotID of a Menu Item.....	20

3.4	Determining the HelpIDs or UniqueIDs of Controls.....	21
3.5	Declaring a New Menu Item.....	23
3.6	Declaring a New Dialog or Control.....	24
3.7	About the hid.lst File.....	25
3.7.1	Location of the hid.lst file.....	25
3.7.2	Creating a working hid.lst.....	25
3.7.3	Viewing hid.lst errors.....	25
<b>4</b>	<b>The Structure of Test Scripts.....</b>	<b>26</b>
4.1	Introduction.....	26
4.2	sub main.....	26
4.3	GetUseFiles.....	26
4.4	testcase ... endcase.....	27
4.5	try ... catch ... endcatch.....	27
4.6	sub and function.....	28
<b>5</b>	<b>Internal Commands, Methods and Functions for TestTool.....</b>	<b>29</b>
5.1	Testscript Structure.....	31
5.2	Global.....	31
5.2.1	Statistic.....	33
5.3	Commands on The Office Side.....	34
5.4	Windows, Controls and Objects.....	34
5.4.1	Any Control, Window, Object.....	34
5.4.2	CheckBox.....	36
5.4.3	Image-CheckBox .....	36
5.4.4	ComboBox .....	36
5.4.5	Dialog.....	36
5.4.6	ModelessDialog.....	36
5.4.7	DockingWin.....	37
5.4.7.1	Splitting Windows .....	37
5.4.8	EditWindow.....	37
5.4.9	Edit-Field.....	38
5.4.10	MultiLineEdit-Field.....	38
5.4.11	FloatWin (Flyer).....	38
5.4.12	Listbox.....	38
5.4.13	MultiListBox.....	38

5.4.14	MenuBar.....	39
5.4.15	Menu and Context Menu.....	39
5.4.15.1	special methods.....	40
5.4.16	MessBox / InfoBox.....	40
5.4.17	WarningBox / ErrorBox .....	40
5.4.18	QueryBox.....	40
5.4.18.1	with checkbox(es).....	40
5.4.19	MoreButton .....	40
5.4.20	PushButton, ImageButton.....	41
5.4.21	RadioButton, ImageRadioButton.....	41
5.4.22	SpinField.....	41
5.4.23	PatternField.....	41
5.4.24	NumericField.....	41
5.4.25	MetricField.....	41
5.4.26	CurrencyField.....	41
5.4.27	DateField.....	41
5.4.28	TimeField.....	41
5.4.29	StatusBar.....	41
5.4.30	System Dialog.....	42
5.4.31	TabPage.....	42
5.4.32	Toolbar.....	42
5.4.33	TreeListbox.....	43
5.4.34	BrowseBox.....	43
5.4.35	Control.....	43
5.4.35.1	with Checkbox(es).....	44
5.4.36	TriStateBox.....	44
<b>6</b>	<b>Working With XML Files.....</b>	<b>45</b>
6.1	Introduction.....	45
6.1.1	Main Commands.....	45
6.1.2	Working With Elements, Attributes And Characters.....	46
6.1.3	Navigate And Search in The DOM.....	46
<b>7</b>	<b>Typekeys Instruction.....</b>	<b>47</b>
7.1	Description.....	47

<b>8 Resource Types of all Controls and Windows.....</b>	<b>49</b>
<b>9 All Supported Languages.....</b>	<b>50</b>
<b>10 The Style and Coding Standard used in the Automated Testing.....</b>	<b>51</b>
10.1 Revisions.....	51
10.2 Introduction.....	51
10.3 Section One - MUST.....	51
10.3.1 Variable.....	51
10.3.2 Indentation.....	52
10.3.3 Blank Spaces .....	53
10.3.4 Comments.....	53
10.3.4.1 Block Comments.....	54
10.3.4.2 Single Line Comments.....	54
10.3.4.3 Trailing Comments.....	54
10.3.5 File Organization.....	54
10.3.5.1 Sections inside of .inc files.....	54
10.3.5.2 Sections inside of .bas files.....	54
10.3.6 Program Organization.....	55
10.3.6.1 Sections inside of testcases/subs/functions.....	55
10.3.7 Methods, Hints, Annotations.....	55
10.3.7.1 Selecting the same string in different languages.....	55
10.3.7.2 Saving files during a test.....	55
10.3.7.3 Using testing levels.....	56
10.4 Section Two - ADVANCED.....	56
10.5 Appendix.....	56
10.5.0.1 if..then..else.....	57
10.5.0.2 select...case.....	57
10.6 Bibliography.....	57
<b>11 Testcase Documentation in Test Scripts.....</b>	<b>58</b>
11.1 Description.....	58
11.2 The *.bas files.....	59
11.3 The *.inc-files.....	60
11.4 Standards for Testcase Documentation.....	60

<b>12 Configuration file entries.....</b>	<b>62</b>
12.1 GUI Platform.....	62
<b>13 Alphabetical Index.....</b>	<b>63</b>

# 1 The OpenOffice.org TestTool

## *Tool for Automated Testing of OpenOffice.org*

### 1.1 About the TestTool

The TestTool is a standalone program that is used for the automated testing of OpenOffice.org. The TestTool communicates with the TCP/IP-Interface of OpenOffice.org and can test each installation of OpenOffice.org on a PC or in a local area network (LAN). The current TestTool can be used on OpenOffice.org 1.1beta and higher. However, as there can be some incompatible changes in future OpenOffice.org builds you may need to use a newer version of the TestTool.

You do not need a complete OpenOffice.org installation to run the TestTool so long as the OpenOffice.org libraries that are required by the tool are in the same directory as the **testtool**-script on UNIX or the **testtool.exe** application on Win32. Furthermore, the **.testtoolrc** (UNIX) and the **testtool.ini** (Win32) files must be in the *home* directory of the user (UNIX) or in the *profile* directory of a user on Win32.

The TestTool communicates with OpenOffice.org using SlotIDs, UniqueIDs, and HelpIDs that are associated with each menu item, window, dialog, and window or dialog control in OpenOffice.org. The IDs are automatically generated during the OpenOffice.org build-process or are assigned by developers.

**SlotIDs:** Each menu item has a SlotID that is used, for example, to open a dialog or perform an action.

**HelpIDs:** Each control, window, or dialog automatically receives a HelpID for the internal Help-System. The TestTool uses HelpIDs to identify specific controls, windows, or dialogs.

**UniqueIDs:** A developer can assign a UniqueID to a control that does not have a HelpID so that the TestTool can identify the control.

You can create test scripts for the TestTool that the same functionality as StarBasic (like VisualBasic). Several commands are available for the TestTool, for example, to get or put information into OpenOffice.org controls. For a complete list of the available commands, see chapter 5: *Internal Commands, Methods and Functions for TestTool*.

You can simulate most mouse or keyboard actions with the TestTool as well as gather information from controls or change the default settings in OpenOffice.org. In other words, the TestTool lets you simulate an OpenOffice.org user.

### 1.2 Location of the TestTool

The *TestTool* application and the *TestTool Environment* can be checked out via CVS on openoffice.org. We provide also beta or final versions archives of the *TestTool Environment* and the TestTool application on <http://qa.openoffice.org/qatestool>.

## 1.3 Installing the TestTool

To install the TestTool, extract the contents of the downloaded TestTool archive to your local disk. If more than one user will use the TestTool, copy the contents of the extracted archive to a network drive. You can run the TestTool on Solaris (SPARC and x86), Linux, and Win32.

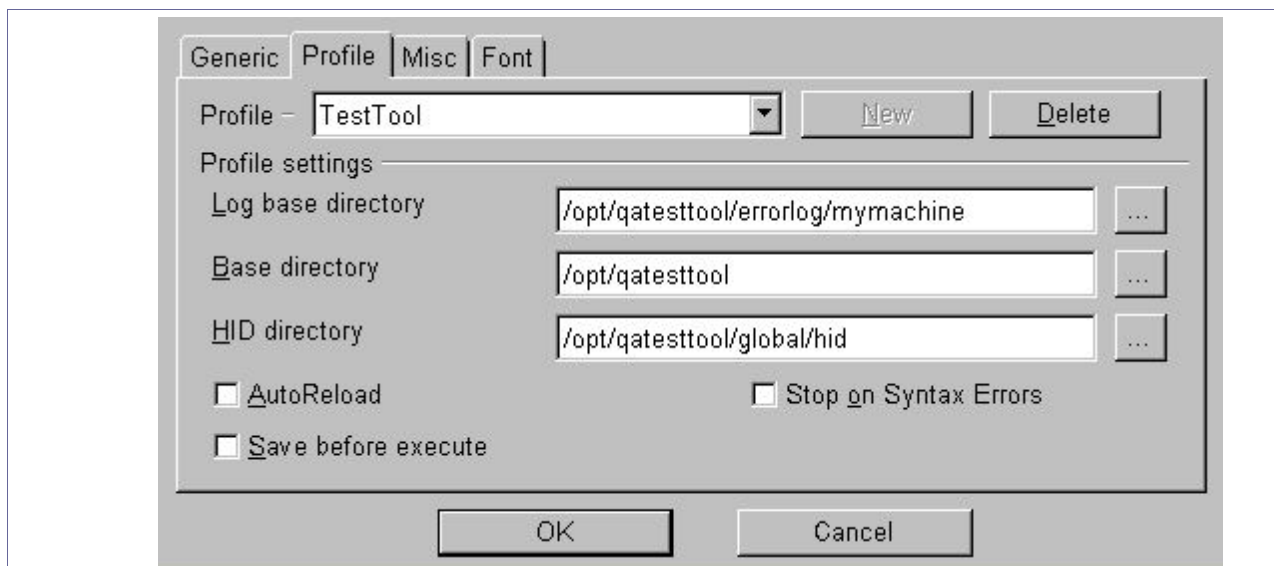
When you run the TestTool executable file, the `testtool.ini` and `.testtoolrc` control files are created on Win32 and UNIX, respectively.

If you place a TestTool control file in the directory that contains the TestTool executable file, the control file is automatically copied to the *profile* directory on Win32, and the user's *home* directory on UNIX.

**Linux only:** If you get a message similar to "Couldn't bootstrap uno servicemanager for reason : Couldn't create registry file ..services.rdb for writing", ensure that you are using 'nfslock'. For example, on Linux run `'/etc/init.d/nfslock status'`. If it is not running, switch to root and run `'/etc/init.d/nfslock start'` and `'chkconfig nfslock on'`. This is necessary for the TurboLinux and the SuSe 8.1 Linux distributions.

## 1.4 Setting up the TestTool

Choose **Extra - Settings - Profile**.



1. **Profile:** Enter a name for your profile.
2. **Log base directory:** Enter the path where you want to save the test results.  
For example: `/opt/qatesttool/errorlog/mymachine` or `d:\qatesttool\errorlog\mymachine`
3. **Base directory:** Enter the *root path* for the *TestTool Environment* (**without a slash at the end!**)  
For example: `/opt/qatesttool` or `d:\qatesttool`
4. **HID directory:** Enter the path to the *hid.lst* for the *TestTool Environment*  
For example: `/opt/qatesttool/global/hid` or `d:\qatesttool\global\hid`



If you want, you can also edit these settings in the .testtoolrc / testtool.ini files.

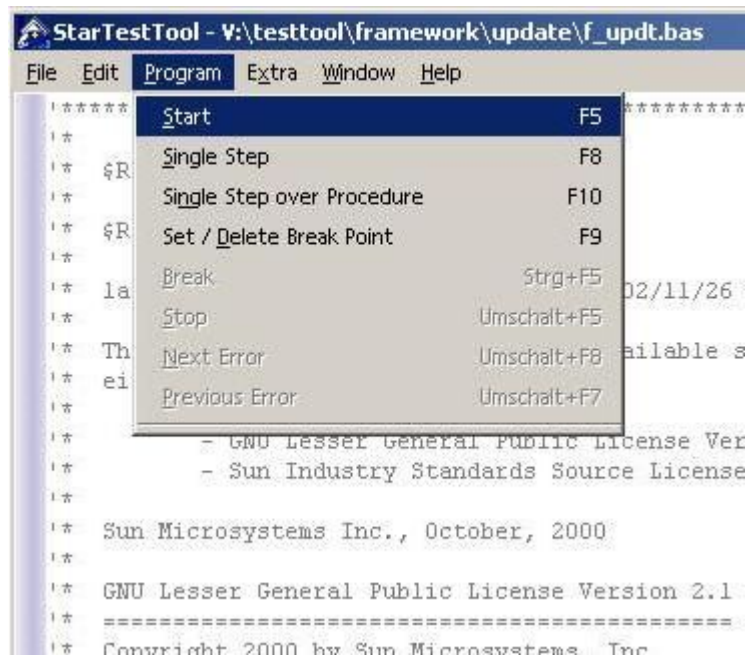
```
CurrentProfile=_profile_TestTool  
(...  
[_profile_TestTool]  
LogBaseDir=/opt/qatesttool/errorlog/extern  
BaseDir=/opt/qatesttool  
HIDDir=/opt/qatesttool/global/hid
```

## 1.5 Adding the TestTool Library to an OpenOffice.org Installation

Before you can run the TestTool scripts, you need to add a special library to the **[[Drive:]] {OpenOffice.org}/program**-directory of the OpenOffice.org installation that you want to test. On UNIX, the library is *libsts\** (For example: *libsts644ss.so* for a Solaris SPARC library) and on Win32, the library is *sts\*\*\*mi.dll*, where \*\*\* corresponds to the major source number of the OpenOffice.org installation (For example: *sts644mi.dll*).

## 1.6 Starting a TestTool Script

Start local OpenOffice.org installations using the TestTool command line parameter *-enableautomation*. In the *TestTool Environment*, executable scripts use the *\*.bas* extension. To run a script, load a *\*.bas*-file, and then run the script from the menu or with a shortcut.



**F5**

### Start

Starts the script, opens a new or old result file, and inserts the result of the test in the file.

**F8**

### Single stepping

Runs the script by single steps.

**F10**

### Single step over procedures

Runs the script one sub routine at a time.

**F9**

### Set / Select a break point

Creates a breakpoint (red circle) in the script on the blue border on the left side.

**↑**

**F5**

### Cancel

Cancels a running script.

**ctrl**

**F5**

### Interrupt

Interrupts a running script.

**↑**

**F8**

### Next Error

Jumps to the next error after a syntax error in a script.

**↑**

**F7**

### Previous Error

Jumps to the previous error in a script.

## 1.7 TestTool Editor

The TestTool uses syntax highlighting for all BASIC, StarBasic, and internal TestTool commands (For more information, see chapter 5: *Internal Commands, Methods and Functions for TestTool*). Highlighting can increase the load time for large files.

## 1.8 Result File

The TestTool automatically saves a result file after a test script is run. The name of the result file is the name of the script \*.bas-file that generated the result, but with the \*.res extension (For example: *first.bas* → *first.res*). To set the path for saving the result file, choose **Extra - Settings - Profile** and enter the path in the **Log Base directory** box.

If a result file already exists, the file is opened and the new result is added at the beginning of the file.

The results are presented as a hierarchical tree list where you can click plus sign (+) or minus sign (-) to expand or collapse the outputs.

- The first output is *Reading the files*. Expand this entry to see the declaration files. If the entry is orange, the declaration files (\*.win / \*.sid) contain errors. The sum of the errors is displayed at the end of a test as *Warnings occurred during initialization*.

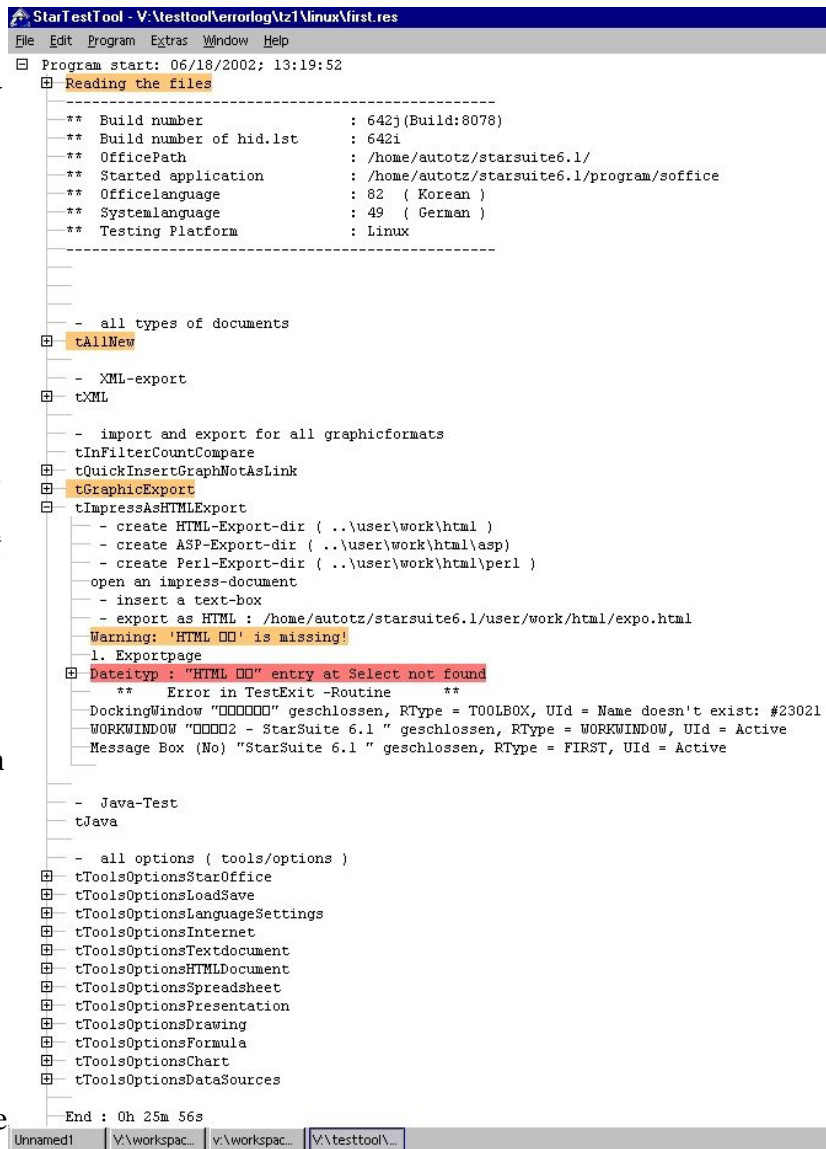
- The next output is a short entry that is generated automatically by the *start up routines*. The entry displays information about the TestTool version, the paths where OpenOffice.org is installed, and where the application was started. The office- and the system language is also displayed.

- Each entry that has a [+] in front of it is a *testcase*. If the *testcase* entry is a color other than black, the test contains errors.

- An **orange testcase** indicates that one or more warnings are present. To see where the error occurred, expand the entry and examine the *warnlogs*, which are the outputs from the developer of the test script. To jump directly to the place in the test script where a warning occurred, double-click the warning.

- A **light red testcase** (not shown in example) indicates a *QAErrorLog* entry that was inserted by the developer of the testcase to give the tester some information (for example, about an existing bug or workaround)

- A **red testcase** indicates that one or more errors are present. The tree is expanded to display the error(s). In the example result file, the error occurred when the TestTool did



```
StarTestTool - V:\testtool\errorlog\tz1\linux\first.res
File Edit Program Extras Window Help
Program start: 06/18/2002; 13:19:52
+ Reading the files
-----
** Build number          : 642j (Build:8078)
** Build number of hid.lst : 642i
** OfficePath            : /home/autotz/starsuite6.1/
** Started application    : /home/autotz/starsuite6.1/program/soffice
** Officelanguage         : 82 ( Korean )
** Systemlanguage        : 49 ( German )
** Testing Platform      : Linux
-----

- all types of documents
+ tAllNew
- XML-export
+ tXML
- import and export for all graphicformats
tInFilterCountCompare
+ tQuickInsertGraphNotAsLink
+ tGraphicExport
+ tImpressAsHTMLExport
- create HTML-Export-dir ( ..\user\work\html )
- create ASP-Export-dir ( ..\user\work\html\asp)
- create Perl-Export-dir ( ..\user\work\html\perl )
open an impress-document
- insert a text-box
- export as HTML : /home/autotz/starsuite6.1/user/work/html/expo.html
Warning: 'HTML <<>>' is missing!
1. Exportpage
+ Dateityp : "HTML <<>>" entry at Select not found
** Error in TestExit -Routine **
DockingWindow "000000" geschlossen, RType = T00LBOX, UID = Name doesn't exist: #23021
WORKWINDOW "00002 - StarSuite 6.1 " geschlossen, RType = WORKWINDOW, UID = Active
Message Box (No) "StarSuite 6.1 " geschlossen, RType = FIRST, UID = Active

- Java-Test
tJava

- all options ( tools/options )
+ tToolsOptionsStarOffice
+ tToolsOptionsLoadSave
+ tToolsOptionsLanguageSettings
+ tToolsOptionsInternet
+ tToolsOptionsTextdocument
+ tToolsOptionsHTMLDocument
+ tToolsOptionsSpreadsheet
+ tToolsOptionsPresentation
+ tToolsOptionsDrawing
+ tToolsOptionsFormula
+ tToolsOptionsChart
+ tToolsOptionsDataSources

End : 0h 25m 56s
Unnamed1 | V:\workspac... | V:\workspac... | V:\testtool\...
```

not find the 'HTML ...' entry in the file-export-dialog (the squares indicate Korean characters that cannot be displayed because the corresponding font is not installed.). The error occurred because the name of the export filter was changed in the OpenOffice.org version that was tested.

- When you expand a red entry by double-clicking the plus sign (+) in front of the entry, a level by level (sub main → *testcase* ... → subroutine ... → .....) view of the error is displayed. You can also see what the TestTool did to return OpenOffice.org to a defined *base state* so that the TestTool can run the next *testcase* without any open windows or dialogs.

For example:

- a *DockingWindow* is closed (geschlossen = closed) with *ID 23021*
- the *WORKINGWINDOW*"... 6.1' is closed
- and a *Message Box* is closed with *NO*
  - Explanation:
    - the *navigator-window* is closed
    - the *working document* is closed

However, since changes were made to the document, a *message box* opened and had to be closed by clicking **No**.
  - The TestTool then started the next testcase at the *base state of OpenOffice.org* (that is, with one open document and no open dialog).
- At the end of a test, the number of errors and warnings is displayed

## 1.9 Starting a Test Script From the Command Line

To start a test-script from the command line, use the following syntax:

<code>testtool.exe [/port=xxx /host=xxx] [startprogram] [/run]</code>	
<b>/Port</b>	The com port on the machine that the TestTool uses to communicate with OpenOffice.org. If you do not include this parameter, the TestTool looks for relevant information in the <i>.testtoolrc</i> or <i>testtool.ini</i> files.
<b>/Host</b>	The <i>hostname</i> of the machine that contains the OpenOffice.org installation that you want to test. If the installation is on the same machine as the TestTool, enter <b>localhost</b> . If you do not include this parameter, the TestTool looks for relevant information in the <i>.testtoolrc</i> or <i>testtool.ini</i> files.
<b>/Startprogram</b>	The *.bas-file that contains the main-routine.
<b>/run</b>	Runs the test script, writes the result to the result file, and exits the TestTool at the end of the test. If you do not include this parameter, the TestTool remains open at the end of the test.

Example `testtool.exe /port=12481 /host=localhost /  
: opt/qatesttool/writer/update/w_update.bas /run`





The TestTool runs the **w\_update.bas**-test on the local machine through port 12481, and exits the TestTool when the test is over.

## 1.10 Testing a Non-Product Version

You can only test and debug a non-product version of OpenOffice.org on the Win32 platform.

1. Define the files where you want to store the debug-settings for OpenOffice.org by inserting the following section into the **win.ini** file:

```
[SV]
dbgsv=c:\dbgsv.ini
dbgsvlog=c:\dbgsv.log
```

2. In OpenOffice.org, press    , and then select “TestTool” in the **Error** list box.

In the TestTool, assertions are indicated by **yellow and green dots**. The TestTool cannot handle some assertions (for example, *OSL-assertions*). Instead, you must manually reproduce the assertion, or use a *product version* to test OpenOffice.org. To manually reproduce an assertion, select “MessageBox” in the **Error** list box.

## 2 The TestTool Environment

### 2.1 Introduction

The structure of the TestTool Environment was changed after the release of source SRC641 (OpenOffice.org 1.0, StarOffice 6.0 / StarSuite 6.0). The following description of paths is only valid for source versions SRX641 and higher.

The TestTool Environment contains the start scripts, the test scripts, and the libraries (*include files*) that are required to automatically test OpenOffice.org. The TestTool Environment is modular, so that a module exists for each application or area that you want to test in OpenOffice.org. To test a single OpenOffice.org application, such as Writer, you only need the “application module” and the “global” module.

*When you export modules from the cvs, the module names and the path names are the same.*


<i>module name / path name</i>	<i>Description</i>
<i>application modules</i>	
<b>base</b>	Database / Data source functionality in OpenOffice.org
<b>calc</b>	Calc (spreadsheet)
<b>chart</b>	Chart, the functionality of charts in a spreadsheet
<b>graphics</b>	Impress (presentation application) and Draw (drawing application)
<b>math</b>	Math (formula)
<b>setup</b>	Installation
<b>writer</b>	Writer (text document, HTML document, master document)
<b>xml</b>	XML file format for all of the OpenOffice.org applications
<i>general modules</i>	
<b>framework</b>	General functionality for all applications (for example, galleries and extras)
<b>global</b>	All general routines (for example, startup, tooling, declaration files)
<b>errorlog</b>	Output directory for all result files
<b>tools</b>	useful standalone applications (hid.lst perl script)

The following subcategories are defined for the TestTool Environment modules:

<i>module name / path name</i>	<i>Description</i>
<b>update</b>	Resource test: Activates all menu items and opens all dialogs of the tested application.
<b>loadsave</b>	Loads, saves, or exports tests
<b>options</b>	Tests all of the settings in the Tools - Options area of the tested application.
<b>level1</b>	Performs a general functionality test for each feature in the tested application, including each menu item.
<b>level2</b>	Performs intensive tests for separate areas in the tested application
<i>special modules</i>	
<b>input</b>	Includes general input files that are required by all tests
<b>tools</b>	Includes libraries that contain the general subroutines that are required by all test scripts in the module or application.

These are only the defined name, you can also find other subcategories in the paths. Mostly the named of the directory mirror the tested area.

If you find a \*.bas-file in one of these directories you are in a test-module. Those module should include the following directories/files:

<i>module name / path name</i>	<i>Description</i>
<b>*.bas</b>	Indicates executable test scripts. To run the script, open the *.bas file in TestTool, and then press  or choose <b>Program - Start</b> .
<b>inc</b>	Includes the libraries and any associated files that are required for a test (included by <i>use-method</i> in the *.bas-file)
<b>input</b>	Inputs the files that are required for the test.
<b>tools</b>	Includes libraries that have general subroutines that are required by test scripts (*.bas). The libraries must be in the same directory as the test script.



## 2.2 *global*-Module

The *global-directory* is required for each test. The directory contains the main routines for running a test script, including the *hid.lst*, the declaration files that identify menu items, dialogs and controls in OpenOffice.org, and general tooling routines.

<i>directory name</i>	<i>description</i>
<b>hid</b>	Contains the <i>hid.lst</i> file.
<b>input</b>	Contains common files that are required by the TestTool. <ul style="list-style-type: none"><li>• translated, <b>office language dependent</b> default filter names for each application (<i>filters-directory</i>)</li><li>• translated, <b>office language dependent</b> OLE-object names (<i>olenames-dir</i>)</li><li>• files for each graphic format that OpenOffice.org can read (<i>graf_inp-dir</i>)</li></ul>
<b>sid</b>	Includes all SlotID declarations.
<b>system</b>	Contains the routines to start a test script in these inc-files.
<b>tools</b>	Contains general tooling routines in these inc-files (for example, <i>declare.bas</i> ).
<b>update</b>	Contains general resource-test routines in these files (options-dialog and autopilots)
<b>win</b>	Contains all HelpID declarations.

## 2.3 CVS information

The complete TestTool Environment is checked into a CVS-tree under the module name *gatesttool*. You can then check out the module or the *global*-module along with the application or module that you want to test. Since the content (such as global routines and the *hid.lst*) of the *global*-module can change, update the *global*-module regularly.

## 3 Declaration of OpenOffice.org for the TestTool

### 3.1 Introduction

To function properly, the TestTool needs to identify the menu items, dialogs, and controls in OpenOffice.org. The identification of these items for testing is not done automatically, but rather by developers through a process called *Declaration*. OpenOffice.org is nearly fully declared, with each new feature being declared as soon as the feature is added. However, discrepancies exist between the declaration of older and newer features. Older dialogs and controls are declared in German, whereas all new features are declared in English. As the declared names are only variables, you can use this documentation to correctly identify older features.

## 3.2 Types of Declaration

The TestTool can only be used on menu items, windows, dialogs, and controls that have been declared (see also chapter 5: *Internal Commands, Methods and Functions for TestTool*). You can find the declarations in \*.sid and \*.win files in the ../global/sid and ../global/win directories.

There are five types of declarations in OpenOffice.org:

- SlotID** Most menu items in OpenOffice.org have SlotIDs (short integer with max. 5 numbers) that are executed directly to call a function, open a dialog, and so on. However, these IDs cannot be used by the TestTool. Instead you can use the declaration names (*longnames*) for the SlotIDs that are listed in the \*.sid –files contained in the ../global/sid directory. SlotIDs are not automatically generated during the OpenOffice.org process, but rather are assigned by developers when a new menu item is inserted.
- HelpID** Most windows, dialogs, and controls automatically receive HelpIDs during the build process of OpenOffice.org. The HelpID is used to open the correct help topic for dialogs or controls in OpenOffice.org. The TestTool uses HelpIDs to identify an item in OpenOffice.org, but rather . However, HelpIDs cannot be used in a TestTool script. Instead, *speakable names* are declared in the \*.win-files in ../global/win for identification. You can use any of the commands described in chapter 5 (*Internal Commands, Methods and Functions for TestTool*) on the items that are identified by a HelpID. File names that start with an “e” refer to items that are in English, whereas those that do not start with “e” are in German.
- UniqueID** A developer can assign an UniqueID to an item in OpenOffice.org that does not have a HelpID, so that you can refer to the item a TestTool script. Please note that the TestTool internally treats HelpIDs as UniqueIDs.
- UNO-Slot** Some newer menu items in OpenOffice.org (*BASE* or *HELP*) do not use SlotIDs, but rather UNO-slots (*Universal Network Objects*). The UNO-slots (For example, *FileOpen\_uno .uno:Open*<sup>1</sup>) are the declared the same way as SlotIDs and are listed in *e\_all.sid* file in the ../global/sid directory.
- Macro-URL** Some menu items (For example, **File - New - Autopilot**) are Macro-URLs and as a result do not have SlotIDs. At present, the TestTool cannot handle Macro-URLs and a workaround is used (the menus are opened).

---

<sup>1</sup> The name 'Open' is taken from the document: <http://framework.openoffice.org/servlets/ProjectDocumentView?documentID=367> from the column Command

### 3.3 Determining the SlotID of a Menu Item

Before you can determine the SlotID of a menu item in OpenOffice.org, you need to set the `HELP_DEBUG` environment variable to `TRUE`.

1. Do one of the following:

- On the Win32 platform, open a DOS window, type `set HELP_DEBUG=TRUE`, and then press Enter.
- On UNIX, open a terminal window (for example, `xterm`), type `set HELP_DEBUG=TRUE`, and then press Enter.

2. In the same window, type `soffice.exe` (Win32) or `soffice` to start OpenOffice.org.

3. Choose **Help** and ensure that the **Extended Tips** option is selected.

4. Open the menu containing the item that you want to determine the SlotID for and rest the mouse pointer over the item.

The Extended Tip is displayed. To keep the Extended Tip open, press `Ctrl+F2`. The SlotID is displayed at the bottom of Extended Tip window (for example, if you choose **Format - Character** in Writer, the SlotID is displayed as ***swriter – 10296***)

If you declared a SlotID using the method described in the *Declaring a new menu item* section of this chapter and receive an error similar to `': UNO URL "slot:58991" could not be executed: No dispatcher was found.'` when you try to determine the SlotID, you must then use the following method to determine the SlotID: `MenuGetItemId(iNumber as Integer)`.

Example:

For the Calc menu item **Insert - Function list**, where the SlotID is 58991 and the *longname* is `HID_SC_FUNCTIONLIST`, use this testtool-script:

```
• DocumentCalc.UseMenu           ' to access the menu bar
  MenuSelect MenuGetItemId(4)     ' to open the 4th menu Insert
  print MenuGetItemId(11)        ' to get the ID for the 11th entry
from                               '
                                   ' the top including separators
  print MenuGetItemText(MenuGetItemId(11)), MenuGetItemCommand
(MenuGetItemId(11))              ' to verify it is the correct entry
```

This leads to the ID 26248 with the  
longname 'FID\_FUNCTION\_BOX'.

### 3.4 Determining the HelpIDs or UniqueIDs of Controls

Instead of using the SlotID method to determine the HelpIDs or the UniqueIDs of controls and dialogs, you can use the **DisplayHID** TestTool command or the `..\global\tools\declare.bas` script. This script starts OpenOffice.org, but does not reset the application (that is, close open documents). This behavior allows you to open a dialog and then run the `declare.bas` script.

When you run the `declare.bas` script, the the following window opens in **OpenOffice.org**:

To display the **HelpID**, the **resource type number**, and the **title of the dialog (or control)**, hold



down the **Display ID** button ( ). In this example, the **View - Zoom** dialog in a Writer document was opened. The **ID** of the dialog is `10000`, the **resource type number** is `316` and the **title** of the dialog is `Zoom`.

For the same example, the following dialog opens in the **TestTool**:

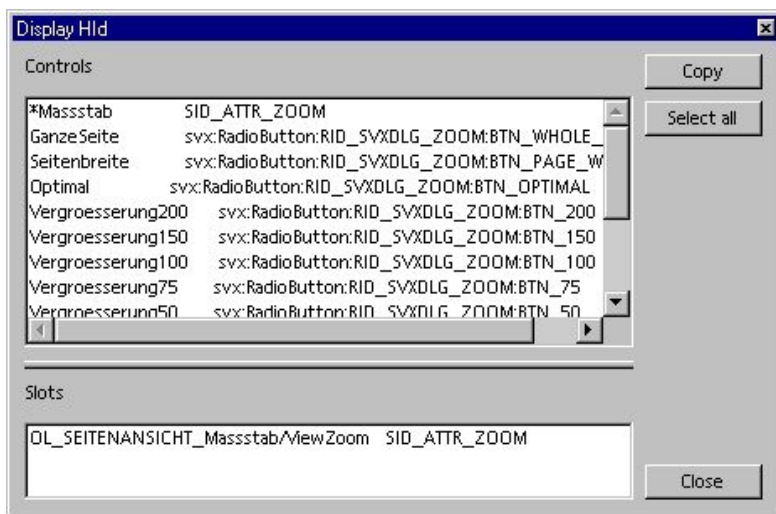


Illustration 1 DisplayHID Window (TestTool) with declaration

**Note:** This example contains German strings in the declaration.

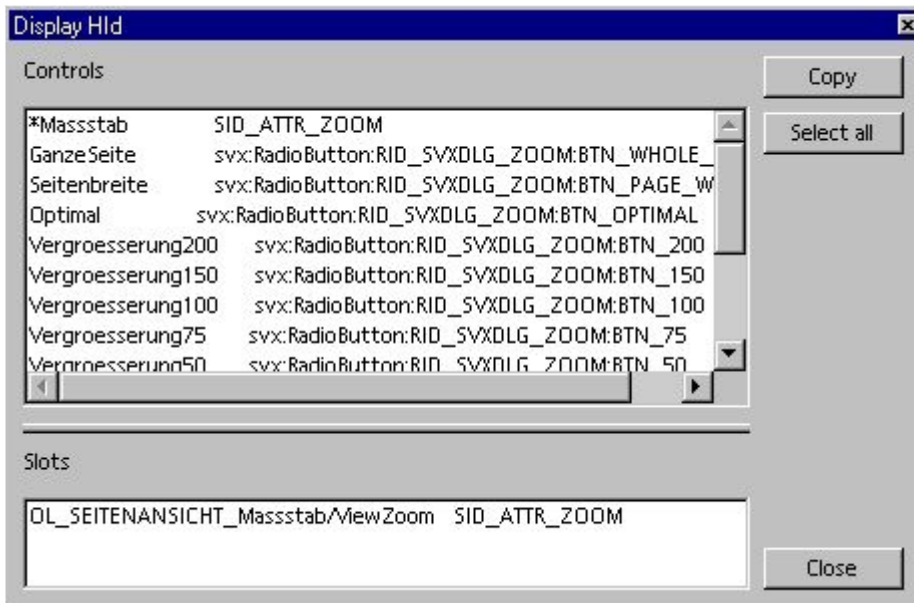
The complete dialog is declared in the TestTool environment. All entries have a name and no HelpID is displayed with the complete help text. If the dialog contains the line `--0:` *EDIT: 90% No entries in hid.lst*, at least one *Listbox* or another control is present in your dialog. A *Listbox* is an edit-field that has a button that you can use to display a list. The list contains one entry with an ID and another entry with a zero.

The *Zoom* dialog in the TestTool Environment is called *Massstab*,

whereas the controls in the dialog are called *GanzeSeite* (=Entire Page), *Seitenbreite* (=Page width), and so on.

These are the same names that are used in the test scripts that work with this dialog. Although the **OK**, **Cancel**, and **Close** buttons are in a global class for dialogs, they are not declared for each dialog. If you want to close a dialog that does not contain one of these *standard buttons*, you will need to find another method to the close the dialog.

In the **Slots** field, you can see which SlotIDs are declared for this dialog, for example, *OL\_Seitenansicht\_Massstab* is a declaration for a button on a toolbar, while *ViewZoom* is the slot that opens this dialog when you choose the menu item **View - Zoom**.



This example is also for the *Zoom* dialog, but without the declaration. The asterisk (\*) in front of the first entry indicates that the dialog is correctly defined in the TestTool Environment. The first column lists the names of the controls and the second column lists the

*Illustration II DisplayHid Window (TestTool) without declaration.*

*resource type* (in this example, radio buttons), the **longname** that was generated automatically by the OpenOffice.org build process, and the HelpIDs (not visible in this dialog). The longnames and HelpIDs are also listed in the *hid.lst* file.

## 3.5 Declaring a New Menu Item


Declaring a new menu item is a four-step process: determining the SlotID of the new menu item (see *Determining the SlotID for a Menu Item* section in this chapter), determining the **longname** using the SlotID, inserting the longname in the *e\_all.sid* file, and restarting the test script. Please note that English menu items are listed in the **e\_all.sid** file and German menu items are listed in the **all.sid** file in the *./global/sid* directory. Only add new menu items to the *e\_all.sid* file.

Example: Adding a new item to the **View – Zoom** menu.

1. Open a DOS or terminal window and type `HELP_DEBUG=TRUE`.
2. Start *soffice.exe*. If the *Quickstarter* is running, exit the Quickstarter.
3. Choose **Help**, and ensure that **Extended Tips** is selected.
4. Open the **View** menu in a Writer document and rest the mouse pointer over the **Zoom** command.  
The name of the application and the SlotID are displayed at the bottom of the Extended Tip window (in this example, *swriter – 10000*).
5. In the *./global/hid/* directory, open the *hid.lst* file and search for the **longname** of the 10000 entry (in this example *1000 SID\_ATTR\_ZOOM*).
6. Select and copy the *longname*.
7. Change to the *./global/sid* directory, open the *e\_all.sid* file, and go to the *View* section.
8. Add a new entry called *ViewZoom*, and paste the *SID\_ATTR\_ZOOM longname* behind the entry.
9. Save the *e\_all.sid* file. The new menu item is added to the TestTool environment.

## 3.6 Declaring a New Dialog or Control

Example: Adding a new dialog or control to the **View - Zoom** menu item:

1. In OpenOffice.org, Choose **View – Zoom**.
2. In the TestTool, open the `../global/tools/declare.bas` script and press  to run the script.
3. In OpenOffice.org, hold down the left mouse button and rest the mouse pointer over the *DisplayHid* dialog.
4. Release the mouse after the dialog is read. The “DisplayHID” dialog appears in TestTool.
5. Select all of the entries in the dialog and press *copy*.
6. In the `../global/win/...` directory, open the `edia_t_z.win` file, where the leading **e** stands for *English* and **dia** stands for *dialog*, and the **t\_z** stands for dialog names that start with *T through Z*. For *tab pages (registers)*, a **tab** is added to the filename instead of **dia**. **a\_d** means only names with leading “**a**” until leading “**d**”)
7. Paste the content of the clipboard into the open file.
8. Add an asterisk (\*) as a separator in front of the dialog name *Zoom* and ensure that all the controls have names.
  - For example:

```
*Zoom SID_ATTR_ZOOM
EntirePage svx:RadioButton:RID_SVXDLG_ZOOM:BTN_WHOLE_PAGE
PageWidth svx:RadioButton:RID_SVXDLG_ZOOM:BTN_PAGE_WIDTH
Optimal svx:RadioButton:RID_SVXDLG_ZOOM:BTN_OPTIMAL
Percent200 svx:RadioButton:RID_SVXDLG_ZOOM:BTN_200
Percent150 svx:RadioButton:RID_SVXDLG_ZOOM:BTN_150
Percent100 svx:RadioButton:RID_SVXDLG_ZOOM:BTN_100
Percent75 svx:RadioButton:RID_SVXDLG_ZOOM:BTN_75
Percent50 svx:RadioButton:RID_SVXDLG_ZOOM:BTN_50
Variable svx:RadioButton:RID_SVXDLG_ZOOM:BTN_USER
ReductionAsVariable svx:MetricField:RID_SVXDLG_ZOOM:ED_USER
```

You cannot use numbers or special characters in the names.

You can, however, use a plus sign (+) to refer to a dialog or a control that you previously declared. In other words, you only need to declare a dialog or a control once. A good example is the **File – Open** Dialog which is also used for **Insert - Graphics**.

- For example:

```
*GeneralFileDialog HID_EXPLORERDLG_FILE
UebergeordneterOrdner
svtools:MenuButton:DLG_SVT_EXPLORERFILE:BTN_EXPLORERFILE_UP
NeuerOrdner svtools:ImageButton:DLG_SVT_EXPLORERFILE:TN_EXPLORERFILE_NEWFOLDER
...
+GrafikEinfuegenDlg GeneralFileDialog
Stil HID_FILEOPEN_IMAGE_TEMPLATE
Verknuepfen HID_FILEDLG_LINK_CB HID_IMPGRF_CB_LINK
Vorschau HID_FILEDLG_PREVIEW_CB HID_IMPGRF_CB_PREVIEW
Link HID_FILEDLG_LINK_CB
Preview HID_FILEDLG_PREVIEW_CB
```



## 3.7 About the *hid.lst* File

The *hid.lst* is a list that is automatically generated during the OpenOffice.org build process. The *hid.lst* contains *longnames*, *SlotIDs*, *HelpIDs*, and *UniqueIDs*.

An inherent problem of the OpenOffice.org declaration method is that IDs can change when a developer, for example, rearranges the order of controls in a dialog or adds a control to a dialog. As a result the same controls on the same dialog in two different versions of OpenOffice.org can have different IDs. However, the *longname* (unique name) of a control is automatically assigned during the build process. The *longname* only changes if the dialog is renamed or if the OpenOffice.org project is renamed.

The version of the *hid.lst* that was by a script is listed in the header for the script in the result file. You only need to use a new *hid.lst* when new features are added to the OpenOffice.org GUI.

### 3.7.1 Location of the *hid.lst* file

The *hid.lst* is located in the program directory of your OpenOffice.org installation.

### 3.7.2 Creating a working *hid.lst*

1. Get the Perl script from CVS at qa/qatesttool/tools/hid.  
Batch files for windows (*hid.bat*) and Unix (*hid.sh*) are found in the same directory.
2. Copy the original *hid.lst* into the same directory (qa/qatesttool/tools/hid/) and run the appropriate batch file.
3. Copy the generated *hid.txt* file to your workspace (qa/qatesttool/global/*hid.lst*).
4. Open the *hid.txt* in a text editor and add the version of the build to the first line of the file (for example, 644m9s1\_HID\_Eigen 010101001010101010101010).
5. Generate a diff to the version stored in CVS. This gives you an indication as to how useable the new *hid.lst* is.

### 3.7.3 Viewing *hid.lst* errors

To view *hid.lst* errors that occur when you run an automated script or *declare.bas*, expand the second line in the resultfile called 'Reading the files'.

There are two possible *hid.lst* errors :

1. The *hid.lst* is corrupted (several errors in the declaration part of the result file).
2. The OpenOffice.org GUI has changed and the declaration is not correct. You have to check the dialogs with *declare.bas* (from qa/qatesttool/global/tools/).

The following error that occurs whenever you run *declare.bas* and you can safely ignore the warning: `'/q_testtool/qatesttool/global/win/setup.win'Warning: Name double: *Active'`

## 4 The Structure of Test Scripts

### 4.1 Introduction

The script language for the TestTool is StarBasic, the OpenOffice.org form of BASIC (similar to VisualBasic). In addition to StarBasic, the TestTool has methods and commands for communicating with and controlling OpenOffice.org.

To automate the TestTool using scripts, use the following guidelines:

- Start a test only when the application that you want to test is in its base state.
- Do not stop the complete test if an error occurs during a testcase.
- Print the results of a test to a file (*=result file*)

### 4.2 sub main

The start routine for a test script is not the same as the simple start routine for a BASIC script. The **sub main** keyword must be written in lowercase, with one space between the two words, otherwise only a normal BASIC script is started. The **sub main** routine makes an internal call to the **LoadIncludeFiles** function and must reside in same \*.bas file as the function.

The **LoadIncludeFiles** function contains the main processes and routines that are required to start a test script:

```
sub LoadIncludeFiles
  use
    "global\system\inc\gv
    ariabl.inc"      'automatic start of LoadIncludeFiles
                    'includes the inc file that contains the most important global
                    'variables (g_variabl.inc)
  use
    "global\system\inc\ma
    ster.inc"       'includes the global library (master.inc)
                    'calls the sub routine to get the information required for a test (sets
                    'global variables, starts the office, ...)
  Call GetUseFiles
  gApplication =
    "WRITER"       'sets the global variable gApplication for the application that you
                    'want to test (here: Writer = Text document )
end sub
```

### 4.3 GetUseFiles

In **GetUseFiles**, the following actions are executed:

- Includes all important *tooling* files
- Loads complete *declaration* ( all \*.sid and \*.win files )
- Gets the *environment information* of the TestTool (for example, platform and system language)
- Gets the *environment information* of the OpenOffice.org (platform, system language, installation type [FAT or network], version, build number, installation path(s), language, and so on)
- Sets all applicable *global variables*
- Adjusts some *important settings* in OpenOffice.org
- Creates the *header in the result file*
- Starts *OpenOffice.org*

The GetUseFiles subroutine was written especially for the TestTool and is contained in the *master.inc* in the *../global/system/inc* directory. The TestTool can only work after this subroutine is

run.

## 4.4 testcase ... endcase

A tested area (testcase) should be as small as possible, for example, one feature or one dialog. This way, if a testcase fails, the remainder of the test script can still be executed. Insert each testcase into a **testcase ... endcase** routine. If an error occurs during the testcase routine, the OpenOffice.org state is recovered to a known state.

When the test script reaches a **testcase** routine, the **testenter** subroutine is automatically called. The **testenter** subroutine checks the base state of OpenOffice.org, automatically closes any open dialogs and all but one document window. The subroutine then copies the associated error messages to the result file.

Similarly, when the test script reaches the **endcase** routine, the **testexit** subroutine is automatically called to check and recover the last known base state of OpenOffice.org. The subroutine then copies the associated error messages to the result file.

If an error occurs during a test process, the testcase stops and jumps directly to the **testexit** subroutine. All errors are written to the result file. If an error occurs outside the **testcase ... endcase** structure, the test script stops executing.

The **testcase ... endcase** construction must be used for each testcase. Use this construction in the same way as a **sub-** or **function-**routine.

The **testenter** and **testexit** routines are located in `../global/system/inc/master.inc`.

**Note:** The words **testcase** and **endcase** are keywords and must be written in lowercase. Do not use these words in documentation strings or in any other strings in the script.

**Note:** Never exit a testcase with 'exit sub', 'return', or with any command that does not call the TestExit sub routine. If you do, then no cleanup is performed. Instead, use `goto endsub` to recover the OpenOffice.org application that you are testing.

## 4.5 try ... catch ... endcatch

To avoid a known OpenOffice.org problem in a test script, do not test the affected area with the test scrip, otherwise use **try-catch-endcatch**.

If an error occurs in the **try-part** of the script, the script automatically jumps to the **catch-part** and not to the recovery section of the testexit routine. If you do have a method to handle the error in the the **catch-part** of the test script, the test breaks and the script automatically jumps to the **testexit-routine** of **endcase**.

You can only branch this method once, that is, you can also use one **try-catch-endcatch** construction in the **try-part** and in the **catch-part**.

```

try
    ...
    try
        ...
    catch
        ...
    endcatch
    ...
catch
    ...
    try
        ...
    catch
        ...
    endcatch
    ...
endcatch

```

Note: *try*, *catch*, and *endcatch* are keywords and as such, cannot be used in documentation strings or as variables in the BASIC-scripts.

Only use the *try-catch-endcatch* construction when you need to circumvent a problem in a long test script. In all other instances, it is better to run into an error and then use the TestTool to recover the test.

## 4.6 sub and function

You can use sub and function routines in test scripts in the same way as in BASIC. You can call the routines from any part of the test script, including *testcases*, and *try-catch-endcatch* constructions.

## 5 Internal Commands, Methods and Functions for TestTool

June 19, 2003

[Testscript Structure](#)

[Global](#)

[Remote Testing](#)

[Windows, Controls and Objects](#)

[Any Control, Window, Object](#)

[CheckBox](#)

[Image-CheckBox](#)

[ComboBox](#)

[Dialog](#)

[ModelessDialog](#)

[DockingWin](#)

[Splitting Windows](#)

[EditWindow](#)

[Edit-Field](#)

[MultiLineEdit-Field](#)

[FloatWin \(Flyer\)](#)

[ListBox](#)

[MultiListBox](#)

[MenuButton](#)

[Menu and Contextmenu](#)

[MessBox / InfoBox](#)

[WarningBox / ErrorBox](#)

[QueryBox](#)

[MoreButton](#)

[PushButton, ImageButton](#)

[RadioButton,  
ImageRadioButton](#)

[SpinField](#)

[PatternField](#)

[NumericField](#)

[MetricField](#)

[CurrencyField](#)

[DateField](#)

[TimeField](#)

[StatusBar](#)

[System Dialog](#)

[TabPage](#)

[Toolbar](#)

[TreeListbox](#)

[BrowseBox](#)

[Control](#)

[TriStateBox](#)



## 5.1 Testscript Structure

<b>Kontext</b> String:Window	Specifies the dialog or window that you want to test. You can use all of the controls in the dialog or windows, so long as you declared the ' <b>Kontext</b> ' in the win-files. Furthermore, if you do not declare the ' <b>Kontext</b> ', you cannot work with the dialog or window. For example, If you want to write to a Writerdocument, you must include the following: <b>Kontext "DokumentWriter"</b> <b>DokumentWriter.TypeKeys "This is a test."</b>
<b>sub main</b> ... ... <b>end sub</b>	Specifies the main routine for the TestTool. Write 'sub main' in lowercase letters with only one space between the two words. As the 'LoadIncludeFiles' routine is called at the start of a test script, the routine must be in the bas-file that contains the 'sub main' routine.
<b>sub LoadIncludeFiles</b> <b>use "inc\gvariabl.inc"</b> <b>use "inc\master.inc"</b> <b>call GetUseFiles ()</b> ... ... <b>end sub</b>	This routine must be included in the bas-file that contains the 'sub main' routine.
<b>sub TestEnter</b> ... ... <b>end sub</b>	The TestEnter routine is located in the master.inc file and is automatically called when the testscript enters a testcase routine.
<b>sub TestExit</b> ... ... <b>end sub</b>	The TestExit routine is located in the master.inc file and is automatically called when a testscript exits a testcase-routine.
<b>testcase</b> .... .... <b>endcase</b>	A special test routine that tests one item, such as a control. The words testcase and endcase must be written in lowercase. If an error occurs in a testcase, the testcase ends and OpenOffice.org is reset to a base state. The error is then written to the resultfile.
<b>try</b> .... <b>catch</b> .... <b>endcatch</b>	If you want to circumvent an error, use the <i>try-catch-endcatch</i> method. If an error occurs during the try-part, the test breaks and jumps to the catch-part. You can only branch this method once, that is, you can add a try..catch..endcatch in the try- or in the catch-part, but only in one level.

## 5.2 Global

<b>ActivateDocument</b> nNr	Activates the document window that has the number that you specify. <u>Note</u> : The list of document windows does not correspond to the order in which the windows are opened, that is, you do not know which document corresponds to which number.
<b>AppAbort</b>	Deletes all communication-instructions in the queue between the TestTool and OpenOffice.org.
<b>AppDelay</b> nNr	Waits in OpenOffice.org (not in TestTool) for the number of milliseconds that you specify.
<i>Boolean</i> <b>ApplicationBusy</b>	Returns 'FALSE' if the OpenOffice.org is in the <i>waitcursor state</i> , otherwise returns 'TRUE'.
<b>Assert</b> [Text]	Outputs an assertion from TestTool.

<b>AutoExecute</b> = Boolean	If AutoExecute = FALSE, all commands behind it are collected, otherwise the commands are executed. You can also use Execute to run the collected commands.
<b>CaptureAssertions</b> [Bool]	To catch the assertions in TestTool, set this variable to TRUE, otherwise set the variable to FALSE to throw the assertions back to OpenOffice.org.
<b>caselog</b> ""	Stops the output at startup (used behind the win-declaration output)
<b>DialogHandler</b> String	Old instruction – N/A
<b>DisplayHid</b> [TRUE]	Call DisplayHid to display a window in OpenOffice.org that lets you select the windows that you want get the UniqueIDs for. To declare windows, use the <b>declare.bas</b> script.
<b>DisplayPercent</b>	A method on Windows for returning information about the mouse position in the window. The position is given as a percentage of the distance between the top left corner of the window (0,0) and the bottom (0,100) and right (100,0) edges of the window.
<b>ErrorLog</b>	Writes the last error to the resultfile (without a callstack).
<b>ExceptLog</b>	Writes the last error to the resultfile with a callstack.
<b>Execute</b>	Executes the collected commands when AutoExecute = FALSE
<i>String</i> <b>GetApplicationPath</b>	Returns the correct path to TestTool.ini / .TestToolrc.
<i>String</i> <b>GetClipboard</b>	Returns the string from the OpenOffice.org clipboard. If the clipboard contains a picture, nothing is returned.
[ <a href="#">tt_env</a> ] <i>String</i> <b>GetClipboardText</b>	Supplies the real string from the clipboard. Use this function to return the information that is selected in a document. This function is not an internal TestTool command, but rather a function from the TestTool Environment.
<i>USHORT</i> <b>GetDocumentCount</b>	Returns the number of open documents in the OpenOffice installation that you are testing.
<i>String</i> <b>GetTestCaseFileName</b>	Returns the file name of the current testcase, otherwise this function returns an empty string. This file name is useful for the statistics output in the recover routine.
<i>Nr</i> <b>GetTestCaseLineNr</b>	Returns the line number of the current testcase.
<i>String</i> <b>GetTestcaseName</b>	Returns the name of the current testcase. Outside a testcase it returns an empty string. The testcase name is useful for the statistics output in the recover routine.
<b>GPF</b>	Crashes OpenOffice.org.
<i>String</i> <b>MakeIniFileName</b> (String BaseName)	Returns the correct name of an <i>ini</i> -file on the current operating platform. For example, MakeIniFileName (“TestTool”) returns TestTool.ini on Windows and .TestToolrc on UNIX.
<b>NoDebug</b> / <b>Debug</b>	Deprecated.
<b>PrintLog</b> String	Writes a string to the resultfile
<b>Profile</b> [Bool][,nNr1][,nNr2][,nNr3][,nNr4][,String]	Old instruction to get system-information (not used in the tests)



<b>RemoteCommandDelay</b> [msec][,msec]   TRUE   FALSE	<ul style="list-style-type: none"> <li>• Delays the execution of OpenOffice.org commands (=remote) (not local BASIC routines).</li> <li>• <u>functions:</u> <ul style="list-style-type: none"> <li>• <i>msec, [msec]</i> is the delay range (for example, if you enter 100,100, the delay is always 100 msec, whereas if you enter 100,200, the delay can be 100, 120, ... 130, ...190, 200). The second value is OPTIONAL.</li> <li>• <i>TRUE   FALSE</i> Boolean expression for enabling or disabling the delay.</li> </ul> </li> <li>• <u>Examples:</u> <ul style="list-style-type: none"> <li>• <i>RemoteCommandDelay 100</i> <i>RemoteCommandDelay TRUE</i></li> <li>• <i>RemoteCommandDelay 100,200</i> <i>RemoteCommandDelay TRUE</i></li> </ul> </li> </ul>
<b>String ResetApplication</b>	Returns OpenOffice.org to its base state and writes the errors to the resultfile. This function can only be used in the recoverroutine (TestEnter, TestExit).
<b>SetClipboard</b> String	Copies a string to the clipboard
<b>Start</b> exe-file [, String2 ]	Starts OpenOffice.org and opens the communication between OpenOffice.org and the TestTool. Use <b>start sAppExe</b> in a script file to specify the OpenOffice.org application that you want to start.
<b>WarnLog</b> String	Writes a warning to the resultfile.
<b>QAErrorLog</b> String	Writes the string that you specify when this function is encountered in a test script or by internal TestTool code ( <u>Example:</u> Extracting an XML package over existing XML files returns a QAError that tells the QA engineer that the directory is not empty). You enable or disabled the logging of these QA errors/notes with <i>EnableQAErrors</i> .
<b>EnableQAErrors</b> = [Bool]	Enables or disables the logging of QAErrorLogs. The default value is TRUE (=on). See the <i>QAErrorLog</i> command.
<b>WinTree</b>	Shows the complete window hierarchy of OpenOffice.org, so long as you set the output to a variable. Use the following structure to show the complete tree: <b>Dim sInterim</b> <b>sInterim = WinTree</b> <b>Printlog sInterim</b>
<b>Use</b> Include file	If you want to use inc-file in a subroutine, you must add the files with the <b>use</b> -instruction. If you want, you can use relative pathnames (relative to TestTool path in TestTool.ini/.TestToolrc) or absolute pathnames.
<b>UseBindings</b> = TRUE <b>UseBindings</b> = FALSE	A test script aborts if you call a menu item slot that requires parameters. In this case, set 'UseBindings=TRUE' before the slot call. If the scripts still crashes, write the bug UseBindings => FALSE in each ResetApplication-Routine in TestEnter and TestExit.
<h3>5.2.1 Statistic</h3>	
<b>GetErrorCount</b>	Returns the matching count of errors in the running test.
<b>GetWarningCount</b>	Returns the matching count of warnings in the running test.
<b>GetUseFileWarningCount</b>	Returns the matching count of warnings out of the declaration part in the running test.

## 5.3 Commands on The Office Side

These commands work in OpenOffice.org applications and not in TestTool. You can also use them for remote testing. For normal file commands, please use BASIC commands.

<i>String</i> <b>app.Dir</b> Filename	You can get the information if a file/directory exists or not. For example: for the files: <b>if app.dir ("d:\officedir\program\soffice.exe") &lt;&gt; "" than print "The file exists!"</b> for directories: <b>if app.dir ("d:\officedir", 16) &lt;&gt; "" than print "the directory exists!"</b>
<b>app.Kill</b> Filename	Deletes a file.
<b>app.rmDir</b> Filename	Deletes a directory (only an empty directory)
<b>app.mkDir</b> Filename	Creates a new directory.
<b>app.FileCopy</b> Old_filename, New_filename	Copies a file.
<b>app.Name</b> Old_filename, New_filename	Renames a file.
<i>nSize</i> <b>app.FileLen</b> Filename	Returns the length of a file.
Date <b>app.FileDateTime</b> Filename	Returns the date that a file was created.
<b>GetSystemLanguage</b>	Returns the ISO language code (Integer) of the system that the OpenOffice.org application is running on.

## 5.4 Windows, Controls and Objects

### 5.4.1 Any Control, Window, Object

<i>String</i> <b>Object.Caption</b>	Returns the name of dialogs (name on the titlebar of a dialog) or controls. For controls that do not have a name, the returnvalue is empty. Only the fixedtext belongs to a control.
<i>Bool</i> <b>Object.Exists</b> [Timeout in Sek.]	Checks to see if a dialog or a control exists.
<i>String</i> <b>Window.GetFixedText</b> [Nr]	Returns the content of the n <sup>th</sup> fixed text. If you do not specify the [Nr] parameter, the first fixed text is returned.
<i>Nr</i> <b>Window.GetFixedTextCount</b>	Returns the number of fixed texts on a given window or control and associated child windows or controls.
<i>Nr</i> <b>Object.GetRT</b>	Returns the resource type of the object, window, or control
<i>Nr</i> <b>Window.GetPosX</b>	Returns the x-position of the edge at the top left (in pixels).
<i>Nr</i> <b>Window.GetPosY</b>	Returns the y-position of the edge at the top left (in pixels).
<i>Nr</i> <b>Window.GetSizeX</b>	Returns the x-size of a dialog (in pixels).
<i>Nr</i> <b>Window.GetSizeY</b>	Returns the y-size of a dialog (in pixels).
<i>Nr</i> <b>Object.ID</b>	Returns the UniqueID of a window or a control.
<i>Bool</i> <b>Object.IsEnabled</b>	Checks if a control is enabled.
<i>Bool</i> <b>Object.IsVisible</b>	Checks if a control or a window is visible.

Objekt. <b>MouseDown</b> x, y [, nButton ] [,Focus]	Presses the mouse button at the specified x/y-position (in percent) in a window. Ensure that you use the <b>mouseup</b> command after each instance of the <b>mousedown</b> command. <b>nButton</b> is: 1 = left mouse button 2 = middle mouse button 3 = right mouse The default (without nButton) is the left mouse button.  If the <b>MouseDown</b> and <b>MouseUp</b> combination does not work like a normal mouse click, then set the Focus parameter to <b>TRUE</b> . (x,y):= (1,1) top left; (99,99) bottom right
Objekt. <b>MouseDoubleClick</b> x, y [, nButton ] [,Focus]	Double-clicks with a mouse button at the x/y-position (in percent) of a window. If <b>MouseDoubleClick</b> does not work like a normal mouse double click, set the Focus parameter to <b>TRUE</b> .
Objekt. <b>MouseMove</b> x, y [, nButton ] [,Focus]	Moves the mouse. To perform a mouse drag and move or drag and copy, add the <b>MouseDown</b> command before the <b>MouseMove</b> .
Objekt. <b>MouseUp</b> x, y [, nButton ] [,Focus]	Releases the mouse button. If the <b>MouseDown</b> and <b>MouseUp</b> commands do not work like a normal mouse click, set the Focus parameter to <b>TRUE</b> .
Autoexecute = FALSE Objekt. <b>MouseDown</b> x, y [, nButton ] [,Focus] Objekt. <b>MouseUp</b> x, y [,nButton] [,Focus] Autoexecute = TRUE	Use this construction for a rapid mouse click. By setting the appropriate <b>gApplication</b> , you can use the following subroutine for every OpenOffice.org document. <b>gMouseClicked (x, y)</b>
[nNr.] <b>GetMouseStyle</b>	Returns the actual state/look of the cursor, from a list of 85 possible values. Some important states: ARROW = 0, TEXT = 3, MOVE = 6 (moving objects), REFHAND = 28 (links/references) and FILL = 31(fillbox).
<i>String</i> Object. <b>Name</b>	Returns the name of the object from the declaration.
<i>Bool</i> Object. <b>NotExists</b> [Timeout in Sek.]	Checks to see if a control or window does not exist.
Objekt. <b>OpenContextMenu</b> [Focus]	Opens the context menu of a window. If you set the Focus parameter to TRUE, the context menu opens at the position of the mouse. To open the context menu at a specific position in a window, you have to first move the mouse to the position before using the OpenContextMenu command.
Object. <b>SnapShot</b> String:Dateiname [PosX, PosY, SizeX, SizeY ]	Takes a screenshot of a control or a window. If you want, you can also specify the position and size of the screenshot in pixels.
Dokumentobject. <b>TypeKeys</b> String [Nr.] [, Focus]	Prints a string on the document, where Nr. is the number of times that you want to print the string. To use special keys, enclose them with "<>". For example: <b>DokumentWriter.TypeKeys "&lt;Shift F1&gt;"</b>  The <b>Focus</b> parameter is a Boolean expression. Normally, typekeys works without this parameter; however, to set the typekeys command directly in this window, set the Focus parameter to 'TRUE'. Use this parameter only when the normal command does not work correctly.

## 5.4.2 CheckBox

## 5.4.3 Image-CheckBox

[ Restype: 336 ], [ Restype: ??? ]

<b>Object.Check</b>	Selects the check box.
<b>Object.Click</b>	Changes to the next state of the check box, that is, removes the check mark from a selected check box.
<i>Bool</i> <b>Object.IsChecked</b>	Returns the state of the check box (TRUE = selected).
<b>Object.UnCheck</b>	Clears the check box.

## 5.4.4 ComboBox

[ Restype: 340 ]

<i>Nr</i> <b>Object.GetItemCount</b>	Returns the numbers of entries in the combo box.
<i>String</i> <b>Object.GetItemText</b> [Nr]	If you do not include the Nr parameter, the text of the selected entry is returned. If you include the Nr parameter, you can return the text of any entry that you specify, including an entry that is not selected.
<i>Nr</i> <b>Object.GetSelCount</b>	Returns the numbers of the entries that are selected in a combo box (You can select more than one entry).
<i>Nr</i> <b>Object.GetSelIndex</b> [Nr]	Returns the index of the selected entry in the list.
<i>String</i> <b>Object.GetSelText</b> [Nr]	Returns the text of the selected entry.
<b>Object.Select</b> Nr	Selects an entry with the index number that you specify.
<b>Object.Select</b> String	Selects the entry that has the same text as the String that you specify.
<b>Object.SetNoSelection</b>	Sets no selection in a list (sometimes this corresponds to the first entry in the list).
<b>Object.SetText</b> [String]	Enters the text string that you specify in a combo box. If the combo box contains an entry that matches the string, the entry is selected.

## 5.4.5 Dialog

## 5.4.6 ModelessDialog

[ Restype: 314 - 316 ]

<b>Dialog.Cancel</b>	Closes a dialog by pressing the Cancel button.
<b>Dialog.Close</b>	Closes a dialog with the Close button.
<b>Dialog.Default</b>	Not important
<b>Dialog.Help</b>	Presses the Help button to open the help topic for the dialog.
<b>Dialog.Move</b> Nr:x, Nr:y	Moves a dialog in OpenOffice.org by the number of pixels that you specify.
<b>Dialog.OK</b>	Closes a dialog with the OK button.

## 5.4.7 DockingWin

[ Restype: 370 ]

Window. <b>Close</b>	Closes a window by pressing the X button in the title bar of the window.
Window. <b>Dock</b>	Docks a window on one edge of the StarDesktop.
Window. <b>Help</b>	Opens the Help viewer.
<i>Bool</i> Window. <b>IsDocked</b>	Returns the docking state.
<i>Bool</i> Window. <b>IsMax</b>	Returns the state of the window (maximize or minimize).
Window. <b>Maximize</b>	Maximizes a window so that the contents of the window are visible.
Window. <b>Minimize</b>	Minimizes a window so that only the title bar of the window is visible.
Window. <b>Move</b> Nr:x, Nr:y	Moves a window.
Window. <b>Size</b> Nr:x, Nr:y	Resizes a window.
Window. <b>Undock</b>	Undocks a window.

### 5.4.7.1 Splitting Windows

(in dockingmode)

<i>Bool</i> Office. <b>IsPin</b> [Align]	Returns the state of the pin.
<i>Bool</i> Office. <b>IsFadeIn</b> [Align]	Returns the state of a window (visible or not).
Office. <b>Pin</b> <i>Bool</i> [Align]	Presses the pin.
Office. <b>FadeOut</b> [Align]	Fades out a window (visible).
Office. <b>FadeIn</b> [Align]	Fades in a window (not visible).

If there is more than one splitting window in a docking window, you must set the following **Align** operators.

- **AlignLeft**
- **AlignRight**
- **AlignTop**
- **AlignBottom**

## 5.4.8 EditWindow

(Documents - the active SDI window)

[ Restype: 312 ]

### Special Note:

The first document is **not enabled** for the TestTool - most of the slot (calls of menu items) and *TypeKeys* commands do not work. This is a feature and not a bug.

Dokumentobject. <b>TypeKeys</b> String [Nr.] [, Focus]	<p>Prints the string that you specify in the document. The Nr. parameter specifies the number of times that you want to print the string. To use special keystrokes, enclose the keystrokes by "&lt;&gt;" brackets. For example: <i>DokumentWriter.TypeKeys "&lt;Shift F1&gt;"</i></p> <p>The <b>Focus</b> parameter is a Boolean expression. In StarSchedule and base documents, you must set this parameter to TRUE, to print the string. Otherwise, only set this parameter to TRUE when the string is not correctly inserted into the document.</p>
<i>Boolean</i> Dokumentobject. <b>HasScrollbar</b> [Align]	If the document has a scrollbar => TRUE

<i>Boolean</i> Documentobject. <b>IsScrollBarEnabled</b> [Align]	If the document has a scrollbar and it is enabled => TRUE
If more than one scrollbar is visible, use the following parameters to define the position of the scrollbar. The default is AlignRight. - <b>AlignLeft</b> - <b>AlignRight</b> - <b>AlignTop</b> - <b>AlignBottom</b>	For example: if (DocumentWriter. <b>IsScrollBarEnabled</b> AlignLeft) = TRUE then ...

### 5.4.9 Edit-Field

### 5.4.10 MultiLineEdit-Field

[ Restype: 338 ], [ Restype: 339 ]

<i>String</i> Object. <b>GetText</b>	Returns the text from the edit field.
Object. <b>SetText</b> String	Enters the text string in the edit field.
Objekt. <b>TypeKeys</b> String	Appends text at the end of the edit field (does not overwrite existing text)
<i>Bool</i> Object. <b>IsWritable</b>	Checks if a control is writable (only for visible Edit- and MultiLine Edit-fields).

### 5.4.11 FloatWin (Flyer)

[ Restype: 313 ]

Window. <b>Close</b>	Closes the window (x-button in the title bar of the window).
Window. <b>Help</b>	Opens the help for the current window (help button).
<i>Bool</i> Window. <b>IsMax</b>	Returns the state of the window (maximize or minimize).
Window. <b>Maximize</b>	Maximizes the window.
Window. <b>Minimize</b>	Minimizes the window so that only the title bar is visible.
Window. <b>Move</b> Nr:x, Nr:y	Moves the window by the number of pixels that you specify.
Window. <b>Size</b> Nr:x, Nr:y	Resizes the window.

### 5.4.12 ListBox

### 5.4.13 MultiListBox

[ Restype: 341 ], [ Restype: 342 ]

<i>Nr</i> Object. <b>GetItemCount</b>	Returns the number of entries in a list box.
<i>String</i> Object. <b>GetItemText</b> [Nr]	Leave out the Nr parameter to return the text of the selected entry. If you include the Nr parameter, you can return the text of any entry in the list box.
<i>Nr</i> Object. <b>GetSelCount</b>	Returns the number of selected entries in a multi-list box (you can select more than one entry).
<i>Nr</i> Object. <b>GetSelIndex</b> [Nr]	Returns the index number of the selected entry in the list.
<i>String</i> Object. <b>GetSelText</b> [Nr]	Returns the text of the selected entry.

<i>Boolean</i> Documentobject. <b>HasScrollbar</b> [Align]	If the document has a scrollbar => TRUE
<i>Boolean</i> Documentobject. <b>IsScrollBarEnabled</b> [Align]	If the document has a scrollbar and it is enabled => TRUE
If more than one scrollbar is visible, you have to use the following parameters to add the position. The default is AlignRight. - <b>AlignLeft</b> - <b>AlignRight</b> - <b>AlignTop</b> - <b>AlignBottom</b>	<u>Example:</u> if (DocumentWriter. <b>IsScrollBarEnabled</b> AlignLeft) = TRUE then ...
Object. <b>Select</b> Nr	Selects an entry and its index.
Object. <b>Select</b> String	Selects the text of an entry.
Object. <b>SetNoSelection</b>	Set no selection in a list (sometimes this corresponds to the first entry).

### 5.4.14 MenuButton

[ Restype: 331 ]

Button. <b>Click</b>	1. Performs the function that is assigned to a menu button, otherwise the menu of the button is not opened. 2. For menu buttons that are accessed through a context menu, use the <a href="#">menu-instructions</a> .
Button. <b>Open</b>	Opens the context menu of the menu button so that you can use the <a href="#">menu-instructions</a> .
Button. <b>OpenMenu</b>	Opens the context menu of the menu button so that you can use the <a href="#">menu-instructions</a> .
<i>Toolbox</i> . <b>OpenMenu</b> ButtonID	Opens the context menu of the menu button in Toolbars so that you can use the <a href="#">menu-instructions</a> .

### 5.4.15 Menu and Context Menu

#### Special notes:

The following instructions can only be carried out on an open context menu or an open sub context menu, otherwise an error is returned. Only use these instructions if you cannot find an equivalent instruction in the *menu.inc* file.

Warning: Make sure that you close each context menu after the instruction has been carried out, otherwise the test encounters a GPF (crashes).

Objekt. <b>OpenContextMenu</b> [Bool]	Opens a context menu for a window or a document. The Boolean parameter in StarSchedule and in base documents must be set to TRUE. The default value is FALSE.
<i>nNr</i> <b>MenuGetItemCount</b>	Returns the numbers of menu entries (including the menu separators)
<i>nID</i> <b>MenuGetItemID</b> <i>nPos</i>	Returns the ID of an menu entry (the ID is required for the remaining menu instructions).
<i>nNr</i> <b>MenuGetItemText</b> <i>nID</i>	Returns the text of the menu entry that has the same <i>nID</i> as the one that you specify (from MenuGetItemID).
<i>nNr</i> <b>MenuGetItemPos</b> <i>nID</i>	Returns the position of the menu entry that has the same <i>nID</i> as the one you specify (from MenuGetItemID).

<i>Bool</i> <b>MenuIsItemChecked</b> nID	Returns the state of a menu entry with a hook (from MenugetItemID).
<i>nNr</i> <b>MenuIsItemEnabled</b> nID	Returns the state “disabled” or “enabled” (from MenugetItemID)
<i>Bool</i> <b>MenuIsSeperator</b> nPos	Returns TRUE if the menu entry is a separator.
<b>MenuSelect</b> nID	Activates the menu item that has the same nID as the one that you specify. If the menu does not automatically close after you activate it, you must close the menu using MenuSelect (0).
<b>MenuSelect (0)</b>	Closes all active menus.
<b>5.4.15.1 special methods</b>	
DocumentWindow. <b>UseMenu</b>	Use this method before you activate a menu in a document. For example: <b><i>Kontext &amp;ldquo;DocumentWriter&amp;rdquo;</i></b> <b><i>DocumentWriter.UseMenu</i></b> After you apply this method, you can then use the normal <a href="#">menu-methods</a> .
<i>String</i> <b>MenuGetItemCommand</b> nItemID	Reads out the command of a MenuItem. This is useful for determining the UNO URLs so you can declare the UNO Slots.
Description: The first menu is the menu bar with the entries (File, Edit, View ...). To open the File-Menu, open the submenu of the first menu entry.	

### 5.4.16 MessBox / InfoBox

### 5.4.17 WarningBox / ErrorBox

### 5.4.18 QueryBox

[ Restype: 304 - 308 ] (MessageBox)  
**(these boxes are handled by the active-operator)**

Active. <b>Cancel</b>	Closes the box with the Cancel button.
Active. <b>Click</b> ButtonID	Not important
Active. <b>Default</b>	Closes the box with the Default button (that has the focus).
Active. <b>GetText</b>	Returns the text from the messagebox.
Active. <b>No</b>	Closes the box with the No button.
Active. <b>OK</b>	Closes the box with the OK button.
Active. <b>Repeat</b>	Closes the box with the Repeat button.
Active. <b>Yes</b>	Closes the box with the Yes button.

#### 5.4.18.1 with checkbox(es)

<i>String</i> Active. <b>GetCheckBoxText</b>	Puts the text of the check box into a string.
<i>BOOL</i> Active. <b>IsChecked</b>	Returns TRUE if the box is checked, otherwise FALSE is returned.
Active. <b>Check</b>	Selects the check box on the message/info/warning/error/query box.
Active. <b>UnCheck</b>	Clears the check box on the message/info/warning/error/query box.

### 5.4.19 MoreButton

**(for additions in a dialog)**

[ Restype: 332 ]



<i>String</i> Object. <b>Caption</b>	Returns the text of the button.
Object. <b>Click</b>	Clicks the button.
Object. <b>Close</b>	Closes the addition window.
<i>Bool</i> Object. <b>IsOpen</b>	Returns the state of the addition window (TRUE == Open).
Object. <b>Open</b>	Opens the addition window.

#### 5.4.20 *PushButton, ImageButton*

[ *Restype: 326* ], [ *Restype: 330* ]

Object. <b>Click</b>	Clicks the button.
----------------------	--------------------

#### 5.4.21 *RadioButton, ImageRadioButton*

[ *Restype: 334* ], [ *Restype: 335* ]

Object. <b>Check</b>	Selects the radio button.
<i>Bool</i> Object. <b>IsChecked</b>	Returns the state of the radio button (TRUE = checked).

#### 5.4.22 *SpinField*

#### 5.4.23 *PatternField*

#### 5.4.24 *NumericField*

#### 5.4.25 *MetricField*

#### 5.4.26 *CurrencyField*

#### 5.4.27 *DateField*

#### 5.4.28 *TimeField*

[ *Restype: 353 - 359* ]

<i>String</i> Object. <b>GetText</b>	Returns the text from the field.
Object. <b>More</b> [ Nr ]	Moves one entry higher.
Object. <b>Less</b> [ Nr ]	Moves one entry lower.
Object. <b>SetText</b> String	Prints text in the field (only in the right format).
Object. <b>ToMax</b>	Goes to the maximum value.
Object. <b>ToMin</b>	Goes to the minimum value.

#### 5.4.29 *StatusBar*

(on edit-windows - documents)

<i>Int</i> <i>DocumentWindow</i> . <b>StatusGetItemCount</b>	Returns the number of items on the status bar.
---	--

<i>Int</i> DocumentWindow. <b>StatusGetItemID</b> nNr	Returns the ItemID of a control on the status bar.
<i>String</i> DocumentWindow. <b>StatusGetText</b> [ ItemID ]	Returns the text of the status bar or of the item that has the ItemID that is not on the status bar.
<i>Bool</i> DocumentWindow. <b>StatusIsProgress</b>	Returns true if the status bar is a progress bar.

### 5.4.30 System Dialog

(system File- or Folder-dialogs - only for Win32)

<i>Boolean</i> <b>ExistsSysDialog</b> (which)	Returns TRUE if a system dialog is open (which = FolderPicker or FilePicker).
<b>CloseSysDialog</b> (which)	Closes the system dialog (which = FolderPicker or FilePicker).
<b>FolderPicker</b>	If you want to handle a system folder dialog, you have to substitute the FolderPicker method for the 'which' statement in the above methods.
<b>FilePicker</b>	If you want to handle a system file dialog, you have to substitute the FilePicker method for the 'which' statement in the above methods.

### 5.4.31 TabPage

[ Restype: 372 ]

for activating a Tabpage with the 'Active'-Operator (normal TabDialogs)	for activating a Tabpage with the TabControl-Operator (only special dialogs)
<i>Kontext</i> Active. <b>SetPage</b> TabZeichen Kontext "TabZeichen"	<i>Kontext</i> "XYZ-Dialog" TabControl. <b>SetPage</b> TabZeichenXYZ <b>Note:</b> Do not use <i>Kontext</i> "Active" before Active.SetPage – this no longer works.
<i>Nr</i> Active. <b>GetPage</b>	Returns the UniqueID for the active tab page.
<i>Nr</i> Active. <b>GetPageCount</b>	Returns the number of tab pages in the dialog.
<i>Nr</i> Active. <b>GetPageId</b> [Nr]	Returns the TabpageID from the dialog. This not the UniqueID and is only needed for the <b>SetPageID</b> instruction.
Active. <b>SetPage</b> TabpageObjekt	Changes to the tab page that you specify.
Active. <b>SetPageId</b> Nr	Changes to the tab page that has the TabpageID that you specify.
Active. <b>SetPageNr</b> Nr	Not important

### 5.4.32 Toolbar

[ Restype: 369 ]

Button. <b>Click</b>	Activates or deactivates a button.
<i>String</i> Button. <b>GetItemText</b> nNr	Returns the text of a button (if available).

### 5.4.32 Toolbar

[ Restype: 369 ]

<i>nNr</i> Button. <b>GetState</b> <i>nNr</i> , <i>nType</i>	Returns the state in dependent on <b><i>nType</i></b> .  <i>nType</i> = 0 => HelpID (0 = Separator) <i>nType</i> = 1 => ItemTyp (1=Button, <i>nType</i> = 2 3=Separator ...) <i>nType</i> = 3 => ItemState (1=pressed, 0=not sonstiges pressed) => ItemID => HelpID
<i>String</i> Toolbar. <b>GetText</b>	Returns the internal name of a toolbar.
Button. <b>TearOff</b>	Tears off a toolbar, but only after a toolbar has been opened by clicking on a button. (You do not need to click click the button a second time)
<i>Nr</i> Leiste. <b>GetItemCount</b>	Returns the number of items in a toolbar (buttons and separators).
<i>String</i> Leiste. <b>GetNextToolbox</b>	Returns the next toolbar if you can change to another toolbar that has a button on it.
Leiste. <b>SetNextToolbox</b> [ <i>String</i> ]	Changes to the next toolbar, or to the toolbar whose name you enter.

### 5.4.33 TreeListBox

### 5.4.34 BrowseBox

### 5.4.35 Control

[ Restype: 376 ]

#### Special Notes (SetControlType):

OpenOffice.org uses two types of list box controls, namely *TreeListBox* or a *Browsebox*. Each list box type can use one of the following list box styles: list boxes with check boxes in front of each entry, list boxes with icons in front of each entry, or list boxes with plus signs “+” or minus signs “-“ in front of each entry. Use the *SetControlType* method to set the control type of a list box in a test routine, otherwise the TestTool produces a GPF (=crash). Following a crash, the *recover routine* resets the control to the default value of *TreeListBox*. Some of the following commands do not work for both types of list box controls.

<b>SetControlType</b> XXX	XXX = <b>CTBrowseBox</b> => Sets the control as a Browsebox (for example, Explorer) XXX = <b>CTTreeListBox</b> => Sets the control as a TreelistBox.
<i>Nr</i> Object. <b>GetItemCount</b>	Returns the number of entries.
<i>String</i> Object. <b>GetItemText</b> <i>Nr</i>	Returns the selected text.
<i>Nr</i> Object. <b>GetSelCount</b>	Returns the number of selected entries.
<i>Nr</i> Object. <b>GetSelIndex</b> [ <i>Nr</i> ]	Returns the index of the selected entry in the list.
<i>String</i> Object. <b>GetSelText</b> [ <i>Nr</i> ] [, <i>Nr</i> ]	<b>GetSelText</b> returns the text of the selected entry. <b>GetSelText 1,2</b> returns the second text of the first selected entry (if the list box has more than one column. <b>GetSelText 2</b> returns the first text of the second selected entry.

### 5.4.33 *TreeListBox*

### 5.4.34 *BrowseBox*

### 5.4.35 *Control*

[ Restype: 376 ]

<i>String</i> Objekt. <b>GetText</b>	Returns the text of the selected entry.
Object. <b>Select</b> Nr [, Bool ]	Selects an entry by number.
Object. <b>Select</b> String	Selects an entry by string.
<b>5.4.35.1 with Checkbox(es)</b>	
<i>Boolean</i> Object. <b>IsChecked</b>	If the check box is selected => TRUE else FALSE
<i>Boolean</i> Object. <b>IsTriState</b>	If the check box is in tri state => TRUE else FALSE
<i>Nr</i> Object. <b>IsGetState</b>	Returns the state of the box: Unchecked == 0 Checked == 1 TriState == 2
Object. <b>Check</b>	Selects the check box.
Object. <b>Uncheck</b>	Clears the check box.
Object. <b>TriState</b>	Sets the check box in tri state.

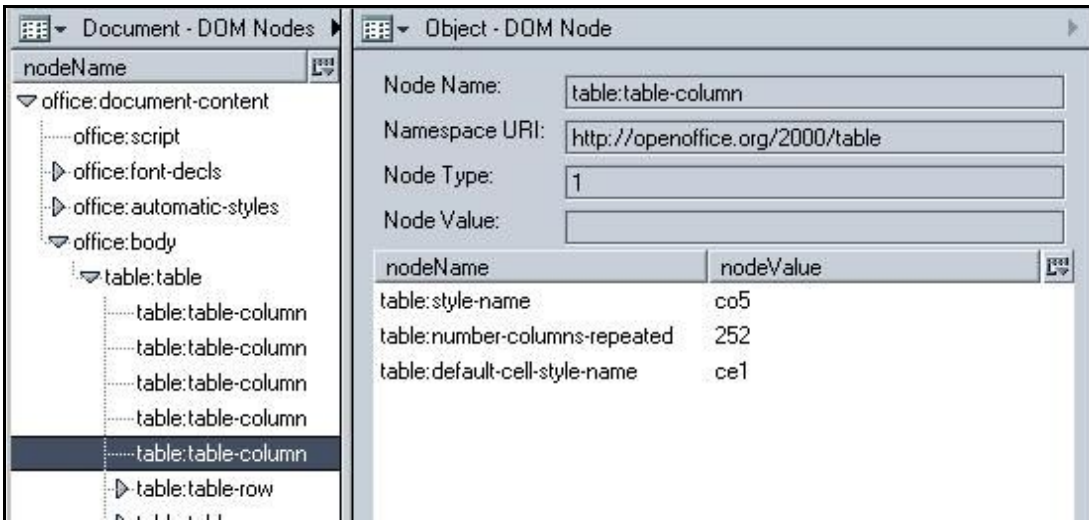
### 5.4.36 *TriStateBox*

[ Restype: 337 ]

Object. <b>Check</b>	Selects the tristatebox.
Object. <b>Click</b>	Changes to the next state of the tristatebox.
<i>Nr</i> Object. <b>GetState</b>	Returns the state of the box: Unchecked == 0 Checked == 1 TriState == 2
<i>Bool</i> Object. <b>IsChecked</b>	Returns the checkstate (TRUE == checked).
<i>Bool</i> Object. <b>IsTriState</b>	Returns the tristate (TRUE == in tristate).
Object. <b>TriState</b>	Puts the object in the tristate.
Object. <b>UnCheck</b>	Clears the tristatebox.

## 6 Working With XML Files

### 6.1 Introduction



In the SAX interface that is included in OpenOffice.org, you can use the XML parser "eXpat" so that you can easily work with XML files in the TestTool environment. The *SAXReadFile* command creates a *DOM* (Document Object Model; a tree like 'view') where you can navigate with TestTool commands.

#### 6.1.1 Main Commands

<b>SAXReadFile</b> <i>filename</i>	<u>Recommended command</u> to open the XML file (or stream). The command <b>checks well-formedness</b> and creates a DOM.
<b>SAXRelease</b>	After TestTool has finished using the XML file, this command closes the DOM.
<i>String</i> <b>SAXCheckWellformed</b> <i>filename</i>	Opens an XML file, checks the well-formedness of the file, and returns any errors in a string. This command does not create a DOM. Open DOMs are closed after this command is executed.

## 6.1.2 Working With Elements, Attributes And Characters

<i>String SAXGetNodeType</i>	<p>The operations that you can perform with nodes depends on the type of node. There are two types of nodes that are also <u>constants</u>:</p> <ol style="list-style-type: none"> <li>1. <i>NodeTypeElement</i> <ol style="list-style-type: none"> <li>a. Includes other nodes and elements (this is the most important function)</li> <li>b. Functions <ul style="list-style-type: none"> <li>• <i>SAXGetElementName</i></li> <li>• <i>SAXGetAttributeCount</i></li> <li>• <i>SAXGetAttributeName</i></li> <li>• <i>SAXGetAttributeValue</i></li> </ul> </li> </ol> </li> <li>2. <i>NodeTypeCharacter</i> <ol style="list-style-type: none"> <li>a. <i>SAXGetChars</i> is the only function that returns the characters from that node.</li> </ol> </li> </ol>
<i>String SAXGetElementName</i>	Returns the name of the XML element at the current 'pointer' location.
<i>String SAXGetChars</i>	Returns the characters from the node type: <i>NodeTypeCharacter</i> .
<i>String SAXGetAttributeValue integer</i>	Returns the value of the n <sup>th</sup> attribute in the recent node.
<i>String SAXGetAttributeValue String</i>	Returns the value of the named attribute in the recent node.
<i>Integer SAXGetAttributeCount</i>	Returns the count of attributes in the recent node.
<i>String SAXGetAttributeName int</i>	Returns the name of the n <sup>th</sup> attribute in the recent node.

## 6.1.3 Navigate And Search in The DOM

<i>Integer SAXGetChildCount</i>	Returns the count of child nodes (XML elements).
<b>SAXSeekElement</b> <i>integer</i>	<p>Searches for the n<sup>th</sup> element in the node.</p> <p>To go to the parent node, set the integer to 0 (zero).</p>
<b>SAXSeekElement</b> <i>string</i>	<p>Searches for the named element in the node. The string can be one of the following:</p> <ul style="list-style-type: none"> <li>• “/” go to root node</li> <li>• “Name” searches in child nodes for named node (element)</li> <li>• “*...” special string from <i>SAXGetElementPath</i></li> </ul>
<i>Boolean SAXHasElement integer</i>	Checks if a child exists at the n <sup>th</sup> node.
<i>Boolean SAXHasElement string</i>	Checks if a child exists at the named node.
<i>String SAXGetElementPath</i>	Returns a special string for later navigation with the <i>SAXSeekElement</i> .

## 7 Typekeys Instruction

### 7.1 Description

This list includes all of the keyboard keys that you can use in the *typekeys* command, provided that you use the following structure:

```
[control].typekeys "<[key(s)]>"
```

Examples:

```
DocumentWriter.TypeKeys "<F11>"
```

This method allows you to simulate keyboard events such as [ESCape] or [ConTRoL] in the

TestTool. For key combinations such as  AND , use the following structure:

```
[control].typekeys "<MOD1 F2>"
```

Ensure that you leave a *blank* between the keyboard keys.

<i>Value</i>	<i>Key</i>	<i>Value</i>	<i>Key</i>	<i>Value</i>	<i>Key</i>
	Characters	Y	Y	UNDO	Undo
0	0	Z	Z	REPEAT	Repeat
1	1	Cursor Keys		FIND	Find
2	2	DOWN	Cursor down	PROPERTIES	Properties
3	3	UP	Cursor up	FRONT	Front
4	4	LEFT	Cursor left	SHIFT	Shift
5	5	RIGHT	Cursor right	MOD1	Ctrl / Strg
6	6	HOME	Pos 1	MOD2	Alt
7	7	END	End	MODTYPE	Alt Gr
8	8	PAGEUP	Page up	F1	F1
9	9	PAGEDOWN	Page down	F2	F2
A	A	Special Keys		F3	F3
B	B	RETURN	Return	F4	F4
C	C	ESCAPE	Escape	F5	F5
D	D	TAB	Tab	F6	F6
E	E	BACKSPACE	Backspace	F7	F7
F	F	SPACE	Space / Blank	F8	F8
G	G	INSERT	Insert	F9	F9
H	H	DELETE	Delete	F10	F10
I	I	Arithmetic Keys		F11	F11
J	J	ADD	+	F12	F12
K	K	SUBTRACT	-	F13	F13
L	L	MULTIPLY	*	F14	F14
M	M	DIVIDE	:	F15	F15
N	N	POINT	.	F16	F16
O	O	COMMA	,	F17	F17
P	P	LESS	<	F18	F18
Q	Q	GREATER	>	F19	F19
R	R	EQUAL	=	F20	F20
S	S	Function Keys		F21	F21
T	T	OPEN	Open	F22	F22
U	U	CUT	Cut	F23	F23
V	V	COPY	Copy	F24	F24
W	W	PASTE	Paste / Insert	F25	F25
X	X			F26	F26



## 8 Resource Types of all Controls and Windows

To see the internal resource type ID for a control or a window (for example, a dialog) in the declaration window, run the ***declare.bas*** script in TestTool. You can then use the ID to identify the correct type of control or window to find the corresponding commands. You can also use ***window.GetRT*** to determine the resource type ID. The ID for *control* (324) is often used in OpenOffice.org and represents a special control, such as a *treelistbox* or a *browsbox*, that was created by a developer. Use trial and error to determine which commands can be used on a control.

<i>resource type</i>	<i>resource-ID</i>	<i>resource type</i>	<i>resource-ID</i>
BASE	256	FIXEDTEXT	343
MESSBOX	304	FIXEDLINE	344
INFOBOX	305	FIXEDBITMAP	345
WARNINGBOX	306	FIXEDIMAGE	346
ERRORBOX	307	GROUPBOX	348
QUERYBOX	308	SCROLLBAR	349
WINDOW	309	SCROLLBARBOX	350
SYSWINDOW	310	SPLITTER	351
WORKWINDOW	311	SPLITWINDOW	352
FLOATINGWINDOW	313	SPINFIELD	353
DIALOG	314	PATTERNFIELD	354
MODELESSDIALOG	315	NUMERICFIELD	355
MODALDIALOG	316	METRICFIELD	356
SYSTEMDIALOG	317	CURRENCYFIELD	357
PATHDIALOG	318	DATEFIELD	358
FILEDIALOG	319	TIMEFIELD	359
PRINTERSETUPDIALOG	320	PATTERNBOX	360
PRINTDIALOG	321	NUMERICBOX	361
COLORDIALOG	322	METRICBOX	362
FONTDIALOG	323	CURRENCYBOX	363
CONTROL	324	DATEBOX	364
BUTTON	325	TIMEBOX	365
PUSHBUTTON	326	LONGCURRENCYFIELD	366
OKBUTTON	327	LONGCURRENCYBOX	367
CANCELBUTTON	328	TOOLBOX	369
HELPBUTTON	329	DOCKINGWINDOW	370
IMAGEBUTTON	330	STATUSBAR	371
MENUBUTTON	331	TABPAGE	372
MOREBUTTON	332	TABCONTROL	373
SPINBUTTON	333	TABDIALOG	374
RADIOBUTTON	334	BORDERWINDOW	375
IMAGERADIOBUTTON	335	BUTTONDIALOG	376
CHECKBOX	336	SYSTEMCHILDWINDOW	377
TRISTATEBOX	337	FIXEDBORDER	378
EDIT	338	SLIDER	379
MULTILINEEDIT	339	MENUBARWINDOW	380
COMBOBOX	340	TREELISTBOX	381
LISTBOX	341	HELPTEXTWINDOW	382
MULTILISTBOX	342	INTROWINDOW	383

Taken from build environment file: 'res\_type'

## 9 All Supported Languages

The following languages are supported by the *OpenOffice.org build process*. To create a language dependent testcase, use the corresponding „Tel.-Code“ number for a *select case* function.

Language	Tel.-Code	Num.-Code	ISO-Code	Asian <sup>2</sup>	CTL <sup>3</sup>
English (US) <sup>4</sup>	01	1033	en-US		
English (GB)	44		en-GB		
German <sup>4</sup>	49	1031	de		
Greek	30	1032	el		
Finnish	35	1035	fi		
Hungarian	36	1038	hu		
Czech	42	1029	cs		
Slovak	43	1051	sk		
Danish	45	1030	da		
Norwegian	47	1044	no		
Brazil (Port.)	55	2070	pt-BR		
Korean	82	1042	ko	*	
Turkish	90	1055	tr		
Arabic	96	1025	ar		*
Hebrew	97	1037	he		*
Russian	07	1049	ru		
Polish	48	1045	pl		
Japanese	81	1041	ja	*	
Chinese (simplified)	86	2052	zh-CN	*	
Chinese (traditional)	88	1028	zh-TW	*	
Portuguese	03	2070	pt		
Dutch	31	1043	nl		
French <sup>4</sup>	33	1036	fr		
Spanish <sup>4</sup>	34	1034	es		
Italian <sup>4</sup>	39	1040	it		
Swedish <sup>4</sup>	46	1053	sv		
Catalan	34c / 37		ca		
Afrikaans	31b		af		
Thai	66		th		*
L10N-Framework	99		ISO_CODE		
Hindi	91		hi-IN		*

<sup>2</sup>Asian: Option in *Tools->Options->Language Settings->Languages* Asian Support checked by default

<sup>3</sup>CTL: Option in *Tools->Options->Language Settings->Languages* CTL Support checked by default

<sup>4</sup>bold: supported European Language for Automated QA Scripts

# 10 The Style and Coding Standard used in the Automated Testing

## 10.1 Revisions

<i>Version</i>	<i>Author</i>	<i>Date</i>	<i>Comment</i>
0.1	Thorsten Bosbach	10/17/2002	Draft
0.2	Joerg Sievers	10/18/2002	Conversion to .sxw
0.3	Thorsten Bosbach	11/21/2002	Update with Feedback
0.4	Joerg Sievers	02/27/2003	Ready for OOo
0.5	Thorsten Bosbach	03/17/2003	Cleaned for OOo

## 10.2 Introduction

This chapter proposes guidelines to make the test code easier to read and to maintain.

There are two sections in this chapter:

- *MUST*: For everyone; especially for people who are not familiar with the (TestTool-)BASIC programming language.
- *ADVANCED*: For people who who want to improve their code style.

The *ADVANCED* section is based on the articles [Gr00], [Gr01], [Gr02], and [Gr03] to help you develop a better layout for your code.

These guidelines are relatively new and have not been applied to all of the test code that has been written. However, any new code should follow these guidelines so that the code can be properly maintained by all.

## 10.3 Section One - MUST

### 10.3.1 Variable

A variable is usually defined with 'DIM name AS type'. Since the TestTool environment uses 'OPTION explicit', a variable must be defined before it is used, otherwise an error occurs.

You also have to declare the variable type and when you define a variable. The variable type is also indicated by a lowercase prefix at the start of the variable name.

Example: for an integer variable:

```
DIM iCounter AS integer
```

<i>prefix</i>	<i>type</i>
i	integer
i	long
d	single
d	double
s	string
b	boolean

The following prefixes must be combined with an additional prefix:

a	array
g	global (from global.inc)
l	list (for variables declared for use of functions from t_list.inc)

Other naming schemes:

t	testcase
s	sub
f	function

After the lowercase prefix that indicates the variable type, start each word in the variable name with an uppercase character. Write CONSTANTS in uppercase characters only, after the lowercase prefix.

You can only make one declaration per line.

For variable and function names, you can only use letters from *a-z/A-Z* and numbers from *0-9*. Do not declare one-letter variables, except for loop counters and array indices such as *i, x, y, j, n*.

Do not use the a lowercase L (*l*) for a variable name as it cannot be distinguished from the number *1* (one) or an uppercase *i* (*I*).

A variable name can have a maximum of 255 characters and must start with a letter. You can use both lowercase and uppercase letters. When declared, numerical values must be set to 0 and strings must be set to "".

### 10.3.2 Indentation

Use horizontal indentation to show the logical level of components in the code, for example, so you can easily see which lines of an if-then-else-clause belong to the 'then' part and which lines belong to the 'else' part. Use four spaces for instead of a tab for an indent as tab handling is different between different platforms and editors. Do not mix spaces and tabs in your code.

Only use four SPACES per indentation level

A label must have a negative indentation.

If the length of a block of code is longer than 20 lines, include a comment as to why this is at the end of the block.

The following is an example of a well-structured (indented) block of code:

```
function fThisIsAnExample() as integer
    dim i as integer
```

```

i = 1
if (i <> 42) then
    bla
    bla
    if (i < 42) then
        bla
        bla
    label:
        bla
        bla
    else
        bla
        ... > more than 20 lines :-
        bla
        bla
    endif ' end of i < 42
    bla
else
    blo
    blo
    for i = 1 to 42
        blu
        blu
        blu
    next i
    blo
endif
end function

```

### 10.3.3 Blank Spaces

Only use *soft spaces* and not Tabulators in the code. Always include ONE SPACE AFTER comma (,) and semi-colon (;) delimiters but not BEFORE the delimiters. For example:

```
DIM i, z, s AS integer
```

The assign operator '=' must have one space before and after it, for example, iNumber = 3

You can either include one space before and after a binary operator (binary: it needs two operands) or leave the spaces out. Note: The inclusion of spaces with binary operators does not affect the mathematical precedence.

```
iState = 3 + 4; iState = 3+4; iState = 3 * 4 + 5; iState = 3*4 + 5
BUT NOT: iState = 3 * 4+4
```

Include a SPACE between a keyword and its opening bracket, but not between a function name and its opening bracket.

### 10.3.4 Comments

Each declaration should include a descriptive comment, so that someone else can easily read your code. Temporary variables (for example, loop counters) are the exception to this rule. You do not need to include a description if the name of the temporary variable indicates the variable type (for example, iTemp). Avoid including information that is likely to become out-of-date. Do not enclose comments in large boxes drawn with asterisks or any other character.

There are three styles of comments: block, single-line, and trailing.

### 10.3.4.1 Block Comments

Block comments describe the contents of a file or the behavior of a function. Include block comments at the beginning and inside of each function.

The block comment that precedes a function describes what the function does, including input parameters, algorithms, and returned values.

For example:

```
function fExample (iExample as integer) as boolean
    '/// ...functional description...           '///'
    '///+ INPUTVALUES:                         '///'
    '///+ affected variables outside of this function: '///'
    '///+ RETURNVALUE:                         '///'
    dim iTest as integer ...
end function
```

### 10.3.4.2 Single Line Comments

Use single line comments to describe loop conditions that are not obvious and also the return values of functions.

Indent a single line comment so that it is at the same indentation level of the code that the comment describes. Include a space between the comment text and the opening ('). For example:

```
if (iTemp > 1) then
    ' Get input file from command line.
    if (iTemp < 9) then
        printlog("can't open %s\n")
    endif
endif
```

### 10.3.4.3 Trailing Comments

Use trailing comments to document declarations. Trailing comments appear on the same line as the code that they describe.

Include enough space to separate a trailing comment from the statement that it describes.

If a block of code includes more than one trailing comment, indent the comments to the same level.

## 10.3.5 File Organization

Separate each code section by a single blank line.

### 10.3.5.1 Sections inside of .inc files

1. A *small* or *big* header
2. *Subs* if they call the following testcases in this file
3. *testcases*
4. *subs*
5. *functions*

### 10.3.5.2 Sections inside of .bas files

1. A *small* or *big* header

2. Declaration of global variables
3. Function *main* with:
  - 3.1. Inclusion of files
  - 3.2. *hStatusIn()* (indicates if the status of the testrun is displayed)
  - 3.3. *Call* of functions
  - 3.4. *hStatusOut()* (indicates if the status of the testrun is displayed).
4. *sub LoadIncludeFiles*

### 10.3.6 Program Organization

#### 10.3.6.1 Sections inside of testcases/subs/functions

1. Declaration of variables. You can only declare variables in this section.
2. Initialization of variables
3. Some code
4. Set the returnvalue for functions
5. Never exit a testcase with *exit* *sub*, *return*, or something else as these do not call the *TestExit* *sub* routine and as a result, no cleanup is performed. Instead, use *goto endsb* to recover the OpenOffice.org application that you are testing.

### 10.3.7 Methods, Hints, Annotations

#### 10.3.7.1 Selecting the same string in different languages

Since most strings are language dependent, there are several ways to select what you want in each language.

1. Use a list that contains the same word in each language that you want to test. You can then use the list with an existing global routine. The routine must be initialized in a *.bas* file, for example:
 

```
global glLocale (15*20) as string
if hSetLocaleStrings ( gTesttoolPath + "graphics\tools\locale_1.txt" , glLocale () ) = FALSE then
    warnlog "Locales file doesn't exist"
endif printlog glLocale (1)
```
2. If the entry is in the same position in all languages, use the number of the entry to select it.
3. Find out if someone knows how to get the language dependent string in another way, for example, by using a name filter.

The decimal separator is also language or locale dependent. Although you can use the *qatesttool/graphics/tools/id\_tools.inc:GetDecimalSeperator* routine to return the separator, you can usually just use integers instead of decimal numbers.

#### 10.3.7.2 Saving files during a test

*Do not save files to a directory in the TestTool environment during a test run!* This can confuse the cvs as well as create a conflict when more than one user running a test tries to save to the same global location. Instead, save the files to the *\${officepath}/user/work* directory. This directory is created when a test is started.

For example:

1. If it does not exist, create the following directory: *../user/work/math/level\_1/export*. On UNIX, use *mkdir* to create the directory, as it generates the entire path.
2. Delete any existing files in the directory.
3. Use *ConvertPath()* to determine the right path separator.

```

dim i as integer
dim sFilter (50) as string
dim lExList(500) as string ' files to be deleted
dim sPath as string      ' filename and path to export

sFilter(0) = 0
sPath = ConvertPath (gOfficePath + "user/work/"+Lcase(gApplication)+"/update/")
if dir (sPath) = "" then
    app.mkdir (sPath)
endif
GetFileList (sPath, "*", lExList())
if (KillFileList (lExList()) <> TRUE) then
    Warnlog "Couldn't delete all in Output-Directory, these files are still there:"
    for i=1 to ListCount (lExList())
        printlog " <> " + lExList(i)
    next i
end if

```

### 10.3.7.3 Using testing levels

One approach to perform tests is to use three testing levels: update, level 1, and level 2.

**Update:** The update level ensures that all elements, including *HelpIDs*, that are required by the test are present in a dialog. This is accomplished by performing actions on the elements. If an element is missing, the test fails. You do not need to check the result of changed elements. The test must be language independent. Close the dialog with CANCEL.

**Level 1:** Performs more actions on the dialog to determine if the function of the dialog has changed. Leave the dialog open with OK. Run the test in at least the 01, 49 installations (where the numbers represent the English and German language codes, respectively), and an Asian language of your choice.

**Level 2:** These are the more complicated tests.

## 10.4 Section Two - ADVANCED

Ensure that the word types that you use in declarations for Boolean subroutines or Functions that have a Boolean returnvalue can be answered with yes/no or true/false, for example, *FisAsianEnabled()*

In an *if* statement, Boolean values do not need to be compared to the values TRUE and FALSE.

In the table below, the two sides are semantically identical.

If (bHasData = TRUE)	If (bHasData)
If (bHasData = FALSE)	If (not bHasData)

Word types are in the declaration of variable names.

Names for Boolean variables should name a property, for example, *bWindowAvailable*.

Names for all other variables should be nouns, for example, *iWindowNumber*.

## 10.5 Appendix

Acronyms used in this file that need an explanation? Examples



### 10.5.0.1 if..then...else

```
if (iNumber = 1) then ____printlog ("snoopy") else ____if (iCount = 1) then _____printlog ("lucy")
____else
_____printlog ("garfield") ____endif endif
```

### 10.5.0.2 select...case

```
select case iNumber ____case 1: printlog ("1") ____case 2: printlog ("2") ____case 3: printlog ("3")
____case else: printlog ("???) end select
```

## 10.6 Bibliography

[Gr00] GRÜNFELDER, Stephan: Quellcode wie aus einem Guss : Codierungsstandards erleichtern Einarbeitung, Wartung und Zusammenarbeit.

In: Elektronik (2001), Nr. 11, S. 59-61

[Gr01] GRÜNFELDER, Stephan; GRIESAUER, Franz: Guter Code braucht Ordnung : Universelle Regeln zur Namensgebung und zu Zahlenformaten in Programmiersprachen.

In: Elektronik (2001), Nr. 18, S. 52-59

[Gr02] GRÜNFELDER, Stephan; GRIESAUER, Franz: Guter Code braucht Ordnung : Teil 2: Das Code-Layout – Übersichtliches Aussehen macht Code besser erfassbar.

In: Elektronik (2002), Nr. 8, S. 74-81

[Gr03] GRÜNFELDER, Stephan: Guter Code braucht Ordnung : Teil 3: Fehlervermeidung bei der Erstellung von C-Programmen.

In: Elektronik (2002), Nr. 14, S. 66-71

# 11 Testcase Documentation in Test Scripts

## 11.1 Description

Documentation for the automated test scripts is created using a separate tool that is currently not open sourced.

The documentation is put into a database and is also used for the *status page feature*. This feature allows you to click on an error and view the documentation for the *testcase* in which the error occurred. You can also insert the number of known errors and warnings in the header of each file. You can then change the state of a test result if you know that a certain bug will not be fixed for the next release.

Documentation can only be inserted in *testcase*, *sub*, and *function* routines. Any documentation that is inserted outside of these routines cannot be parsed by *ttDocs.exe*.

If you want include documentation for a test in the database, you need to use the same standards that were created for *\*.bas-* and *\*.inc* files.

## 11.2 The \*.bas files

Each \*.bas file requires a header, for example:

```
(... cut ...)  
'* The Initial Developer of the Original Code is: Sun Microsystems, Inc.  
'*  
'* Copyright: 2000 by Sun Microsystems, Inc.  
'*  
'* All Rights Reserved.  
'*  
'* Contributor(s): _____  
'*  
'*  
'/*  
'*  
'* owner : owner@foo.bar  
'*  
'* short description : only a short description for this test  
'*  
'\*****
```

Underneath the *SISL/LGPL header* in your file, use a forward slash (/) to start the documentation header. Include the *owner* of this test and a *short description* (max. one line) of the test in the header. End the header with a backward slash (\).

You only need to document the **sub main** routine in the \*.bas-file as this file does not contain any other *testcases*, *functions*, or *sub routines*. In the *sub main routine*, include all the \*.inc-files that you want to use for the test. Do not use a single quote (') at the beginning of a file name to exclude the file from the documentation as this also excludes the file from the test. Thus, if you want to have complete documentation, only use the single quote on calls for the testcases in this routine.

### Example:

```
sub main  
use "framework\options\inc\opt_tool.inc"  
use "framework\options\inc\opt_so_1.inc"  
use "framework\options\inc\opt_so_2.inc"  
use "framework\options\inc\opt_so_3.inc"  
use "framework\options\inc\opt_lan1.inc"  
call hStatusIn ("framework", "f_opt.bas",  
"Test all general settings (tools/options)" )  
call opt_so_1  
call opt_so_2  
' call opt_so_3  
call opt_lan1  
call hStatusOut  
end sub
```

All files that are included with 'use ...' can now be parsed by *ttDocs*. You do not need to include global inc-files. Only include the files that are required for the test.

The test runs *opt\_so\_1*, *opt\_so\_2*, and *opt\_lan1*, but skips over the call to *opt\_so\_3* because of the single quote. However as this is only a call and not an include, the documentation for *opt\_so\_3* remains available.

To determine the dependence between a test (\*.bas file) and the tested area in the application, you need to use some of the following parameters when you call the **hStatusIn** routine:

```
call hStatusIn ('test area', 'name of the bas file', 'short description of the  
test')
```

where test area is: **Base**, **Calc**, **Chart**, **Draw**, **Framework**, **Impress**, **Math**, **Setup**, **Writer** or **XML**.

The documentation only becomes available after the \*.bas file determines the dependence between a test and the tested area (that is, **hStatusOut** returns the information) and inserts the information in the database

## 11.3 The \*.inc-files

Each \*.inc files requires a two-part header. The first part of the header is the same as the header for a \*.bas file, except that it ends with a forward slash (/) instead of a backslash (\). The second part of the header lists the *testcases*, *functions*, and *sub* routines that you want document. The following is an example of a header for a \*.inc file:

```
(... cut ...)  
'* The Initial Developer of the Original Code is: Sun Microsystems, Inc.  
'*  
'* Copyright: 2000 by Sun Microsystems, Inc.  
'*  
'* All Rights Reserved.  
'*  
'* Contributor(s): _____  
'*  
'*  
'/******  
'*  
'* owner : owner@foo.bar  
'*  
'* short description : short description for this inc file  
'*  
'*****  
' **  
' #1 the_first_testcase 'wrn=1|err=0 ' you can insert here a short  
description  
' #1 the_second_testcase  
' #0 the_third_testcase  
' #1 a_sub_routine  
' **  
'\*****
```

Underneath the *SISL/LGPL header* in your file, use a forward slash (/) to start the first part of the documentation *header*. Include the *owner* of this test and a *short description* (max. one line) of the test in the header. End the first part of the header with a forward slash (/).

In the second part of the header, list the routines that you want to document. Add #1 in front of a routine name to enable the documentation for the routine in *ttDocs.exe* and #0 to disable the documentation for the routine. End the header with a backslash (\). Do not use tabs in the header. If you want to include a tab, use spaces instead.

If you want, you can insert the numbers for errors and warnings of each testcase in the header so that the numbers can be used in the *status-page feature* (result pages for all test states). If you know that a testcase produces a warning for a bug that has not been fixed in some versions, use *'wrn=Nr*, where *Nr* is the number of the warning. Similarly, use *err=Nr* for known errors that you want to ignore. The testcase is then displayed in green on the status page, although the warning occurred. Use a pipe (|) separator between *'wrn=.* and *err=.*

## 11.4 Standards for Testcase Documentation

Start a documentation line with a single quote and three forward slashes (///) to identify the line for the documentation parser (ttDocs). If you want, you can end the line with three forward slashes, but this is not required. Only documentation lines that are included in testcases, functions, and sub-routines can be parsed by ttDocs.

When you use /// for documentation strings, the result is a numeration in HTML. If you do not want this to result in a hard-coded line break, use ///+.

Example:

<i>Documentation in script file</i>	<i>Result on HTML-Documentation</i>
<code>'/// This is the 1. line of text [...] '/// This is the 2. line</code>	<ul style="list-style-type: none"><li>. This is the 1. line of text</li><li>. This is the 2. line</li></ul>
<code>'/// This is the 1. line of text [...] '///+ This is the 2. line</code>	<ul style="list-style-type: none"><li>. This is the 1. line of text This is the 2. line</li></ul>

You can use HTML tags to structures your documentation. Anything enclosed by < and > is interpreted as an HTML tag. To insert the < and > characters into the documentation, use `&lt;` for < and `&gt;` for >. For further tips, see <http://selfhtml.teamone.de/selfhtml.htm> (in German) or any other page which describes HTML entities. To insert a single backslash in the documentation output, you need to write **two backslashes** in the documentation source.

## 12 Configuration file entries

The configuration file (UNIX: *.testtoolrc*, Win32: *testtool.ini*) contains several sections. The section name is surrounded by '[' , ']' and contains several entries like: EntryName=Value.

### 12.1 GUI Platform

*Current*= depends on the Platform.

UNIX		Win32	
Subsystem	Current	Subsystem	Current
Solaris SPARC	01	Win 95	100
Solaris x86	02	Win 98	395
Linux	03	Win NT 3	351
		Win NT 4	400
		Win SE	410
		Win ME	490
		Win 2000	500
		Win XP	501

## 13 Alphabetical Index

-enableautomation	10	declaration	22, 26	HID directory	8	Quickstarter	23
/Host	13	Declaration	18	hid.lst	17, 22, 25	recover routine	43
/Port	13	declaration	21	HIDDir	9	ResetApplication	33
/run	13	Declare	23p.	home directory	7p.	Resource test	16
/Startprogram	13	declare.bas	17, 49	hostname	13	resource type	22
.bas	16	dialogs	18p.	hStatusIn	59	Resource Type	49
.bas files	59	Display ID	21	hStatusOut	59	resource type number	21
.GetRT	49	DisplayHid	24, 32	inc	16	resource-test	17
.inc-files	59p.	DisplayHID	21	include files	15	result file	12, 26
.res	12	Document Object Model	45	input	16p.	SAX interface	45
.sid	12, 19	DOM	45	installation path	26	SAXGetElementName	46
.testtoolrc	7pp., 13	e_all.sid	23	installation type	26	SAXGetElementPath	46
.UseMenu	20	Editor	11	Interrupt	10	SAXReadFile	45
.win	12, 19	EnableQAErrors	33	IsChecked	44	SAXRelease	45
.win	24	endcase	27, 31	ISO-Code	50	SAXSeekElement	46
[SV]	14	endcatch	27	IsTriState	44	script language	26
all.sid	23	environment information	26	level1	16	services.rdb	8
application module	15	ErrorLog	32	level2	16	SetClipboard	33
assertions	14	errors	12	libraries	15	SetControlType	43
autopilots	17	ExceptLog	32	ListBox	21	SetPage	42
bas	10	exit a testcase	27, 55	LoadIncludeFiles	26, 31	SetText	36, 38
bas-file	12p.	exports test	16	loadsave	16	sid	17
base state	13	FAT	26	localhost	13	Single stepping	10
BaseDir	9	function	28	Log Base directory	12	SlotID	7, 17, 19, 21, 23, 25
BASIC	11, 26	Function Keys	48	LogBaseDir	9	Slots	21
break point	10	functionality test	16	longname	22p., 25	Special Keys	48
Browsebox	43	g_variabl.inc	26	longname	20	special library	9
build number	26	gApplication	26	longnames	19, 25	special module	16
build process	50	general module	15	Macro-URL	19	StarBasic	7, 11, 26
Cancel	10, 21	GetClipboardText	32	master.inc	26	start routine	26
catch	27	GetItemCount	36, 38, 43	menu item	20	start scripts	15
catch-part	27	GetItemText	36, 38, 42p.	menu items	18p.	start up routines	12
Characters	48	GetPage	42	menu.inc	39	sub	28
Check	41	GetRT	49	MenuItemCommand	20	sub main	26, 59
Click	41	GetSelCount	38	MenuItemId	20	syntax highlighting	11
Close	21	GetSelIndex	38	MenuItemText	20	system	17
command line	13	GetSystemLanguage	34	non-product	14	system language	26
command line parameter	10	GetText	38, 40p.	OK	21	Tabulators	53
context menu	39	GetUseFiles	26	OLE-object	17	TCP/IP-Interface	7
controls	18p.	global	15, 17	OpenContextMenu	39	Tel.-Code	50
CTL	50	global variables	26	OptionsMenu	39	test scripts	15
CurrentProfile	9	goto endsub	27, 55	options	16	testcase	27, 31
Cursor Keys	48	GPF	32, 39	options-dialog	17	testexit-routine	27
CVS	17	GUI Platform	62	OSL-assertions	14	TestTool Environment	15, 17
dbgsv	14	header	26, 59p.	owner	59p.	TestTool Library	9
dbgsv.ini	14	HELP_DEBUG	20, 23	print	20	testtool.exe	7
dbgsv.log	14	HelpID	7, 17, 19, 21, 25	PrintLog	32	testtool.ini	7pp., 13
dbgsvlog	14	hid	17	profile directory	7p.		
debug	14			QAErrorLog	12, 33		
debug-settings	14			qatesttool	17		

title of the dialog (or control)	typekeys	47	update	16p.	win.ini	14
	TypeKeys	37p.	UseMenu	40	window.GetRT	49
tools	unique name	25	VisualBasic	7, 26	windows	19
TreeListbox	UniqueID	7, 19, 25	warnings	12	wrn=Nr	60
try	UNO	19	WarnLog	33	XML parser	45
try-part	UNO-Slot	19	win	17	xterm	20