

Jubula User Manual

Jubula Team

BREDEX GmbH

June 29, 2012



BREDEX GmbH
Mauernstr. 33
38100 Braunschweig
Germany
Tel: +49-531 - 243 30 - 0
Fax: +49-531 - 243 30 - 99
www.bredex.de

GUI*dancer* is a registered trademark of BREDEX GmbH

Title: Jubula User Manual
Author: Jubula Team
File: UserManual
State: RELEASE
Version: V6.0.01011
Released by: BREDEX GmbH
Released at: June 29, 2012

Contents

1	Introduction	7
1.1	Comparison to other testing approaches	7
1.2	How to read this manual	11
2	Samples: example tests	13
2.1	Accessing the prepared Project	13
2.2	The structure of the example Project	14
2.3	Adder Tests	15
2.4	DVD Tool Tests	17
2.5	Meters Tests	18
3	Tasks	21
3.1	Starting and connecting to the AUT Agent . . .	21
3.2	Starting the Integrated Test Environment (ITE) .	25
3.3	Logging into and switching databases	26
3.4	Migrating to newer versions of Jubula	28
3.5	Working with Projects	30
3.6	Defining applications under test (AUT's)	45
3.7	Starting and configuring AUT's	48
3.8	Working with browsers: renaming, deleting, using IDs	58
3.9	Working with editors: opening, adding/delet- ing/rename items, commenting, extracting and replacing, reverting changes	60
3.10	Working with categories in the browsers and editors	68
3.11	Working with Test Cases	71
3.12	Working with test data	80
3.13	Working with component names	102
3.14	Working with Test Suites	110
3.15	Working with Test Jobs to test multiple AUT's .	113
3.16	Information on Test Steps	117
3.17	Working with manual Test Cases	120
3.18	Object mapping	122
3.19	Test execution	136

3.20	Working with test results	141
3.21	Dealing with errors in tests: Event Handlers . .	146
3.22	Preferences	149
3.23	Observing Test Cases	159
3.24	Working with the Problem View	164
3.25	Working with the Teststyle guidelines	165
3.26	Working with Metrics in Jubula	173
3.27	Adapting the user interface	175
3.28	Searching in Jubula	177
3.29	Troubleshooting	183
3.30	Finishing up	190
3.31	Using the test executor for testing from the command line	192
3.32	Using the dbtool client to import, delete and export from the command line	197
3.33	Using Chronon in Jubula	200
3.34	Launch Configurations	205
4	Toolkit-specific information	209
4.1	Testing Swing AUT's	209
4.2	Testing RCP AUT's	210
4.3	Testing GEF AUT's	213
4.4	Testing HTML AUT's	220
5	Best practices	223
5.1	Keyword design – how to structure your tests .	223
5.2	Naming conventions	228
5.3	Using categories to structure your tests	231
5.4	Best practices for using Event Handlers to deal with errors	234
6	User interface	239
6.1	Perspectives	240
6.2	Browsers	248
6.3	Editors	249
6.4	Views	251
6.5	The status bar	257
7	Concepts	259
7.1	Overview	259
7.2	Testing with Jubula	259
7.3	Architecture	261
7.4	Database structure	262
7.5	Approaches to testing	263
7.6	Test hierarchy	265
7.7	Reusability	268

Contents

7.8	Multi-lingual testing	269
7.9	Object mapping	270
7.10	Test execution	271
7.11	Observing user actions	272
7.12	Event Handlers	273
7.13	Extensibility	274
7.14	Summary	274
8	Glossary	275
9	Index	281

Chapter 1

Introduction

Jubula is a tool for the automated testing of Graphical User Interfaces (GUI's) written with Java (Swing, SWT/RCP, GEF) and HTML. The focus of the tool is on testing an application's business logic (workflows, use cases) from the user perspective (functional, black-box, acceptance testing).

Jubula is a keyword-driven tool. Tests are automated by dragging and dropping pre-defined modules (or Test Cases, or keywords) to make sequences of actions for your application. Each Jubula Project contains one or more libraries of these pre-defined modules for you to use. Test automation with these keywords is hierarchical – using the libraries, you can create modules of your own and reuse them to make more complex tests and so on.

Using the keyword-driven approach has various advantages, which are detailed in the next section (→ page 9) .

1.1 Comparison to other testing approaches

Testing is critical to the success of software projects. It is especially important to acceptance test software which is designed for end users.

There are various ways to go about performing acceptance testing, each with their advantages and disadvantages.

1.1.1 Manual Tests

Manual testing, although thorough, cannot keep up with the pace of development. It is impossible to carry out complete continuous integration and regression tests manually.

1.1.2 Programmed Tests

Writing tests in some kind of scripting language is certainly powerful, but it puts a strain on the resources of a team, because the test code itself becomes a project in its own right and also needs to be checked and maintained. The extra costs added by programming GUI tests can be considerable.

Tests written in code also have the problem that they no longer view the software as a black box and may miss important aspects relating to the acceptance. In addition, automation experts (experienced software developers) become the only people who can write or maintain tests. It is generally inadvisable to test your own work, but this is what can happen if testing remains solely in the realm of the developers. Writing tests without coding from the black box perspective not only allows test experts to automate tests (and therefore brings the test perspective to the forefront), but also puts developers in the shoes of the users, which helps to focus and improve the test.

1.1.3 Recorded Tests

Possibly the most popular approach to automated functional testing is macro recording, that is, recording a user's actions for later playback. The appeal of this approach is the apparently quick success that can be seen: a test script can be quickly recorded and played back, giving the impression that automated testing is nothing more than recording a manual test.

However, this approach fails to meet the needs of large or long-term software projects for the following reasons:

- Test specification begins very late in the development cycle, as recording can only begin once the software is available.
- Since only the user action is recorded, checkpoints for verification of test results have to be inserted manually.

- Recorded tests can only test parts of the application which already work.
- There is also the danger that the implementation of the application will be tested, instead of the requirements.
- Recorded scripts are often very large and not particularly well-structured. Making changes at a later point is therefore difficult and requires programming skills, which further increases costs.
- Code generated by recording generally doesn't conform to common software quality attributes such as reliability, stability, portability, maintainability, and usability.

In essence, a recorded script is not an automated test. It must be refactored to remove errors and redundancies, to make its component parts modular and reusable and to insert the intelligence of the manual tester to make the test robust. Once all this has been done, there is probably very little of the original recording left, and a great deal of development work has been done to refactor the script.

1.1.4 The Jubula approach

Jubula makes lets you automate tests which follow the best practices known from software development (readability, modularity and reusability to ensure maintainability), but without any programming effort.

This gives the following advantages:

1.1.5 Early test creation

Jubula tests are created before the AUT is available. This is a radical advantage over capture-replay tools, which force testers to wait until an application is ready to begin with testing. The specification of modular, flexible GUI tests begins early (even as early as at the requirements stage) and continues alongside software development.

The benefit of this is that every version of an AUT can be tested as soon as it becomes available. Testing keeps up with development, so you waste no time in your test process. Earlier testing lets you find issues when they are cheaper and easier to fix and encourages collaboration and communication within the team.

1.1.6 Code-free automation

Tests are automated completely from the user perspective and require no programming effort.

This means that those who understand the user perspective best are able to fully automate tests. There is no need to wait for input (e.g. programmatic creation of test modules) from other team members to automate a test. If developers are writing tests, the black box perspective encourages them to think like a user would when faced with the software. Code-free tests also have the advantage that they are readable by the whole team and also by users or customers.

1.1.7 Manual tester intelligence

The wide range of keywords available in Jubula include high-level actions which can be meaningfully used by testers. There is also a wide range of check and synchronization actions to incorporate the necessary robustness into a test.

1.2 How to read this manual

We have designed Jubula to make testing easier and faster. However, Jubula is a powerful application, and some of its features are quite sophisticated. We therefore recommend having a look at this manual. You can also access the manual from within Jubula by pressing »F1«.

In the reference manual, you will find information on the more technical aspects of working with Jubula, such as a description of the supported components and actions. The reference manual is also shown in the context-sensitive help when you press »F1«.

1.2.1 Layout

This manual is divided into the following chapters:

1. The Chapter "Introduction" (→ page 7) provides a background and introduction to Jubula.
2. The Chapter "Samples: example tests" (→ page 13) exemplifies the capabilities of Jubula, using Jubula Project examples. We recommend this chapter if you learn best by example.
3. The Chapter "Tasks" (→ page 21) provides step-by-step instructions for performing most of the common tasks in Jubula. This chapter is best suited for those who learn by doing.
4. The *Toolkit-specific information* chapter (→ page 209) gives advice and hints on working with AUT's written in the different toolkits supported by Jubula.
5. The Chapter "Best practices" (→ page 223) details how to work with Jubula for successful and easy-to-read test structures and automated test processes.
6. The Chapter "User interface" (→ page 239) presents the Jubula user interface, describing its flexible design. This chapter is useful if you are not yet familiar with the GUI concept of the *Eclipse* platform.
7. The Chapter "Concepts" (→ page 259) introduces the concepts behind Jubula. It will familiarize you with Jubula's structure, components, and functionality. This chapter is

useful (and recommended) for gaining in-depth knowledge of how Jubula works.

8. Finally, the Chapter "Glossary" (→ page 275) provides explanations of all technical terms and abbreviations used throughout this manual.

1.2.2 Conventions Used

Throughout the manual, particular conventions are used to improve readability and clarity:

1.2.2.1 Typesetting Conventions

- Menu paths are shown with a grey background:
`Test → Open`
- Buttons and menu options are italicized and in inverted commas:
"OK"
- Input statements are shown in typewriter format:
`Hello`
- Keyboard shortcuts and commands are in French brackets:
»ENTER«

In addition, important warnings and informative tips are marked as follows:



This is a warning.



This is a tip.

Chapter 2

Samples: example tests

The aim of this chapter is to present several tests highlighting the use of features of Jubula. Three applications under test (AUT's) will be used:

SimpleAdder A simple calculator tool, *Adder*. This tool is available as a Swing AUT, an SWT AUT and as a HTML AUT.

DVDTool A DVD organization tool, *DVD Manager*. With this Swing tool, film categories can be added and browsed, and DVD's and various details about each DVD can be entered.

Meters An RCP AUT to manage meter readings for blocks of flats.

If you want to use these AUT's to try out your own tests, you can find them under *jubula/examples/AUTs*.



2.1 Accessing the prepared Project

You can find the Project containing the tests in the Jubula installation directory under the subdirectory: *examples/projects*.

You can import the Project into the ITE as follows:

1. Select:
Test → **Import**.
2. Browse to the *examples/projects* directory in the Jubula installation directory.

3. Select both the *samples* Project and the *bound_modules_samples* Project.
4. Select "OK" in the *Import Project* dialog.

Once the Projects have been imported, open the *samples* Project. Start and connect to the AUT Agent (→ page 21) , and start the AUT (→ page 136) .

When working on one machine, it may be a good idea to automatically minimize Jubula during test execution. This can be done via :

Window → Preferences .

In the *Test* preferences. Alternatively, the client window can be reduced so that both the client and the AUT can be seen.

2.1.0.1 Result Reports

A result report of the test will be automatically saved into the database when the test has run. The summary of the reports can be viewed in the Test Result Summary View (→ page 142) .

2.2 The structure of the example Project

The Project contains a variety of Test Cases and Test Suites. The Test Cases are grouped according to which AUT they test, and are also further grouped into categories corresponding to individual tests.

2.2.1 The reused Projects

The Project *unbound_modules_concrete* is reused in this Project. This Project is a library of all supported actions in Jubula.

The Project *bound_modules_samples* is also reused. This contains Test Cases which are reused in the samples tests.



We do not recommend making any changes in these Projects.

2.2.2 The categories

Executable Test Cases: This category contains Test Cases which are ready to be executed. Each Test Case corresponds to one Test Suite, and is named accordingly. The executable Test Case contains all the Test Cases necessary for a particular test, and includes all the data.

Tests with the Simple Adder: This category contains the Test Cases which test the Simple Adder program (→ page 15)

Tests with the DVD Tool: This category contains the Test Cases which test the DVD Tool program (→ page 17) .

Tests with the Meters Tool (RCP): This category contains the Test Cases which test the Meters application (→ page 18)

2.3 Adder Tests

2.3.1 Sample 1.1: creating a Test Case from Test Steps

This category contains one Test Case. The Test Case contains four Test Steps, which test an addition in the Simple Adder. The parameter values in the Test Steps have been referenced, and a data set has been added.

This is an example of a test written with Test Steps. However, we recommend using the library of Test Cases to write tests, as shown in the next examples.

2.3.2 Sample 1.2: creating a Test Case using the library

This category contains one Test Case. The Test Case has referenced another Test Case, with four reused Test Cases, which have been reused from the Project *unbound_modules_concrete*.

Press »F6« to find where a particular Test Case was originally specified.



The Test Cases carry out the same steps as in the previous example (→ page 15) . The differences here are:

- The steps to enter a value both reuse the same Test Case, with different referenced parameters, and a different component name.
- The components used in the reused Test Cases are *abstract* components (→ page 108) . This means that the Test Cases are easier to reuse, making tests more robust and maintainable.

This Test Case is reused in the executable Test Case *1.2_SIMPLE_ADDER_TEST*, which is nested in the Test Suite of the same name.

2.3.3 Sample 1.3: using Event Handlers

This category has four subcategories. Each subcategory contains a test which reuses a Test Case to execute a calculation in the Simple Adder which will cause an error. After the error, a reset is carried out.

An Event Handler has been specified in the *bound_modules_samples* Project. The Event Handler has been added to the Test Case, and checks that text in the result field is *jackpot*.

The four tests are as follows:

Continue

The Event Handler in this test has the reentry property *continue*. When the error occurs, the Event Handler is activated. Once the check has been carried out, the test continues, and the reset is performed.

Exit

The Event Handler in this test has the reentry property *exit*. When the error occurs, the Event Handler is activated. Once the check has been carried out, the test finishes. The reset is not performed.

Pause

The Event Handler in this test has the reentry property *pause*. When the error occurs, the Event Handler is activated. Once the check has been carried out, the test pauses. By un-pausing the Test Suite in the client, the test continues.

Retry

The Event Handler in this test is different to the Event Handler

in the other tests. It contains the same steps as the test itself, but the parameter references have been switched. This essentially changes the order in which the numbers are entered into the Simple Adder. The Event Handler has the reentry property *retry*. When the error occurs, the Event Handler is activated. The calculation is redone with the switched values. The failed Test Step (i.e. the original check) is retried, and there is no error. The test is marked as successful.

2.4 DVD Tool Tests

2.4.1 Sample 2.1: testing the menu bar and dialog boxes

This category and the ones following it contain Test Cases which are used in the executable Test Case `2_DVD_TOOL_FULL_TEST` and the Test Suite of the same name.

The Test Case in this category contains a reused Test Case from the *bound_modules_samples* Project. It contains Test Cases to add a category to the DVD Tool via the menu. The parameter for the category name has been referenced, as has a variable to read the category name entered for use later in the test.

The original Test Case in the *bound_modules_samples* Project had no data. When it was reused here, three data sets were added. This means that the Test Case is executed three times, each time entering a different category name, and storing the name as a variable.

2.4.2 Sample 2.2: testing trees

This category contains a Test Case which checks that a tree node exists, based on its textpath. Three data sets have been entered, which use the variables stored in the previous Test Case to complete the textpath.

For example, one of the stored variables was `CATEGORY1`. The textpath to test is entered as `Category/${CATEGORY1}`.

2.4.3 Sample 2.3: testing tables

This category contains a Test Case which contains two Test Cases. The first selects a category node in the tree and then selects the option from the context menu to add a DVD. The second Test Case enters text into the cells in the table. This Test Case has four data sets, so that the four cells can be filled with one Test Case.

2.4.4 Sample 2.4: testing tabbed panes, lists, combo boxes

This category contains four Test Cases. The first two, *Fill in the Content Tab* and *Fill in the Technics Tab* contain Test Cases reused from the *unbound_modules_concrete* Project and the *bound_modules_samples* Project. These Test Cases execute various actions on the tabbed panes in the DVD Tool. There is a lot of reuse of Test Cases, especially on different components. The *Enter a Text Detail* Test Case, for example, is reused three times, with a different component name each time. The *Fill in the Technics Tab* Test Case also contains an Event Handler to check a checkbox if it isn't already done so. Many of the parameters in these Test Cases have been referenced so that values can be entered for them later.

The last two Test Cases each contain a Test Case to select a DVD, and then contain the *Fill in the Content Tab* and *Fill in the Technics Tab* Test Cases. Because the values in these Test Cases were left empty, different data can be entered for each Test Case.

2.5 Meters Tests

The meters application is a piece of software designed to manage meter readings for different flats in a building complex. The building complex is displayed on the left hand side in a tree, which contains nodes showing the flats and their meters.

The most basic use case for this application is to take a meter reading. Taking a meter reading is also a part of the other two use cases shown in this example, moving a tenant in and moving a tenant out of the complex.

This test uses central test data sets to manage its data (→ page 90) .

If you do not have write privileges in the Jubula installation directory, you must create a workspace directory for the meters application and enter this workspace in the AUT configuration for meters. In the *AUT Arguments* field, enter:
`-data <path to workspace>.`

2.5.1 Sample 5: Tests with the Meters tool

The test for the meters tool contains three use cases.

Taking a reading: this use case takes a meter reading for a chosen flat. The actual steps of entering the reading and clicking "OK" or "Next" in the dialog are contained in the Test Case *uc1* as they are reused elsewhere in the test.

Moving a tenant in: this use case selects an empty flat from the hierarchy and selects the option to move a tenant into the flat. It reuses use case 1 because a reading needs to be taken for each of the three meters.

Moving a tenant out: this use case selects a flat and selects the option to move a tenant out. It reuses use case 1 again to enter the readings.

In the *bound_modules_samples* Project, you can see the building blocks that these use cases are made up of in the *AUT bound modules* category.

2.5.1.1 Concatenated parameters

The meters application has a fairly complex tree structure in terms of the data in it. Look in the Test Case *mtr_tre_selectNode* in the *bound_modules_samples* Project to see how the treepath has been concatenated out of the concrete value *Building* and then two further parameters for the location of the meter and the meter number. This makes entering the test data easier.

Chapter 3

Tasks

This chapter provides step-by-step instructions on designing, writing, executing and analyzing tests. For more details about individual tasks, read the Chapter "Concepts" (→ page 259) or Chapter "User interface" (→ page 239) .

3.1 Starting and connecting to the AUT Agent

The AUT Agent is the server component for Jubula. It runs on the same machine as the AUT and allows the ITE to communicate with the AUT and control it during test execution. You must be connected to a running AUT Agent in order to be able to perform object mapping, to observe tests, and to execute tests.

Jubula offers two options for working with an AUT Agent:

- You can start an AUT Agent on a machine (local or remote) and connect to it using the ITE or the Test Executor (→ page 22) .
- You can connect to an embedded AUT Agent that does not have to be started in advance (→ page 23) . This option is only available if you are starting your AUT on the same machine as the ITE, and cannot be used for working with the Test Executor.



If you are working on a local machine, you can also work with the embedded AUT Agent, which does not require you to start an AUT Agent on your machine (→ page 23)

3.1.1 Starting the AUT Agent

You can start the AUT Agent on your local machine (the same machine as the ITE) or on a remote machine in the network.

3.1.1.1 Windows users

1. Start the AUT Agent via the start menu:

Start → Jubula → Start AUT Agent .

2. The AUT Agent is started on port number 60000 unless you enter a different port number as a parameter (→ page 22) .

You can see and stop the AUT Agent in the system tray. Once the AUT Agent is started, you can connect to it from the ITE (→ page 23) .

3.1.1.2 Linux users

Use the script:

```
./autagent (-p <port number>).
```

You can see and stop the AUT Agent in the system tray. Once the AUT Agent is started, you can connect to it from the ITE (→ page 23) .



You must wait for the AUT Agent to be running before you can connect to it.

3.1.1.3 Starting the AUT Agent from the command line: options and parameters

1. Start the AUT Agent from the Windows command line with this command:

```
autagent.exe (-p <port number>)
```

2. If no port number argument is given, the AUT Agent will start on the default port 60000.
3. You can use the following parameters when starting the AUT Agent:

-p: Port number. Enter the port number you wish to start the AUT Agent on.

You can also use the environment variable `"TEST_AUT_AGENT_PORT"` to set the port number that should be used as a default when you start the AUT Agent.



-l: Lenient mode. The lenient mode for the AUT Agent allows AUT's launched by other AUT's to be tested (→ page 114) . You can also change the mode of the currently running AUT Agent via the system tray. Right-click the AUT Agent icon and (de)select *Strict AUT Management* from the menu.



The default mode for the AUT Agent is *strict*.

- v:** Verbose mode. You will see a dialog to tell you whether the AUT Agent has started successfully or not.
- q:** Quiet mode. You will see no dialog if the AUT Agent starts successfully. If the AUT Agent does not start successfully, the error is written to the console.
- h:** Help. Enter this parameter to see a list of options for the AUT Agent.
4. The AUT Agent can be started multiple times – either on the same machine (using a different port number for each instance) or on multiple machines for distributed testing.

3.1.2 Connecting to the AUT Agent

You can connect to an AUT Agent you have started (→ page 22) on a local or remote machine, or you can connect an embedded AUT Agent on your local machine.

You must wait for the AUT Agent to be running before you can connect to it.





connect to AUT
Agent

1. On the toolbar, click on the arrow next to the "Connect to AUT Agent" button.
2. In the drop down list, you can choose an AUT Agent to connect to:
 - Select the embedded AUT Agent option to connect to an AUT Agent that is automatically started for you on your machine.



The embedded AUT Agent uses port 60000 by default. You can change the port that should be used for the embedded AUT Agent in the preferences (→ page 151).

- Select an AUT Agent host and port number to connect to from the list. The list of hosts and ports available in this list can be configured in the preferences (→ page 150) .



If you are not working with the embedded AUT Agent, you must have started an AUT Agent on a machine in the network (→ page 22) to be able to connect to it.



To work with the embedded AUT Agent, you do not need to start an AUT Agent on your machine. However, you can only work with the embedded AUT Agent for local testing (i.e. on the same machine as the ITE). The embedded AUT Agent can only be started by connecting to it from the ITE - the Test Executor and other command line tools cannot start it automatically. The embedded AUT Agent cannot be used if you have written custom extensions for Jubula.

3.2 Starting the Integrated Test Environment (ITE)

3.2.1 Windows Users

1. Start the ITE from the start menu:

Start → Jubula → Jubula.

2. You can also start the ITE using a command line argument:
(Jubula.exe).

3.2.2 Unix Users

Enter Jubula into the shell.

3.2.3 Choosing a workspace

1. When you start the ITE, a dialog appears asking you to choose a workspace.

The workspace is where your preferences are saved.



2. Select the default workspace offered, or locate new one.
3. Select "OK".

3.2.3.1 Deleting the list of used workspaces

To remove workspaces from the drop-down box in the workspaces dialog:

1. Go to:
`%USERPROFILE%\user\`
2. In the Jubula folder, there is a file called *recent-Workspaces.xml* where the list of used workspaces can be deleted.

3.2.4 Help system

1. The information contained within this manual is also available in the ITE.

- In Windows, press »F1« to see context-sensitive help for the current view.
 - In Unix, press »SHIFT+F1«.
 - Select :
`Help` → `Help Content` to search the handbook.
2. The help preferences can be altered via:
`Window` → `Preferences` → `Help`.
 3. See later in this chapter (→ page 156) for details.

3.2.5 Working with the AUT Agent and client on one machine

If the ITE and AUT Agent are installed on one machine, there are a few things to be aware of:

- Don't use the mouse or any keyboard shortcuts during execution. The AUT takes control of the machine it is running on during test execution.
- Make sure that the AUT is visible during test execution:
 - The ITE is minimized by default when test execution begins. Make sure that the AUT is the next window to receive focus when the ITE is minimized.
 - You can change the preference to minimize the ITE via:
`Window` → `Preferences`
If you do this, make sure that the AUT can also be seen on the screen when test execution begins.

3.3 Logging into and switching databases

3.3.1 Logging in to the database

1. Projects are stored in a database. Before you can access a Project, you have to log in to the database.
2. The log-in dialog for the database (Figure 3.1 → page 27) appears automatically the first time you need it (e.g. when you try to open or create a Project).

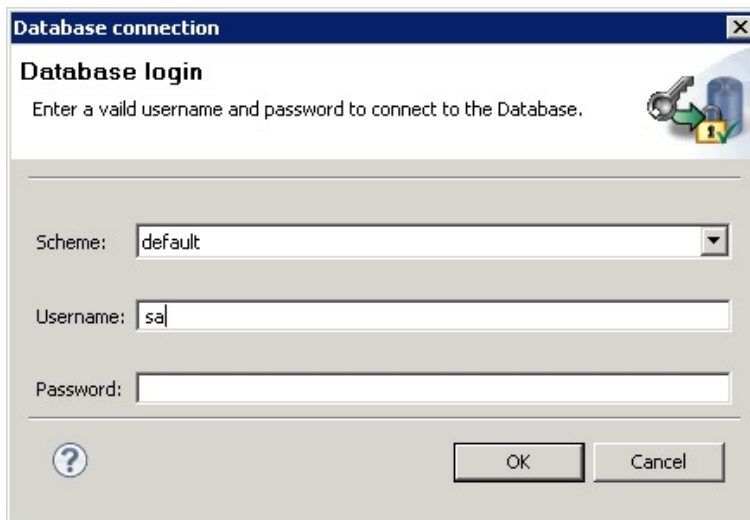


Figure 3.1: Database login

If you are using the default demo (embedded) database, you will automatically be logged into the database.



We do not recommend using the embedded database for productive use of Jubula.



3. Select the database you want to use and enter your username and password.

If you previously had another Jubula version installed, you may see the information that your current database version is not compatible with the latest version. If this is the case, then you can automatically migrate your database (→ page 28) .



3.3.2 Selecting and changing the database connection

1. You can change the database you are connected to by selecting:

Test → Select database .

2. In the dialog that appears, select the database you want to use and enter your username and password.

3.4 Migrating to newer versions of Jubula

If a new version of Jubula uses a new database scheme, you will be informed of this when attempting to connect to the database.

If this is the case, please follow these instructions:

1. From the old version of Jubula, export and backup all the Projects from the database.



Make sure you also export and back up any library Projects used in your tests, e.g. the unbound modules Projects.

2. Back up any extensions you have made to Jubula: any customized plugins and implementation classes you have written.
3. Export your database preferences from the workspace (→ page 154) .
4. Uninstall the old version of Jubula.
5. Clear (empty) the database schema for all necessary Jubula users. You can do this via a database administration tool which will let you carry out the action *Drop Tables*.
6. Install the new version of Jubula.



If you have AUT Agents running on other machines, be sure to install the new version of the AUT Agent there too.

7. Add any extension plugins you backed up from the old version.
8. Import your database preferences into Jubula.

If you are testing RCP AUT's, bear in mind that you will need to remove the old version of the RCP Remote Control plugin from your application and insert the new version in its place. We also recommend starting your application with `-clean` to ensure that the old RCP Remote Remote Control plugin is no longer used.



9. Start Jubula.

You will also need to follow the instructions on updating the version of the unbound modules projects you use (→ page 38)



3.5 Working with Projects

The first step once Jubula has been started is to create, open or import a Project. Once you have done this, you can specify Test Suites, Test Cases and Test Steps.

3.5.1 Creating a new Project

1. From the ITE, select:
`Test` → `New`.
2. If you haven't already logged into the database, a dialog will appear to ask you to do so.
See the previous section (→ page 26) for details.
3. A wizard to create a new Project will appear (Figure 3.2 → page 32).
4. In the *project name* field, enter a meaningful and unique Project name. Do not use any special characters in Project names.
5. Activate the "*reusable project*" checkbox if you want to be able to reference the Test Cases from this Project in other Projects. This will let you use the Test Cases in this Project as the basis or library for other Projects (see the section later for more details on this (→ page 37)).
6. Choose whether this Project should be protected. In a protected Projects, you cannot delete Test Cases or edit parameters for Test Cases. This is only necessary if the Project is reused in another Project – the protection ensures that irreversible changes cannot be made to the reused Project that would adversely affect its dependent Projects.
7. Select the toolkit your Project will use from the combo box. The toolkit is the library used to create the GUI.

Toolkits for Projects

The default toolkit is *concrete*. Choose this if you want to write Test Cases that can be used for different applications (e.g. for Swing and for RCP).

Choose another toolkit if you want to write Test Cases just for a Swing, SWT, RCP or HTML application.

The choice of toolkit you make here will determine what actions are available to you to specify your tests. If you

choose *concrete*, you will not be able to specify tests for components specific to e.g. HTML or SWT.

8. From the list of available languages, select the languages supported by your AUT and move them into the "*project language*" box using the arrow button. Use »CTRL« to select multiple languages.

You will be able to start the AUT in these languages, and translate test data into these languages.

9. Select a default language from the combo box. The default language is the language your Project is started in.
10. You can now click "*Next*" to define an AUT for this Project (→ page 45) .

If you want to define the AUT later, you can click "*Finish*" to create the Project as it is. You can define an AUT later via the Project properties under **Test** → **Properties** .

You can also edit the Project details later in the Project properties dialog (→ page 33) .

New Project wizard

Project properties
Define a new Project.

Project name:

Is Reusable: ☐

Is Protected: ☐

Toolkit for Test Specification:

Select the languages of the test data for the project:

Available languages:

- Albanian (Albania)
- Arabic (Algeria)
- Arabic (Bahrain)
- Arabic (Egypt)
- Arabic (Iraq)
- Arabic (Jordan)
- Arabic (Kuwait)
- Arabic (Lebanon)
- Arabic (Libya)
- Arabic (Morocco)
- Arabic (Oman)
- Arabic (Qatar)
- Arabic (Saudi Arabia)
- Arabic (Sudan)
- Arabic (Syria)
- Arabic (Tunisia)

Project languages:

- English (United States)

Select the default language for this Project:

Click "Next >" to define an AUT now, or click "Finish" to create the Project and add an AUT later via "Project > Properties"

Help < Back Next > Finish Cancel

Figure 3.2: New Project Dialog

3.5.2 Editing the Project and AUT properties

You can open the Project properties dialog (Figure 3.3 → page 33) via:

Test → Properties

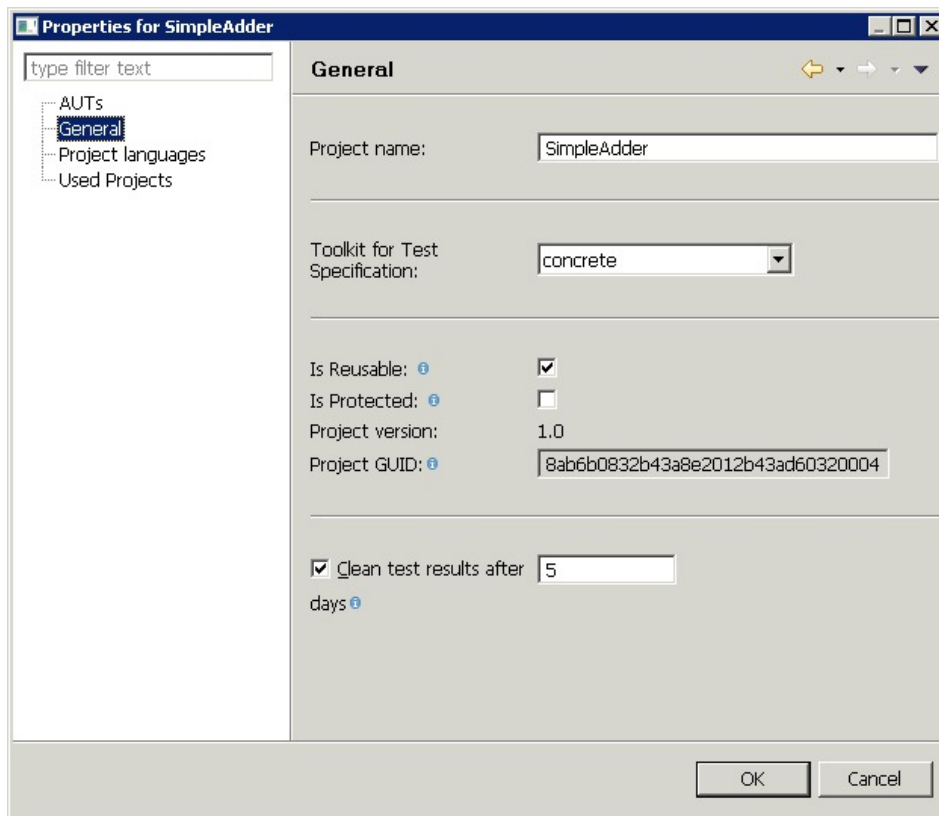


Figure 3.3: Project Properties Dialog

The Project properties dialog lets you see and, in some cases, edit information about:

1. the Project in general (→ page 33) .
2. the Project languages (→ page 34) .
3. the AUT's (→ page 35) .
4. the Projects that you have reused in this Project (→ page 37) .

3.5.2.1 Editing general Project properties

Select "General/" from the tree on the left-hand side of the Project properties dialog. In the screen that appears, you can:

1. Edit the Project name.
2. Edit the toolkit used by the Project (see the following section (→ page 34) for details on this).
3. Edit whether the Project can be referenced (reused) in other Projects.
4. Edit the protected status of the Project. A protected Project does not allow deletion of Test Cases or editing of parameters.
5. See the Project version. This is useful if you have more than one version of a Project.
6. See the GUID (global unit identification). This is a unique ID for the Project.
7. Specify how often the full test result details (→ page 143) should be automatically deleted from the database.

3.5.2.2 Changing the toolkit settings for a Project

If you want to change the toolkit of a Project in the Project properties dialog, the following rules apply:

1. You can change at any time from the *concrete* toolkits to a more specific toolkit (e.g. RCP, HTML).
2. If your previous choice of toolkit was RCP, SWT or HTML, you can only change to another toolkit if your Project does not use any components specific to the originally chosen toolkit.

3.5.2.3 Editing the languages for a Project

To see and edit the Project languages, select "*Project languages*" from the tree on the left-hand side of the Project properties dialog.

You can add and remove Project languages and change the default language in this screen.



You can't delete a language which is being used as an AUT language.

3.5.2.4 Editing the AUT's in a Project

To see and edit your AUT's for a Project, select "AUTs" from the tree on the left-hand side of the Project properties dialog. In the screen that appears, you can see any AUT's you have already added to the Project (Figure 3.4 → page 35). You can choose to edit them, delete them or add a new AUT.

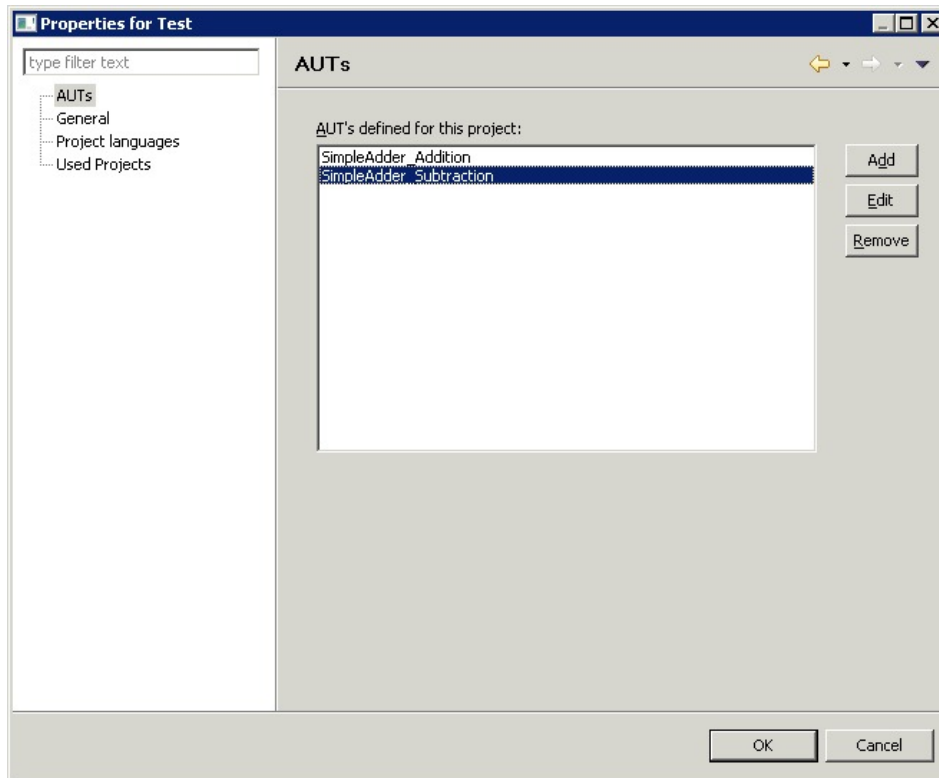


Figure 3.4: AUT Properties Dialog

Defining and editing AUT's from here involves the same steps as defining an AUT in the Project wizard (→ page 45) .

3.5.2.5 Duplicating AUT configurations

You can duplicate an existing AUT configuration from the Project properties dialog:

1. Select "AUTs" from the tree on the left-hand side of the Project properties dialog.
2. Select the AUT configuration you want to duplicate.
3. Click "Duplicate".

4. The AUT configuration dialog will open. The AUT configuration name and the AUT ID are automatically changed so that they remain unique. We recommend writing more meaningful names for the configuration and the ID, however.

3.5.2.6 Editing the AUT configurations in a Project

Select "AUTs" from the tree on the left-hand side of the Project properties dialog.

In the screen that appears, you can see the AUT's you have added to this Project. Select the AUT you want to add a configuration to, and click "Edit".


In the next screen there is a box labelled "AUT configurations". You can choose to add, edit or delete AUT configurations.

Adding and editing AUT configurations from here involves the same steps as adding an AUT configuration in the Project wizard (→ page 48) .

3.5.3 Reusing (referencing) whole Projects in a Project

Jubula lets you reuse (reference) Projects as libraries of Test Cases in other Projects.

To reuse Projects in Jubula:

1. Make sure that the Project you want to reuse is in the database.
2. Select:

and select *Used Projects* from the tree on the left of the dialog that appears (Figure 3.5 → page 37).
3. A list of Projects you can reuse will be offered on the left-hand side of the dialog. You can only reuse Projects which support the same toolkit as your current Project (e.g. Swing, Concrete).



To be able to reuse a Project, you must have checked the *reusable* box in the Project properties for the Project (→ page 33) .

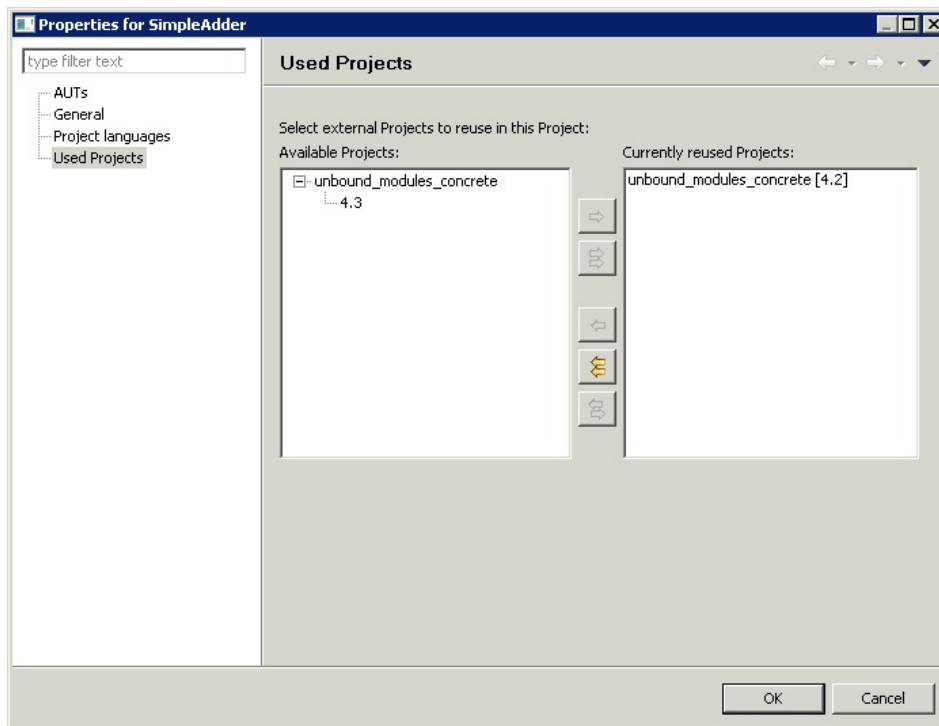


Figure 3.5: Reused Projects

4. From the list of reusable Projects, select a Project and its version to reuse in the current Project. Use the arrows to move it to the list of reused Projects.
5. Click "OK".
6. The Test Cases from the Project you chose to reuse will appear in the Test Case Browser, under a category with the same name as the reused Project. The Test Cases will be colored blue to distinguish them from other Test Cases in this Project.

You cannot edit these Test Cases here – but you can reuse them in your Test Cases for this Project and edit certain details (referenced parameters, component names) when they are reused in other Test Cases.



7. You can change the version of the reused Project via the *Used Projects* Properties dialog, by clicking on the "change used version" button (→ page 38) .

3.5.3.1 Changing the version of a reused Project

You can change the version of the reused Project you are using in your tests. This is useful to update to a new version of the unbound modules, for example.

1. In the *Used Project properties*, select the currently reused Project version from the list on the right.
2. Select the new version of this Project from the list on the left.



In order to be able to see and select the new version of the Project, it must be in your database!

3. Click the *"Switch version"* button, marked with the opposing arrows.
4. The version of the reused Project will be switched.



If changes have taken place, it may be necessary to update your test. You should especially check for any actions that have become deprecated since the last version, and replace them with new actions.

3.5.4 Opening Projects

1. To open a Project from the database, select:
`Test` → `Open`.
2. If you haven't already logged into the database, a dialog will appear to ask you to do so. See the previous section (→ page 26) for details.
3. Choose the Project you want to open from the combo box in the dialog which appears (Figure 3.6 → page 39).
4. If the Project has more than one version, choose which version you want to open. For more information on Project versions, see the section later (→ page 44).
5. The Project is opened in the ITE.

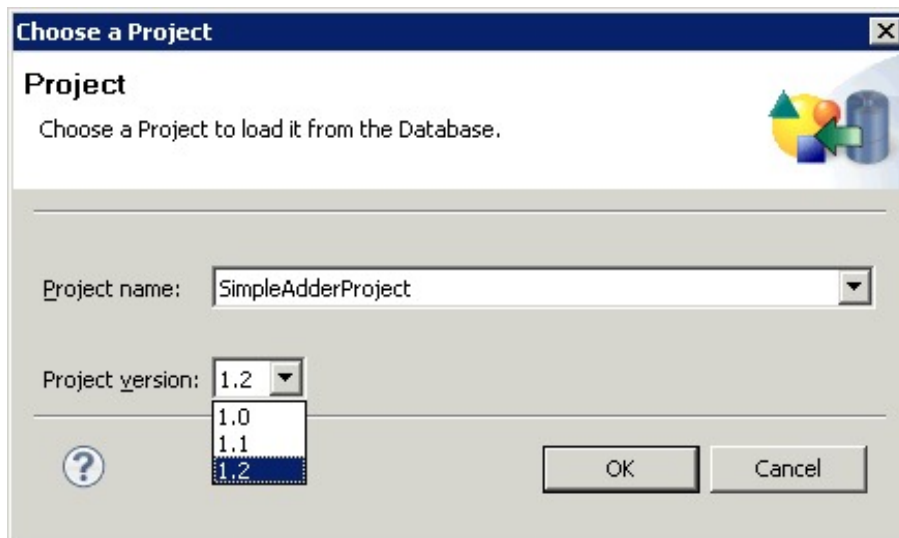


Figure 3.6: Open Project from database

3.5.5 Refreshing Projects

1. To refresh a Project, first select the Project in the Test Suite Browser.
2. Select:
`File` → `Refresh`.

This reloads the data for the Project from the database, ensuring in a multi-user environment that all Project data is current.

If a Project which your current Project reuses has been changed, then you must refresh the current Project for the changes to take effect.



3.5.6 Deleting Projects

1. To delete a Project from the database, select:
`Test` → `Delete`.
2. If you haven't already logged into the database, a dialog will appear to ask you to do so. See the previous section (→ page 26) for details.
3. Choose the Project you want to delete from the combo box in the dialog (Figure 3.7 → page 40).

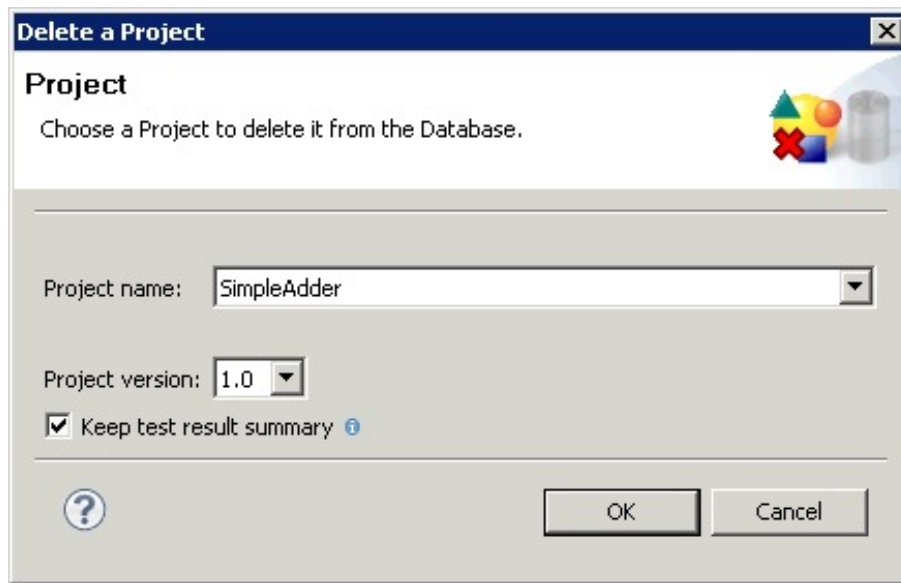


Figure 3.7: Delete Project dialog

4. If there is more than one version of the Project, choose which version you want to delete.
5. Select whether you want to keep the test result summaries associated with this Project or not.



In productive databases where you wish to evaluate test results over longer periods of time, we recommend selecting the option to keep the summaries.

6. Click "OK" to delete the selected Project.
7. Confirm the deletion in the dialog which appears.



Deleting a Project from the database cannot be undone!

3.5.7 Saving a Project as a new Project

1. To save the currently opened Project as a new Project, select:
`Test` → `Save as...`.
2. Enter the new name for the Project in the dialog that appears.

3. This creates a new Project in the database, saved under the name you entered.
4. If no changes have been made since the previous Project save, the new Project has the same content as the previous one.
5. Otherwise, any modifications are saved under the new Project name, and the old Project is kept in the state of its last save.
6. The new Project becomes active in the Jubula ITE.

Any test result summaries for the Project are not duplicated in the new Project. Tests that ran for previous versions of the Project are, however, still in the database to be used for long term analysis.



3.5.8 Importing Projects

1. To import Projects, select:
`Test` → `Import`.
2. In the dialog which appears (Figure 3.8 → page 42), enter or browse to the Projects you want to import.
3. If you have entered the path to a Project, click "*Add*" to add it to the list of Projects to be imported.

Note that the "*Add*" button will only be activated if the path you have entered is correct.



If you add multiple Projects to be imported, you can change the order they are imported in using the arrows next to the list of Projects. You can also remove a Project from the list of Projects to be imported.

If you are importing Projects that are dependent on other Projects (i.e. they reuse other Projects), import the supporting Projects first.



4. If you only want to import one Project, you can select the option to open the Project once it has been imported.

5. When you have finished your selection, click "OK". The Projects you selected will be imported in the order specified.

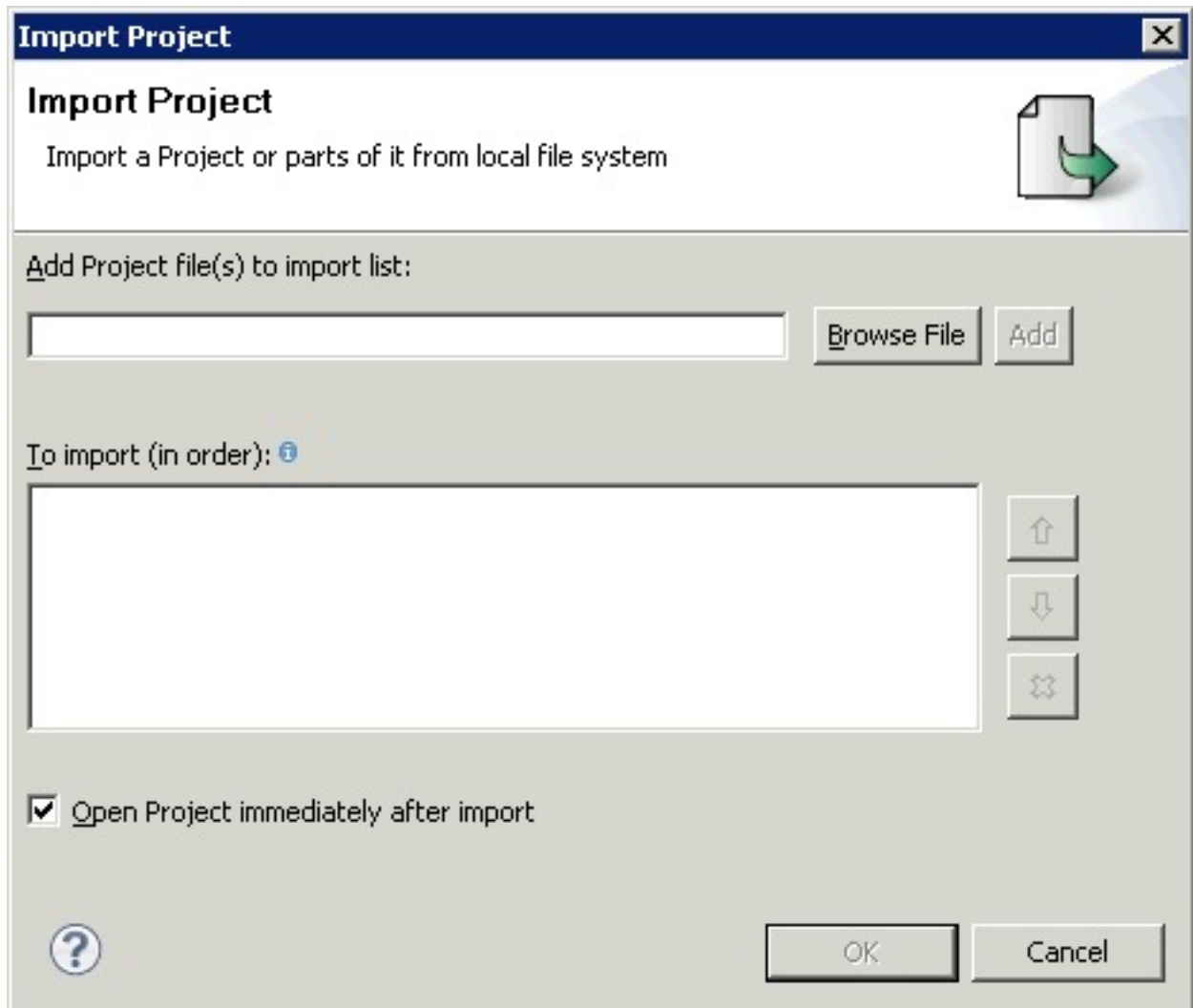


Figure 3.8: Import Dialog

3.5.9 Exporting Projects

You can choose to export the currently opened Project (→ page 43) or all of the Projects in the database (→ page 43) . Projects are exported in .xml format so that they can be added to e.g. version control systems.

Projects are exported along with all their test result summaries.



3.5.9.1 Exporting the currently opened Project

1. To export a single Project from the database, the Project you want to export must be open.
2. Select:
`Test` → `Export`.
3. Choose the location to save your Project, and the name you want to save it under.
4. Click "OK".
5. The Project (including the test result summaries (→ page 142)) will first be refreshed and then exported. It is saved as an .xml file.

When you export a single Project, only the currently opened version is exported.



3.5.9.2 Exporting all of the Projects from the database

1. Select:
`Test` → `Export all`.
2. Choose or create a folder where you want the Projects to be stored.
3. Click "OK".
4. The Projects (including the test result summaries (→ page 142)) will be saved as .xml files.

When you export all Projects, all versions of each Project are exported.



3.5.10 Versioning Projects

1. To create a different version of a Project, select:
`Test` → `Create new version`.
2. An automatic suggestion for the next version number is provided.
3. You can accept this version number or enter a different one.
4. Click "OK" to create the new version.
5. The new version of the Project becomes active in the client.



Any test result summaries for the Project are not duplicated in the new version. Tests that ran for previous versions of the Project are, however, still in the database to be used for long term analysis.

3.6 Defining applications under test (AUT's)

Once you have created a Project, you can define (and edit) AUT's. You can define a new AUT straight after creating the Project in the Project wizard or you can do it later on via the Project properties (→ page 33).

If you will be starting your AUT with the *autrun* command (→ page 52), then you can automatically define your AUT (→ page 53)



The AUT dialog (Figure 3.9 → page 45) appears when you define or edit an AUT.

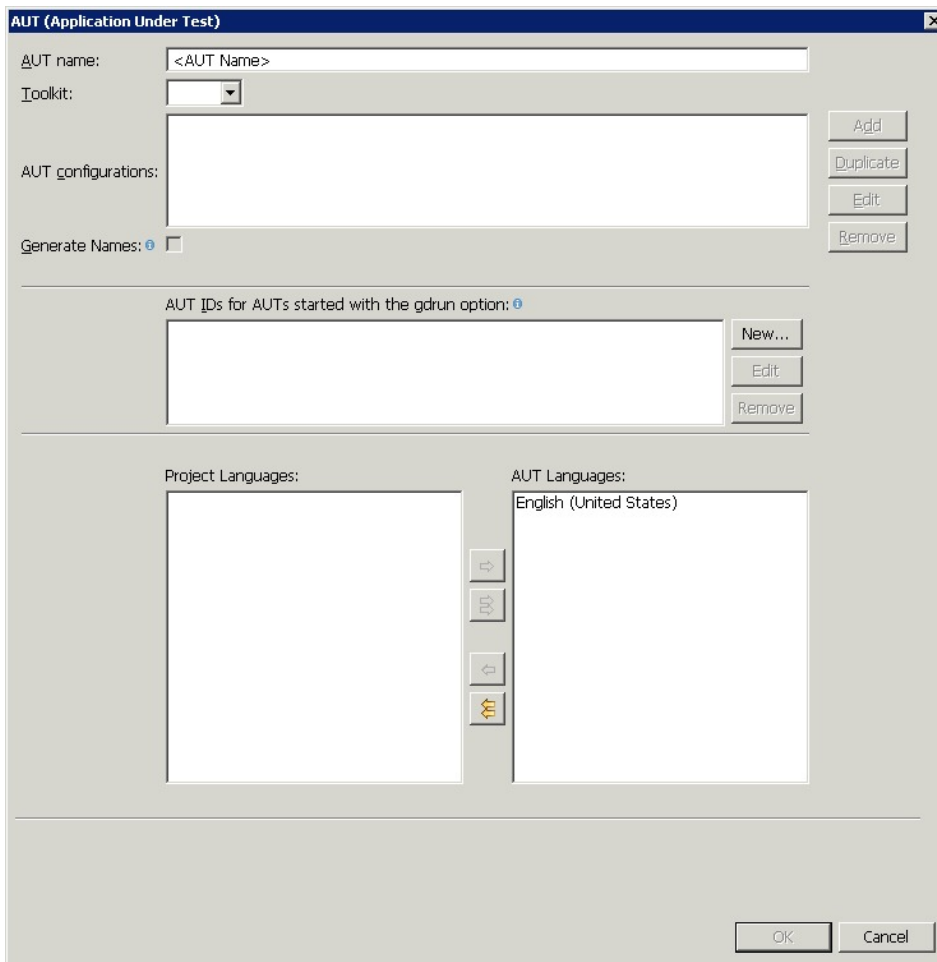


Figure 3.9: AUT Dialog



If you know that you will be working with multiple versions of the same AUT (e.g. a version for Windows and Linux, or two versions that use different databases), then define *one* AUT here and create multiple configurations (→ page 48) for this AUT. This means that your different configurations will all share one object map. If you are working with multiple completely different AUT's, then define the different AUT's here.

1. Enter a meaningful and unique AUT name. This is used to easily identify the AUT later.
2. Select the toolkit the AUT uses from the combo box.
3. If you choose RCP, decide whether or not Jubula should generate names for components in the AUT which have not been named by your developers (→ page 212) . We recommend leaving this option checked, as it increases the robustness of your tests.

4. If you are starting a Java AUT and will be starting it using the *autrun* command (→ page 52) , or if the AUT will be launched from another AUT during the test (→ page 114) , then enter the ID(s) for these AUT's here. **IDs for AUT's started by the *autrun* command**

Enter the AUT IDs you will use for any AUT's started by the *autrun* command (the AUT ID for the AUT is given as a parameter in the *autrun* command (→ page 52) .

IDs for AUT's launched by other AUT's

The AUT ID will take a specific form (→ page 114) and must be defined as such in the AUT definition.



If you will be starting your AUT from Jubula (i.e. via an AUT configuration (→ page 48)) then you do not need to enter any IDs here.

5. From the list of Project languages, select which languages are supported by the AUT.

The languages you select are the languages the AUT can be started in. You will be able to translate the data in your Test Cases into these languages so that a Test Suite will test the AUT in the right language.

If you are editing the AUT and remove an AUT language for which you have already specified data, this will result in the data for that language being lost.



6. If you want to start this AUT via Jubula, you can do so in the Project properties (→ page 48) .

If you do not require an AUT configuration, because you will be starting the AUT using the *autrun* command (→ page 52) , then you do not need to create an AUT configuration.

3.7 Starting and configuring AUT's

3.7.1 Configuring AUT's to be started with Jubula

Once you have created a Project (→ page 30) and defined an AUT (→ page 45) , you can add and edit AUT configurations. The details in the AUT configuration tell Jubula how to start the AUT, e.g. on which machine.

An AUT can have multiple configurations (for example, for local and remote testing). A configuration contains all the information Jubula will require to start the AUT, and may contain platform- or installation-specific information such as paths to working directories, AUT arguments, Java versions, browser choices and activation methods.



If you want to start your Java AUT yourself, and have Jubula connect to it, then use the *autrun* command to start the AUT (→ page 52) . In this case, you do not need to create an AUT configuration.

3.7.1.1 AUT activation

Activation makes sure that the AUT is in focus at the beginning of test execution. This is achieved by clicking somewhere in the AUT window. You can specify the activation method (i.e. where to click) as part of a configuration for an AUT, or you can create a Test Step within a test to do the same thing (→*Reference Manual* p. 96).

The advantage of specifying an activation method here is that it is central and affects each test execution started on this AUT with this configuration.

Bear in mind that you may need to activate your AUT in order for tests to work, especially if the AUT runs on the same machine as Jubula.

3.7.2 Starting Java AUT's (Swing, SWT/RCP/GEF) with Jubula

3.7.2.1 Two options to start Java AUT's

Jubula offers two options to start your Java AUT for testing:

Via an AUT configuration: This option means that you create an AUT configuration in your Project, and the AUT is started by Jubula (→ page 49) .

Using the *autrun* command: This option lets you start an AUT without creating a configuration. Certain start parameters are required for the AUT so that Jubula can locate it (→ page 52) .

3.7.2.2 Configuring a Java AUT to be started by Jubula

The AUT configuration dialog for Java has three different levels of detail: basic, advanced and expert.

See the sections below for information on the different levels.

3.7.2.3 Basic Java AUT configuration

You can use the basic setting (Figure 3.10 → page 50) to configure your AUT if it can be started by an executable file (e.g. .bat, .exe, .cmd, .sh etc.) and if it is written in Java 1.5 or above, and you are using a Java Standard Edition JRE.

If you are testing RCP or GEF AUT's, there are certain specific steps you need to take to configure them. See the sections on RCP testing (→ page 210) , GEF testing (→ page 213) for details.



1. A suggested name for this AUT configuration is generated automatically based on your AUT name and the default AUT Agent host. You can change this name if you wish.
2. The default AUT Agent host is also automatically selected. You can select a new AUT Agent from the combo box or add a new one by clicking "New". For more information on adding and editing an AUT Agent, see the section later (→ page 150) .
3. Enter the executable file name in the *Executable File Name* field. This path can be relative if you define a working directory (→ page 50)).
4. Enter the AUT ID that will be given to this AUT when it is started.

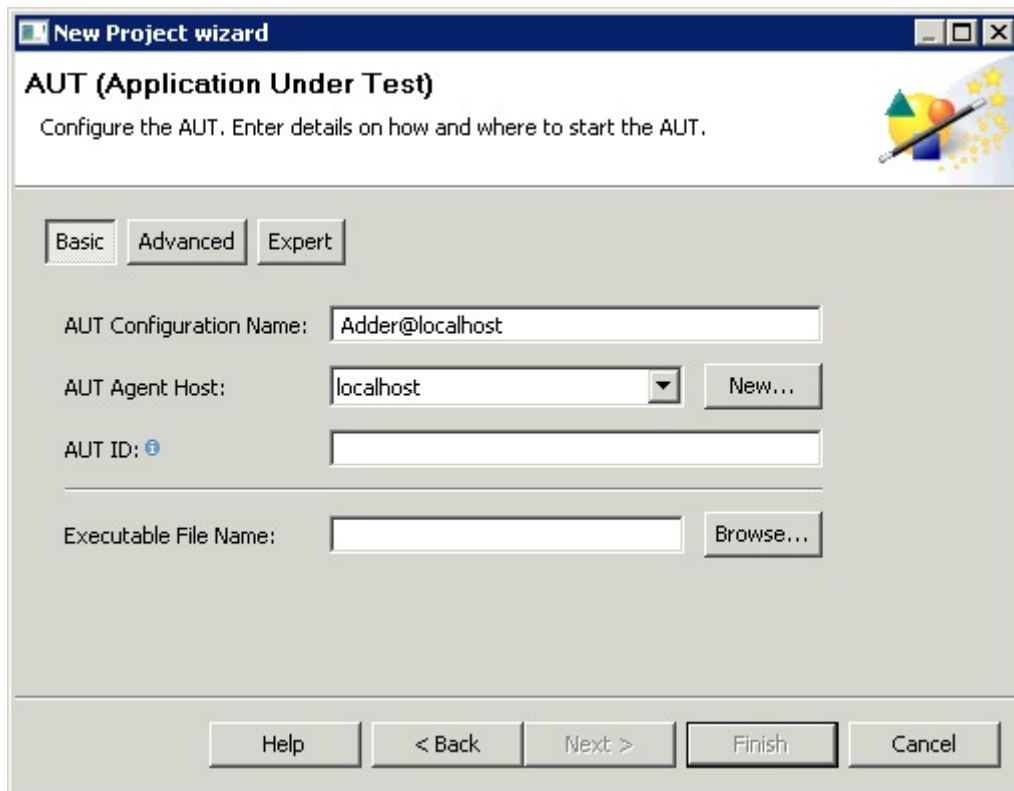


Figure 3.10: AUT configuration window: basic

For information on the advanced properties for the AUT configuration, see the next section (→ page 50) .

3.7.2.4 Advanced AUT configuration

You can use the advanced dialog (Figure 3.11 → page 56) if your AUT is a Java JAR which can be started with a double click, or if your application can be started using the class name and the classpaths to your AUT. The advanced configuration dialog also lets you create a working directory for your AUT, and add command-line arguments needed to start the AUT. You can select a JRE executable and, for SWT/RCP AUT's, a keyboard layout.

1. Enter the JAR path (directory and file name) into the *Executable JAR File Name* field.

This path can be relative (if you define a working directory), or absolute. This JAR file must contain a manifest file which contains the main class and the classpath.

2. You can optionally create a working directory to store files in. The server directory of the Jubula installation is selected as default. To create a working directory elsewhere, browse to and select the location.

The working directory is the root directory for any classpaths, classnames, JAR files and JRE binaries. If you create a working directory, you can enter the paths to these items using a relative path, with the working directory as the root. For more information on relative paths, read the section in the reference manual (→*Reference Manual* p. 377).

3. If your AUT must be started with the class name, add the main class name and the classpaths into their relative fields. The paths can be relative (if you have defined a working directory), or absolute.

Add all the necessary files and directories to start the AUT.



4. Enter any necessary command-line arguments for the AUT in the *AUT Arguments* field.
5. Browse to a JRE executable or add a new one by clicking "New". The Java version used must be 1.4 or later.

Java is installed with Jubula. You can find the Java file in: `Jubula` → `jre` → `bin`. Use `java.exe` if you want to use a console, use `javaw.exe` if you do not want a console.

6. For SWT and RCP AUT's, select which keyboard layout is used on the machine on which the AUT will run. g

The keyboard layout is not the actual keyboard attached to the computer, but is based on the regional language settings for the operating system.



Jubula supports English (US) and German (DE) keyboard layouts out-of-the box. If you want to use a different keyboard layout, see the reference manual for information on creating keyboard layouts (→*Reference Manual* p. 387).

For information on the expert properties for the AUT configuration, see the next section (→ page 52).

3.7.2.5 Expert AUT configuration

You can use the expert dialog (Figure 3.12 → page 57) to configure more detailed information about how the AUT should be started.

1. Add any additional desired *JRE Arguments*.
2. Enter any required *System Environment Variables*, in the format "`<VARNAME>=<value>`", i.e. "`PATH=C:\`". Separate each variable with a new line by pressing »ENTER«.



Please be advised that "embedding" the contents of one variable into another is not supported at this time by Jubula. That is, if you have a variable named `FOO` whose value is "`abc`", and set the value of a second variable `BAR` to "`%FOO%def`", the second variable will *not* contain "`abcdef`", but rather the exact text "`%FOO%def`", without evaluating it.

3. Select an activation method for your AUT. More information on AUT activation is available in the previous section (→ page 48) .

3.7.2.6 Starting Java AUT's with the *autrun* command

The *autrun* command can be used as an alternative to starting your AUT via Jubula (i.e. with an AUT configuration (→ page 48)). It can only be used if your AUT is written in Java 1.5 or above, and you are using a Java Standard Edition JRE.



The *autrun* command cannot be used for HTML or pure SWT AUT's.

The command allows you to start your AUT independently of Jubula, on a machine where the AUT Agent is running. The Jubula ITE, when connected to this AUT Agent will then recognize the running AUT as a testable application.

To use the *autrun* command:

1. Ensure that the AUT Agent is installed on the machine where you will be starting the AUT.

2. Navigate to the *server* directory in the installation via the command line.
3. Start your AUT via the command line by entering `autrun.exe` under Windows or `autrun` under Linux then the following parameters:

Detail	Parameter
-h	-h Gives parameter help
-w, --workingdir	-w <directory> Enter the working directory for the AUT
-a, --autagenthost	-a <hostname> Enter the hostname for the AUT Agent
-p, --autagentport	-p <port number> Enter the port number for the AUT Agent
-swing	If the AUT is a Swing AUT
-rcp	If the AUT is an RCP AUT
-swt	If the AUT is an SWT AUT
-k, --kblayout	-k <en_US> Enter the keyboard layout for SWT/RCP AUT's
-i, --autid	-i <ID> Enter the ID to give to this AUT
-e, --exec	-e <AUT.exe> Enter the executable file for the AUT
-g, --generatenames (optional)	-g <true/false> For RCP AUT's, enter whether to generate technical names. (→ page 45)

Table 3.1: Parameters for *autrun* command

If your AUT is an RCP AUT, use `-data' <WORKSPACE>'` after the executable file to specify the workspace the AUT should use.



3.7.2.7 Creating an AUT definition from a running AUT

Once you have started an AUT using the *autrun* command, you can automatically generate an AUT definition (→ page 45) for this AUT:

- In the Running AUT's View, select the AUT you want to define (it will be marked as an unknown AUT ID).
- Select:
`Create AUT Definition`
from the context menu.
- The AUT definition window will appear. Complete the dialog (→ page 45) and click "OK".

3.7.3 Starting Web AUT's (HTML) with Jubula

The AUT configuration dialog for HTML has three different levels of detail: basic, advanced and expert.

See the sections below for information on the different levels.

3.7.3.1 Basic HTML AUT configuration

Use the basic setting to specify the URL and Browser you wish to start this AUT configuration on.

1. A suggested name for this AUT configuration is generated automatically based on your AUT name and the default AUT Agent host. You can change this name if you wish.
2. The default AUT Agent host is also automatically selected. You can select a new AUT Agent from the combo box or add a new one by clicking "New". For more information on adding and editing an AUT Agent, see the section later (→ page 150) .
3. Enter the AUT ID that will be given to this AUT when it is started.
4. You can optionally create a working directory to store files in. The server directory of the Jubula installation is selected as default. To create a working directory elsewhere, browse to and select the location. For more information on relative paths, read the section in the reference manual (→*Reference Manual* p. 377).
5. Enter the URL of your AUT.



Relative paths to the URL cannot be used!

6. Select the browser you want to start the AUT in.

For information on the advanced properties for the AUT configuration, see the next section (→ page 55) .

3.7.3.2 Advanced HTML AUT configuration

You can use the advanced dialog to enter the browser path for your browser. This lets you use a specific version of the browser (not available for Internet Explorer).

For information on the expert properties for the HTML AUT configuration, see the next section (→ page 55) .

3.7.3.3 Expert HTML AUT configuration

You can use the expert dialog to enter an *ID attribute name* (→ page 220) . If you have used a specific tag to name components in your application, enter the tag in the Expert Configuration area. Jubula will then use this information instead of the *name* attribute in the object recognition.

You can also select an activation method for your AUT. See the section on AUT activation (→ page 48) for more details.

New Project wizard

AUT (Application Under Test)
Configure the AUT. Enter details on how and where to start the AUT.

Basic **Advanced** Expert

AUT Configuration Name:

AUT Agent Host:

AUT ID:

Executable File Name:

Executable JAR File Name:

AUT Base/
Working Directory:

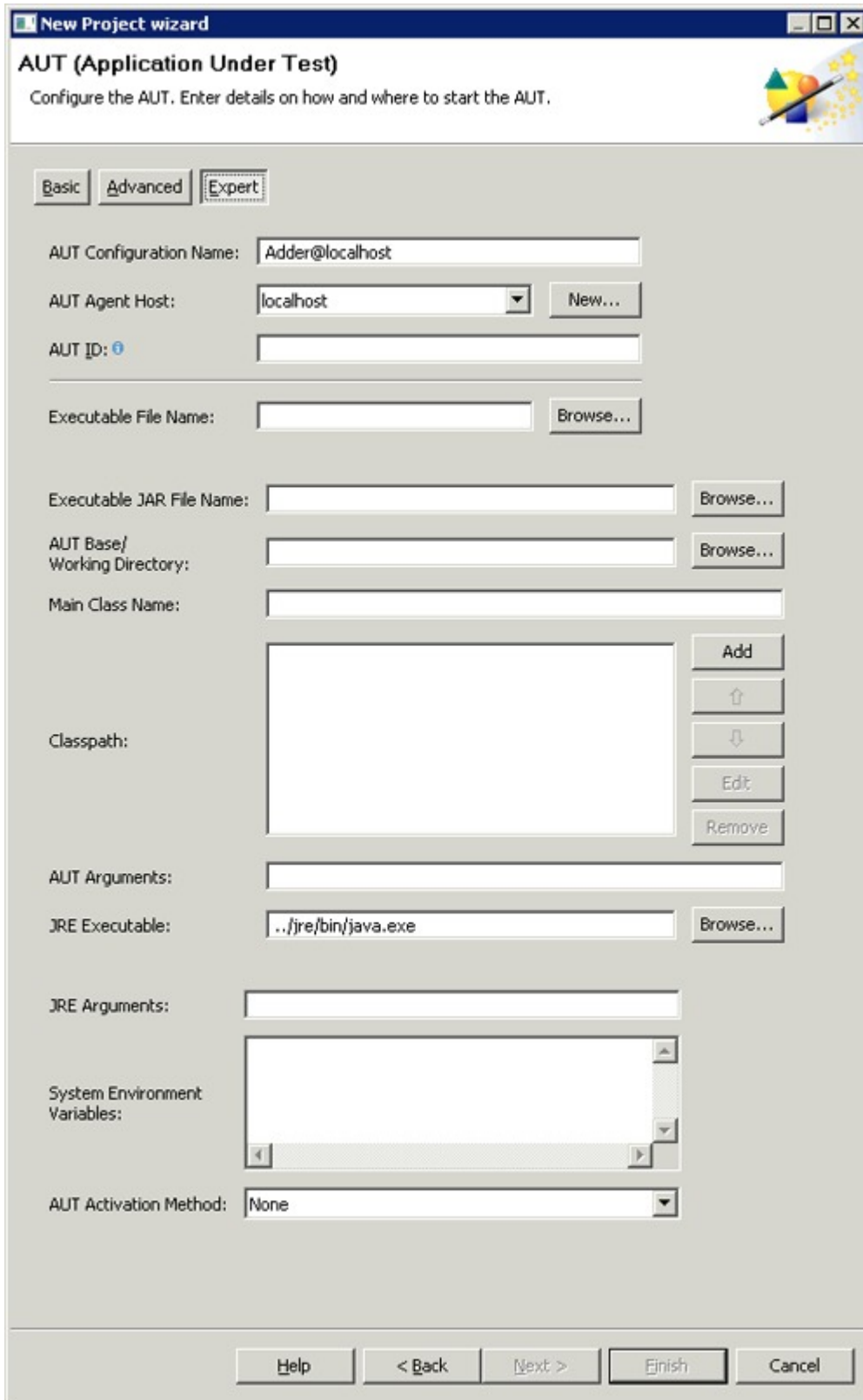
Main Class Name:

Classpath:

AUT Arguments:

JRE Executable:

Figure 3.11: AUT configuration window: advanced



New Project wizard

AUT (Application Under Test)
Configure the AUT. Enter details on how and where to start the AUT.

Basic **Advanced** **Expert**

AUT Configuration Name:

AUT Agent Host:

AUT ID:

Executable File Name:

Executable JAR File Name:

AUT Base/
Working Directory:

Main Class Name:

Classpath:

AUT Arguments:

JRE Executable:

JRE Arguments:

System Environment
Variables:

AUT Activation Method:

Figure 3.12: AUT configuration window: expert

3.8 Working with browsers: renaming, deleting, using IDs

Select an item in a browser to see read-only details in the Properties View, Data Sets View and Component Names View for this item.

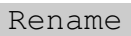
You can rename items from the Test Case Browser, Test Suite Browser or Component Name Browser.

You can open editors for items in certain browsers. See the later section (→ page 60) for more details.

You can also save the ID of Test Cases and Test Suites in browsers to the clipboard, in order to link these elements with external systems.

3.8.1 Renaming items in browsers

To rename an item in the Test Case Browser, Test Suite Browser or Component Name Browser:

1. Select the item you want to rename in the browser.
2. Select:

from the context-sensitive menu.
3. In the dialog which appears, enter a new name and click "OK".



You can also press »F2« to rename an item.

When you rename a Test Case or Test Suite, you change its specification name.

If you have reused this Test Case or Test Suite in other Test Cases, Test Suites or Test Jobs, the name will also be changed in the places where you have reused it but not renamed it.

3.8.2 Deleting items from browsers

1. To delete a Test Case, Test Suite, Test Job or component name, select it in the browser.
2. Select "delete" from the context-sensitive menu.

3. You can also delete items by pressing »DELETE«

If the item you want to delete is referenced somewhere in your test, you must first remove it from the place where it is reused before deleting it.



3.8.3 Working with IDs for Test Cases and Test Suites

Each Test Case and Test Suite in a Project has a unique ID which can be used to refer to this Test Case or Test Suite within the Project and from external systems.

3.8.3.1 Copying the ID of a Test Case or Test Suite to the clipboard

1. Select the Test Case or Test Suite whose ID you want to use from the Test Case Browser or Test Suite Browser.

Make sure you select the original specification of the element, not a place where it has been reused. Press »F6« to find the original specification of a selected element.



2. Select:

Copy ID to clipboard

 from the context-sensitive menu.

3. You can now paste the ID into your external system.

3.8.3.2 Opening an element based on an ID in the clipboard

1. Copy the ID you want to find from your external system.
2. In Jubula, press »SHIFT+F9« to open the Test Case Editor or Test Suite Editor for the element with the ID from the clipboard.

3.9 Working with editors: opening, adding/deleting/renaming items, commenting, extracting and replacing, reverting changes

The editors for Test Cases, Test Suites and Test Jobs all let you work in a similar way in Jubula. When you are working in an editor, the Properties View, Data Sets View and Component Names View on the right hand side of the screen contain information about the selected item.

3.9.1 Opening items in editors

To open an existing Test Case, Test Suite or Test Job in an editor, you can either double-click the item you want to open in its browser or you can select:

`Open With → ... Editor`

from the context-sensitive menu for the item.

You can also open an existing Test Case without having to select it from the Test Case Browser using the *Open Test Case* dialog (→ page 76) .

3.9.2 Adding items to editors

You can add (reference) Test Cases in other Test Cases and in Test Suites. You can also add (reference) Test Suites in Test Jobs.

To add items to an editor:

1. Open the editor by double-clicking the item you want to add things to.
2. You can now:

A Add items via drag and drop from the browser into the editor:

Test Cases :

Test Cases can be added to other Test Cases and Test Suites by selecting the Test Case from the Test Case Browser and dragging it into the editor.

Test Suites :

Test Suites can be referenced in Test Jobs by selecting

the Test Suite to reference from the Test Suite Browser and dragging it into the Test Job Editor.

Once you have added a Test Suite to a Test Job, you must enter the AUT ID for the AUT to be tested in the Properties View.



- B For Test Cases, you can also use the *Reference Existing Test Case* dialog. Using the context-sensitive menu in the editor, select:

Reference Existing Test Case

You can also press »ENTER« in the editor to open the *Reference Existing Test Case* dialog.



Using this dialog Figure 3.13 → page 62 , you can choose a Test Case or Test Cases to add to the editor. You can filter in through the dialog using the field at the top. Use star * as a wild card.

Items you reference in editors are marked with a small arrow to show that they are reused (referenced) here. The name of the item is contained in angled brackets (< >) to show that it is the same name that you used when you specified the Test Case. Items can be renamed to reflect their particular use in this editor (→ page 62) .

You can't add items which would cause an infinite loop.



3.9.3 Deleting items from editors

1. Open the item from which you want to delete other items in the editor
2. Right-click on the item(s) to be deleted (use »CTRL« to select multiple items) and select "*Delete*" from the context-sensitive menu.
3. You can also use »DELETE« to delete items.
4. Save the changes in the editor.

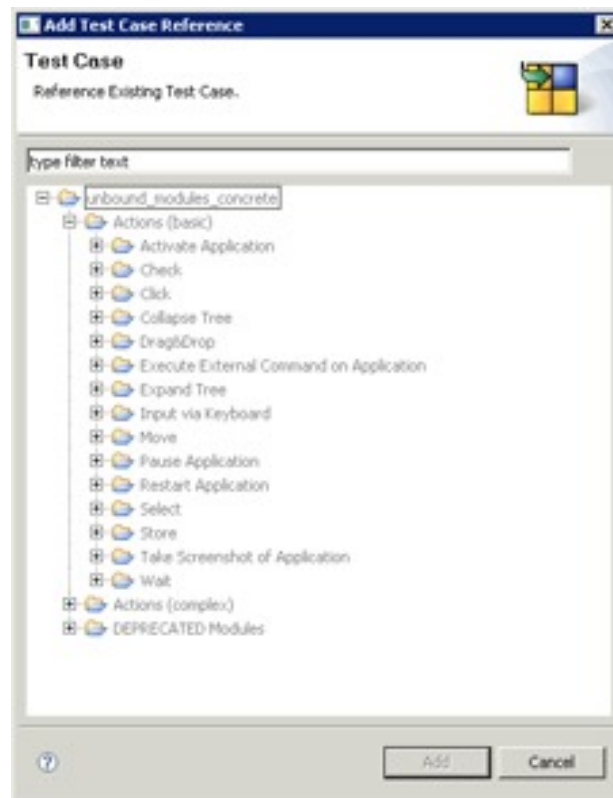


Figure 3.13: Add Test Case reference dialog

3.9.4 Renaming items in editors

You can rename items referenced in Test Cases, Test Suites and Test Jobs.

1. Open the editor containing the items you want to rename.
2. Select the item you want to rename in the editor.
3. In the Properties View, enter the new name for the item in the *Reference Name* field at the top of the Properties View.
4. Save the editor.

The angled brackets from the item you just edited will disappear from the name in the editor. This indicates that the item has been renamed here. You will still see the original item name behind the new name in brackets. In the preferences, you can opt not to see the original name when you have renamed an item (→ page 149) .

The original item in the browser still has its original name. The referenced item you just renamed will keep its new name even if you rename the original item.

If you want to reset the original specification name, empty the *Reference Name* field and save the editor. The item will redisplay the angled brackets and will have the same name as the specification name.

3.9.5 Adding comments to items in editors

You can add comments to items via the editor for an item. You can add comments to the item itself, or to other items used in it.

1. Open the editor for the item you want to add comments to.
2. Enter a comment in the *comment* field in the Properties View.
3. Save the changes
4. Comments can be overwritten when an item is reused.

You can see the comment for a Test Case when you hover over it in the Test Case Browser or the Test Suite Browser or in the search results view.



3.9.6 Commenting out items in editors

In the Test Case Editor and the Test Suite Editor, you can deactivate and reactivate Test Cases and Test Steps.

1. Select the Test Steps or Test Cases you want to deactivate and select:
`Set as active / inactive`
from the context menu.

You can also use »CTRL+7« to toggle the items as active or inactive.



2. The items you selected will be set as inactive. They are shown with green text and the sign `//` before the Test Case or Test Step name.

3. Any inactive items are not considered in Test Suite validation, nor are they executed during a test run.
4. You can reactivate the items by selecting:
`Set as active / inactive`
from the context menu.

3.9.7 Extracting Test Cases from editors: Refactoring

Jubula lets you *extract* Test Cases from other Test Cases and from Test Suites. This lets you create keywords even after you have started specifying.

1. Open the Test Case Editor or Test Suite Editor by double-clicking on the Test Case or Test Suite you want to edit.
2. Select the Test Cases you want to extract by single-clicking them. Use »CTRL« to select more than one item.
3. Right-click in the editor and select:
`Refactor` → `Extract Test Case`.
4. When prompted, enter a name for the new Test Case.
5. The Test Cases you selected will be extracted into this new Test Case.
6. The extracted Test Case appears as a reused Test Case in the current editor. It is marked with a small arrow to show that it is reused, and the Test Case name is in angled brackets (< >) to show that it is the same as the specification name.
7. The Test Case you just created is also visible in the Test Case Browser.



Use this feature when you realize that you are planning on reusing one or more Test Cases for the same or a similar action again. You will save yourself time in test creation and maintenance.

3.9.8 Replacing Test Cases in editors: Refactoring

Jubula lets you *replace* one or more Test Cases in an editor with another Test Case from your library of Test Cases. This is useful if you have created a module to replace one or more Test Cases and you want to be guided through the replacement process.

1. Open the Test Case Editor or Test Suite Editor by double-clicking on the Test Case or Test Suite you want to edit.
2. Select the Test Cases you want to replace by single-clicking them. Use »CTRL« to select more than one item.
3. Right-click in the editor and select:
`Refactor` → `Replace with Test Case`.
4. The first page of a wizard appears in which you can replace selected the Test Cases in a series of steps.

You may only replace single Test Cases which neither have multiple data sets nor central test data / Excel files as data.



Page 1: Replacing the Test Cases

1. On the first page of the wizard Figure 3.14 → page 66 , you can select a new Test Case to replace the selected Test Cases.
2. Browse to and select the Test Case you want to add to the editor.
3. Click "Next" to match the component names for the old and new Test Cases, or click "Finish" to replace the selected Test Cases with the bare new Test Case, without any component names transferred.

Page 2: Matching component names

1. On the second page of the wizard (→ page 67) , you can see an overview of component names for the replacement.

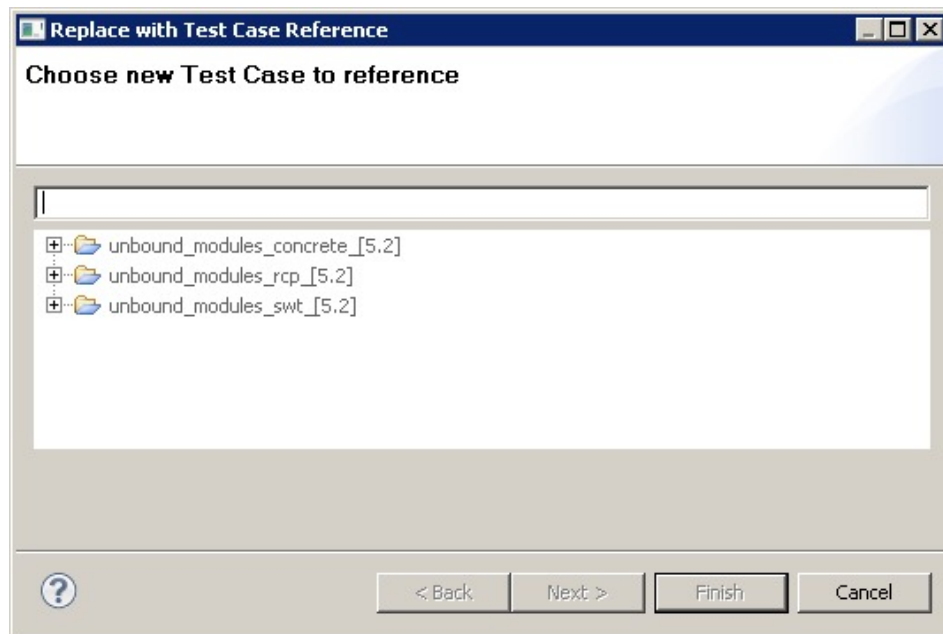


Figure 3.14: Select Test Case



- On the left-hand side you can see any component names that were entered for the Test Cases to be replaced. If the component names were propagated (→ page 104) , you will see a small yellow arrow on the component name icon.

If the old Test Cases contained no component names, then you will see the text *no component names*.

- On the right-hand side, you can see the Component Names View, which shows any component names that are required by the new Test Case.
2. Use this dialog to transfer any component names from the old Test Cases to the new Test Cases. You can enter component names, or you can leave the new component names as they are. You can also set the checkbox in the Component Names View to propagate the name to the next Test Case in the hierarchy.
 3. Once you have transferred the component names, you can click "Next" to add further information or you can click "Finish" to replace the selected Test Cases with the new Test Case and the selected component names.

Page 3: Further information

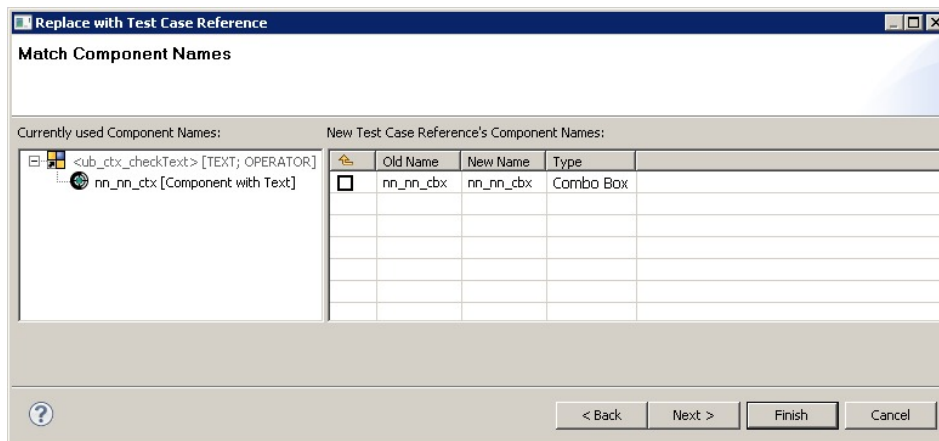


Figure 3.15: Match component names

1. On the final page of the wizard, you can enter a Test Case reference name and a comment for the new Test Case.
2. Once you have entered a name and/or a comment, you can click "Finish" to complete the replacement.

Once you have replaced the Test Case, you must manually adapt the parameters for the new Test Case.

3.9.9 Saving Test Cases from an editor as a new Test Case

Jubula lets you save selected Test Cases from other Test Cases and from Test Suites. This lets you create a new keyword that contains the selected modules without having to manually find them and reuse them.

1. Open the Test Case Editor or Test Suite Editor by double-clicking on the Test Case or Test Suite you want to edit.
2. Select the Test Cases you want to save into the new Test Case by single-clicking them. Use »CTRL« to select more than one item.
3. Right-click in the editor and select:
Refactor → Save as new...
4. When prompted, enter a name for the new Test Case.
5. A new Test Case will be created containing the Test Cases you selected, with the data and component names they used in the current editor.

6. The Test Case you just created is visible in the Test Case Browser.



Use this feature if you have created reusable Test Cases that are correctly modularized, but you require them again (e.g. prerequisites for another Test Case). If the sequence of actions you require is a recurring logical sequence (e.g. enter username, enter password, then click OK for a login dialog), then we strongly recommend using the *Extract* function (→ page 64) instead. This will give you one reusable module, and therefore one central place to make any changes should this sequence change.

3.9.10 Reverting changes in an editor

You can revert an editor you are working in to the state of the last save.

1. In the editor you are in, right-click and select:
`Revert changes`
from the context sensitive menu.
2. Confirm that you want to revert the changes when prompted.
3. The editor will be reverted. Any changes you have made since the last save will be lost.

3.10 Working with categories in the browsers and editors

Categories let you group items together to be able to keep an overview of your data and find things more easily. You can use categories in the Test Case Browser, in the Test Suite Browser, in the Central Test Data Editor and in the Object Mapping Editor (see the section on object mapping for information on categories in the Object Mapping Editor (→ page 124)).

3.10.1 Creating a category

1. In the browser or editor, select:
New → **Category** from the context-sensitive menu.
2. Name the category when prompted.
3. Once you have a category, you can add items to it by drag and drop, and also by double-clicking to add new items (only in the browsers).
4. You can also nest categories in other categories.
5. You can do this either by dragging and dropping a category into another category, or by right-clicking in an already present category and choosing the option to create a new category.

For best practices on how to use categories in the Test Case Browser, see the section on Best Practices (→ page 231)



3.10.2 Creating Test Cases, Test Suites and Test Jobs in an existing category

Creating Test Cases within an already existing category:

1. To create a Test Case in an already existing category, select the category in which you want to create the Test Case with a single-click in the Test Case Browser.
2. Create a Test Case via the context-sensitive menu (→ page 71) .
3. You can also simply double-click on the category in which you want to create the Test Case.

Creating Test Suites and Test Jobs within an already existing category:

1. To create a Test Suite or a Test Job in an already existing category, select the category in which you want to create the Test Suite or Test Job with a single-click in the Test Suite Browser.

2. Create a Test Suite or Test Job via the context-sensitive menu (→ page 110) .
3. To create a Test Suite, you can also simply double-click on the category in which you want to create the Test Suite.

3.11 Working with Test Cases

Test Cases are the “building blocks” (keywords) in Jubula. They can:

- contain other (referenced) Test Cases (→ page 60)
- contain Test Steps (→ page 117)
- be used in Test Suites (→ page 60)
- be used as Event Handlers (→ page 146)

3.11.1 Creating Test Cases

Most of the time spent writing tests will involve creating and editing Test Cases.

To create a Test Case, you must have created or opened a Project (→ page 30) .

The Test Case Browser must also be visible. If it is not visible, change to the Functional Test Specification Perspective perspective and select:

`Window → Show View → Test Case Browser` .

1. Create a Test Case by using the context-sensitive menu in the Test Case Browser:

`New → New Test Case` .

You can also create Test Cases with a double-click on the “Test Cases:” root entry or on a category in the Test Case Browser.



2. A dialog to name the Test Case will appear (Figure 3.16 → page 72). Because Jubula is keyword-driven, it is important to name Test Cases meaningfully – you will be able to read your tests easily and quickly choose which Test Case you need (→ page 228) .
3. Click “OK”.
4. This creates a new Test Case with that name in the Test Case Browser Figure 3.17 → page 72 .

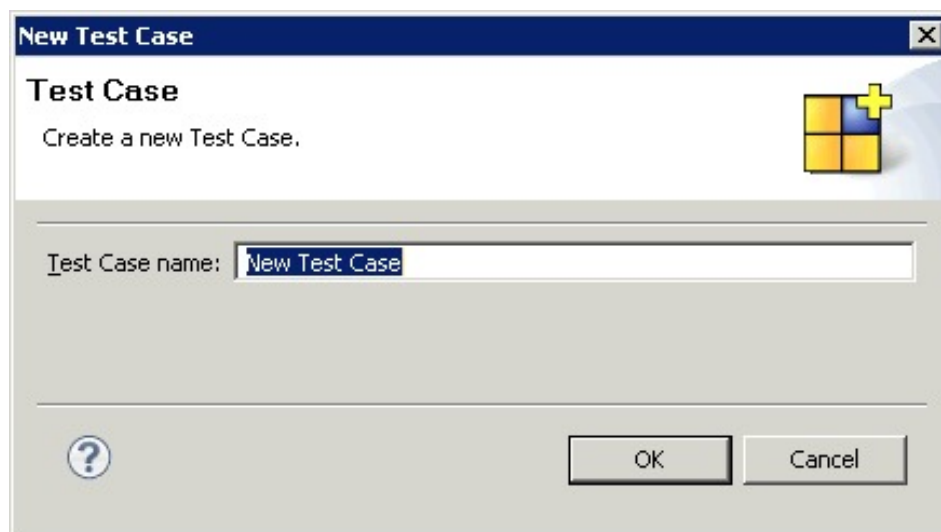


Figure 3.16: New Test Case Dialog

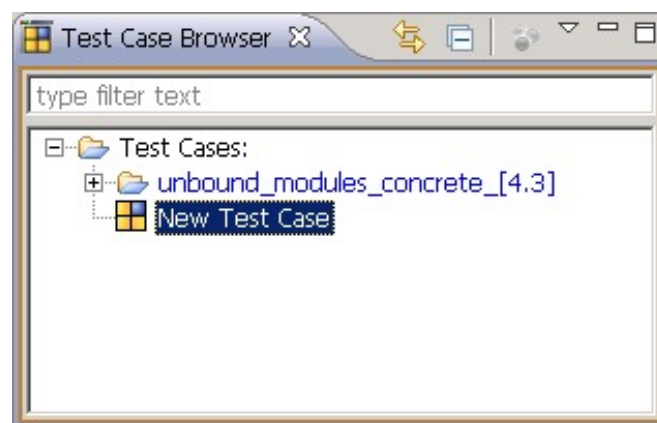


Figure 3.17: Test Case in Test Case Browser



See the Best Practices section for information on how to best structure your Test Cases and tests (→ page 226) .

Next steps

The next step is to add Test Cases from the library of Test Cases to this Test Case (→ page 73) .

You can also:

- Reuse this Test Case to form other Test Cases (→ page 60)
- Edit its content in the Test Case Editor (→ page 60)

- Add this Test Case to a Test Suite to be executed (→ page 60)
- Use this Test Case as an Event Handler (→ page 146) .

3.11.2 Creating tests from the library of pre-defined Test Cases

Everything you need to create tests is already available for you in each Project. The library of Test Cases which appears automatically in each new Project contains all the actions supported by Jubula as well as some examples of more complex keywords. For general information about the library, read the section later on (→ page 74) . To learn how to use the library to create tests, read the next section (→ page 73) .


3.11.2.1 Using the library to create tests

Creating tests with Jubula is really just a matter of:

- Deciding how to structure your tests (i.e. what keywords to create and how to combine them (→ page 223)).
- Choosing the Test Cases from the library (or from your own set of Test Cases) that you need to create these keywords.

To use the Test Cases from the library, you will first need to create a Test Case of your own (→ page 71) .

1. Open the Test Case Editor by double-clicking on the Test Case you want to edit in the Test Case Browser.
2. In the Test Case Browser, browse to a Test Case that you want to add. For help on finding your way around the library, see the later section (→ page 75) .
3. You can add the Test Case by dragging and dropping from the Test Case Browser to the Test Case Editor or you can right click on the Test Case in the Test Case Editor and select:

Reference Existing Test Case →  to see a list of all Test Cases that you can add to this Test Case.

You can filter in this dialog using the field at the top. Use star * as a wild card.





You can open the dialog to reference an existing Test Case by pressing »ENTER« on a Test Case in the Test Case Editor.

4. Once you have added the Test Case, you will need to enter a component name in the Component Names View (→ page 103) and you will need to enter data for the Test Case in the Properties View (→ page 80) .



See the Best Practices section for information on how to best structure your Test Cases and tests (→ page 226) .

3.11.2.2 Information about the library

1. Jubula lets you use a highly reusable library of Test Cases to specify tests.
2. The Projects containing the Test Case libraries are located under:
examples/testcaseLibrary.
3. The Projects available are:
 - unbound_modules_concrete
 - unbound_modules_web
 - unbound_modules_swt
 - unbound_modules_rcp
4. These Projects contain reusable Test Cases which have been created in advance so you do not have to specify them yourself.



Refer to the chapter on Components, Actions, and Parameters ((→*Reference Manual* p. 21)) for information on components, the actions they support, and their parameters.

5. The library is split into categories of *actions* on components. To select something in your AUT, open the *select*

category and then open the category for the type of component you want to select something from.

The names for these Test Cases all begin with "ub". This means that they are *unbound* – they are not in any way dependent on an AUT.



6. Each Test Case in the library contains one Test Step which corresponds to the action in the Test Case name. The component name is a placeholder, and the parameters have been referenced so that you can enter your own data.

The unbound modules Projects which correspond to your chosen Project toolkit are automatically imported into the database and reused in your Project.



We do not recommend making changes to the installed unbound module Projects, for compatibility reasons. If you have Test Cases you want to reuse in other Projects, we advise creating your own library Project.



3.11.2.3 Tips and tricks for using the Test Case library

The Test Cases in the library are organised into actions. In the *basic* category, you will find the various actions offered by Jubula. The *complex* category contains some example keywords which are built up of more than one Test Case.

When specifying your tests, you need to find and choose which actions you will need. This takes some practice, but there are some hints which can help you:

High-Level Actions

Jubula executes *high-level* actions. This means that if you want to select something from a menu using the menupath, for example, you need to look in the category:

Actions (basic)/Select/Menu Bar

and select the Test Case:

ub_mbr_selectEntry_byTextpath

The Test Case finds the menu, opens it, and clicks the item

you specify.

Abstract components

There are some actions which are executable on many different components. Clicks, for example can be executed on practically all components in the interface. You can also check text on labels, combo boxes and text fields. Obviously, it makes sense to specify your Test Cases as abstractly as possible so that they can be reused in more places. This helps keep the maintenance low later.

You will notice in the library that there is no *Button* category under the *Click* category. Instead, you will find various click actions specified for *Graphics Component (grc)*. This is because all components which support clicks belong to the *Graphics Component* group.

In the same way, you will find the category *Check/Component with Text*. You can use the check actions from this category to check text on any component with text – buttons, text fields, combo boxes, labels.

Parameters

Different actions require different data. Some Test Cases in the library have been pre-configured with data to make test specification easier (look in the category *Input via Keyboard-/Application/Key Combination* for a long list).

Some actions let you choose whether you want to enter data using an *indexpath* or a *textpath*. We recommend using the *textpath* so that you are not dependent on the order of e.g. menu entries or tabbed panes.

To make text-based parameters more robust, Jubula often lets you choose an *operator*. You can choose between *equals*, *matches*, *simple match*, *not equals*. Using *matches* lets you use regular expressions so that you don't have to hard-code the whole text parameter.

3.11.3 Opening existing Test Cases

You can open an existing Test Case without having to select it from the Test Case Browser using the *Open Test Case* dialog.

1. This dialog can be opened using the shortcut »CTRL+SHIFT+T« or from the menu:

Edit → Open Test Case

2. In the dialog, you can search for a Test Case or Test Cases in the tree or using the filter (→ page 182) .
3. Select the Test Cases you want to add, and click "OK".
4. The selected Test Cases will be opened in individual Test Case editors.

You can use this option to open a Test Case even if a filter is active in the Test Case Browser.



3.11.4 Editing Test Cases

From the Test Case Editor, you can edit a Test Case. You can:

- add other Test Cases (→ page 60) .
- add and edit Test Steps (→ page 117) .
- rename the Test Case (→ page 58) or the Test Cases it contains (→ page 62) .
- add a comment to the Test Case (→ page 63) .
- add, edit or translate data for the Test Case (→ page 97) .
- extract Test Cases from within this Test Case to make reusable modules (→ page 64) .
- deactivate specific Test Cases in this Test Case so they are ignored during execution (→ page 63) .
- replace one or more Test Cases in it with another Test Case (→ page 65) .

3.11.5 Adding and inserting new Test Cases to a Test Case

1. Open the Test Case Editor by double-clicking on the Test Case you want to edit in the Test Case Browser.

2. From the context-sensitive menu, select:
`Add → New Test Case`
(This places the Test Case after any other items in the Test Case Editor.) or
`Insert → New Test Case`
(This places the Test Case above the currently selected item in the Test Case Editor.)
3. In the dialog that appears, enter a meaningful name for the new Test Case.
4. Click "OK".
5. The Test Case appears in the Test Case Editor in the parent Test Case and also appears in the Test Case Browser. The Test Case is marked with a small arrow to show that it is reused here. The name of the Test Case is contained in angled brackets (< >) to show that it is the same name that you used when you specified the Test Case.
6. Save the changes in the editor.

3.11.6 Moving Test Cases to external Projects

I

If you are reusing a Project or Projects in your currently opened Project (→ page 37) , you can move Test Cases you create to these reused Projects to make them a part of that Project. This lets you add useful Test Cases to external libraries to be reused in other Projects.

1. In the Test Case Browser, select the Test Case, Test Cases or categories you want to move to an external Project that you are currently reusing in the open Project.
2. Right click on the selected items and select:
`Move to external Project`
from the context-sensitive menu.
3. From the dialog which appears, select which reused Project to move them to.
4. The items you selected will appear in the blue colored category of the reused Project you selected and will disappear from their original place.

5. The structure of your Project stays the same – the moved Test Cases are not deleted from places where you have reused them, for example.

If you move Test Cases containing referenced Test Cases from other Projects, make sure that the Project the Test Cases are dependent on is also reused in your external Project.



3.12 Working with test data

3.12.1 Data types and entering data for Test Cases

Most actions supported by Jubula require one or more *parameters* (e.g. how often to click, what text to enter). Once you have added Test Cases or Test Steps to a Test Case, you will see which parameters are required by this action in the Properties View.



You can also enter *data sets* in the Data Sets View (→ page 97) and you can create central data sets to be reused throughout the Project (→ page 90) .

Jubula lets you be very flexible with data, so you have a choice about what to enter in the Properties View:

1. Concrete values (→ page 80) .
2. References (→ page 81) .
3. Variables (→ page 83) .
4. Functions (→ page 85) .
5. A mixture of the above (→ page 88) .
6. An Excel file (→ page 94) .
7. A reference to a central data set (→ page 90) .

3.12.2 Entering concrete values as data in Test Cases

- In the Properties View for a Test Case, you can enter a *concrete value* as a parameter e.g. Hello.
- This means that this particular parameter cannot be changed when you reuse its parent Test Case.
- Enter concrete values for things that you expect to stay the same, like your choice of *operator* (i.e. matches, equals) or the click count. Deciding which data not to parametrize is

an important decision is test creation. See the Best Practices section for more information on this (→ page 226)

Press »CTRL+SPACE« to get content assist in the Properties View for certain parameters.



- Depending on your test structure, you may want to use concrete values for all the parameters in a keyword. If, for example, you have a keyword to open the "New Category" dialog from a menu, you will probably want to write the menupath as a concrete value. After all, this Test Case is designed to open this specific dialog, so the menupath is fixed.

Any concrete values you enter into a Test Case are valid whenever you reuse this Test Case. If you change the concrete values in the original Test Case, all of the places you have reused it will change as well.



3.12.3 Using references for data in Test Cases

- If there are parameters in a Test Case that you want to change when you reuse it, you can enter a reference instead of a concrete value. Deciding which data to parametrize is an important decision in test creation. See the Best Practices section for more information on this (→ page 226) .

Press »CTRL+SPACE« to get content assist in the Properties View for certain parameters.



- To enter a reference for a parameter, in the Properties View, enter a reference name, preceded by an equals sign: =REF_NAME.

Reference names may only consist of letters, numbers and underscores. You cannot use spaces in reference names.



 reference symbol



- Press »ENTER«. A yellow arrow will appear next to the *parameter* field in the Properties View.

It helps to choose names that are meaningful so that you know what sort of data to enter later. Instead of `=TEXT`, you could use `=CATEGORY_NAME`, for example (→ page 230) .

- You will see that the reference becomes a parameter of the parent Test Case. It appears in the Test Case Browser next to the parent Test Case in square brackets.
- You will be able to enter data for this parameter when you reuse the Test Case.



You can enter data for it now if you want to – this is essentially like having default data. They appear when you reuse the Test Case, but you can overwrite them (→ page 100) .

For information on editing and deleting referenced parameters from Test Cases, see the following section (→ page 82)

3.12.4 Using the edit parameters dialog to add, edit and delete references

You can use the edit parameters dialog to add, edit and delete parameters for a Test Case or a central test data set. This section is concerned with using the dialog for editing parameters in a Test Case. For editing parameters for a central test data set, see the section later (→ page 90) .

To edit the parameters for a Test Case:

1. Open the edit parameters dialog by right-clicking on the root node in the Test Case Editor and selecting **Edit parameters** from the context-sensitive menu.
2. In the dialog, you can see any parameters you have referenced for this Test Case, and what type of parameters they are.

3. From the dialog, you can add new parameters to the Test Case, edit existing parameters, and delete parameters. You can also change the order the parameters appear in.
4. Use the *lock* option to lock the parameters for this Test Case. This means that you cannot delete them, rename them or remove them in this Test Case.

In the Properties View, you can only *add* references for parameters. To edit or delete references, you must use this dialog.



3.12.5 Using variables as data for Test Cases

- Jubula lets you store values from the AUT to use later.
- Some actions (e.g. *store value*) let you save a value as a variable. You specify the name of the variable, e.g. `USERNAME`.
- When you want to use the variable, you can enter it as a parameter by preceding it with a dollar sign: `$USERNAME`.

For more information on using variables, see the following sections:

- Working with variables in tests (→ page 83) .
- Working with system variables (→ page 84) .
- Working with the Jubula pre-defined variables (→ page 85) .

3.12.5.1 Reading and using values (variables) from the AUT

You can store values read from the AUT to use as data in other Test Cases.

1. Use one of the *store value* actions on the various components to read a value from a component in the AUT.

You can also use the *store value* action on the *application* component to store a value you enter.





2. In the parameter field, enter a name for this variable (e.g. USERNAME).

Variable names may only contain letters, numbers and underscores

3. When you want to use this value as data for a parameter, enter the variable name preceded by a dollar sign (\$) as the parameter value (e.g. \$USERNAME).

Bear in mind that the variable has to be stored before it can be used as a parameter value.

Read the following sections for more information on:

- Using system variables in tests (→ page 84) .
- Using the pre-defined Jubula variables in tests (→ page 85)

3.12.5.2 Using environment variables in tests

Jubula lets you add variables to your operating system, which can be used in your tests.

You will need to set environment variables which have the form:

`TEST_UDV_<variablename>`

To use the variable in your tests, enter the variable name (everything after the underscore) preceded by a dollar sign. Do not enter the "TEST_UDV_" part.



After entering or changing an environment variable, you will need to restart Jubula. Environment variables are only read from the machine on which the client part of Jubula is running, not from the machine where the AUT Agent is running.

Your system administrator will be able to help you with operating-system specific ways of setting environment variables.

3.12.5.3 Using the Jubula pre-defined test execution variables

1. Jubula contains pre-defined test execution variables which you can use in your tests.
2. The following variables are automatically initialized when executing a Test Suite:

TEST_TESTSUITE: The Test Suite name.

TEST_USERNAME: The account name you are logged into your computer under.

TEST_DBUSERNAME: The database user.

TEST_AUTAGENT: The hostname for the AUT Agent the test is running on.

TEST_PORTNUMBER: The port number for the AUT Agent the test is running on.

TEST_AUT: The AUT name.

TEST_AUTCONFIG: The AUT configuration name.

TEST_CLIENTVERSION: The version of the Jubula client you are using.

TEST_LANGUAGE: The language the AUT and the test are running in, e.g en_US.

3. To use the value of one of these variables in your test, enter:

`${VARIABLE_NAME}`
as the parameter value.

For a list of language codes, see the section in the reference manual ((→*Reference Manual* p. 383)) for details.



3.12.6 Using functions as data for Test Cases

You can let Jubula calculate specific values for you without having to enter the results yourself by using *functions*. There are specific functions that work out-of-the-box with Jubula, and additional functions can be added as well.

3.12.6.1 Syntax for functions

The sign used to introduce a function is the question mark: ? (without quotes).

After the sign, you must enter the name of the function followed by the arguments the function requires, e.g.:

```
?add(arg1, arg2)
```

The arguments are separated by commas and are placed within round brackets.

3.12.6.2 Pre-defined functions

The following functions are available directly in Jubula:

Mathematical functions

The following functions give their results as decimal numbers, e.g. 1.0, 1.2 etc.

add Adds 0 or more numbers to 0, e.g.: `?add(1, 2)`.

sub Subtracts the second number from the first:
`?sub(3, 2)`. This function only accepts two numbers.

mult Multiplies 0 or more numbers by 1 e.g.: `?mult(2, 4)`.

div Divides the first number by the second: `?div(2, 1)`.
This function only accepts two numbers.

trunc Takes two arguments, the decimal to be truncated and the precision (as an integer) to truncate the decimal to. Use 0 to cut off the number to no decimal places (i.e. to receive a plain integer), and use 1 to cut off the decimal to one decimal place etc: `?trunc(2.396, 0)` gives 2 and `?trunc(2.789, 1)` gives 2.7.

round Takes two arguments, the decimal to be rounded and the precision (as an integer) to round to. This function uses *half-up* rounding to round the number so that if the final decimal place after rounding is 5 or higher, the final number will be incremented by 1 e.g.: `?round(2.56, 1)` gives 2.6. If the final number after rounding is 4 or less, there is no incrementation, eg. `?round(2.46, 1)` gives 2.4.

It is currently only possible to use numbers formatted with the decimal mark *period* or *fullstop* (.). Thousands separators may not be used. For example, 1.5 is accepted, but 1,5 is not. 1000 can be entered but 1,000 cannot. Entering 1.000 is equivalent to entering 1.



Use single quotes around negative numbers, e.g. '-0.5'.



Date functions

now Saves the current date in an internal format that can be used as a basis for the `formatDate` and `modifyDate` functions. This function takes no arguments: `?now()`.

formatDate Puts a date into a specific format. The date to be formatted is entered as the first argument, followed by the format string e.g. `?formatDate(?now(), dd-MM-yyyy)`. The formats that can be used here are the formats from the `SimpleDateFormat` class in Java.

parseDate Reads a value that is a date and parses it into an internal format based on the format string given (i.e. how the date should be understood by Jubula). The first argument is the date, and the second is the format string `?parseDate(2011.06.25, yyyy.MM.dd)`. This function should be used when reading and working with dates shown in the AUT.

modifyDate This function can add days (d), months (M), and years (y) to a given date. The date must first be parsed (i.e. using `parseDate`) so that the correct internal format is used. This function takes two arguments: the first is the date to modify, and the second is the modification to perform, e.g. `?modifyDate(?now(), 1d)`. Additions are entered as positive integers (but without a plus sign, e.g. 1d, 1M, 1y) and subtractions are entered as negative integers, e.g. -1d, -1M, -1y.



If you want to use the result of a date function as a part of your test data (i.e. to enter or check), then you will most likely need to use `formatDate` on any date modifications you have performed.

3.12.6.3 Embedding functions in other functions

Functions can be added as arguments to other functions. If, for example, you want to use the result of a subtraction as the first argument of your addition, you could write it like this:

```
?add(?sub(2,1),1)
```

Results in 1.0 + 1, i.e. 1.0

3.12.6.4 Useful examples for functions

Especially when it comes to the date functions, it is often necessary to use multiple functions embedded within each other.

?formatDate(?now(), dd"MMMM"yyyy) e.g. 29 February 2012

?formatDate(?now(), dd"MMM"yyyy) e.g. 29 Feb 2012

?formatDate(?now(), dd.MM.yyyy) e.g. 29.02.2012

?formatDate(?modifyDate(?parseDate(22.2.2012, dd.MM.yyyy),-1d),

This function will parse the date 22.2.2012 into an internal format, subtract one day and then format it as a dd.MM.yy, in this case: 21.2.12.

3.12.6.5 Adding your own functions

You can also add your own functions using an extension point. This is described in the Extension Manual.

3.12.7 Concatenating (combining) parameters

Jubula has various different types of parameter:

- Concrete values (→ page 80) .

- Referenced parameters (→ page 81) .
- Variables (→ page 83) .
- Functions (→ page 85) .

You can use these parameters separately, or you can combine them to create a parameter value. This is useful if a value you want to enter or check consists of parts that change and parts that stay the same.

To combine different types of parameter to make one value, you must write them in a specific way:

1. Referenced parameters must be written with curly brackets around the reference name:

`= { REF_NAME }`

2. Variable names must also be written with curly brackets around them:

`$ { VAR_NAME }`

3. Concrete values are written as normal.

4. For example, you can build a data string that contains all four types of data:

`test_={PROJECTNAME}_${CUSTOMERNUMBER}__?now()`

3.12.8 Viewing and changing data sources for Test Cases

The Properties View shows the currently used *data source* for each Test Case. The following data sources are available:

Referenced Test Case

The data have come directly from the original specification of this Test Case. If the data in the referenced Test Case change, then the changes will also take effect here.

Local Test Case

The data being used were entered for this Test Case. Any changes made in the original specification of the Test Case will not affect the data here.

You can change from *local* to *referenced* test data using the combo box to reset the default value from the referenced Test Case.

Central test data set

The data have come from a central test data set (→ page 90)

Excel data file

An Excel file has been entered as the data source (→ page 94)

3.12.8.1 Changing the data source for a Test Case

- The data source changes automatically to reflect the new data source when you enter an Excel file, a Central Test Data Set or when you enter new data in the Test Case.
- To change the data set back to the original specification data (*referenced Test Case*), first remove all Excel files or Central Test Data Sets, then change the data source in the combo box back to *referenced Test Case*.

3.12.9 Using central data sets

Jubula lets you create and manage central test data sets for a Project which can be reused in Test Cases.

3.12.9.1 Creating and editing central test data sets

To create a central test data set:

1. Open the Central Test Data Editor by clicking the "*Central Test Data Editor*" button on the toolbar or selecting:
Open with → **Central Test Data Editor**
from the Test Suite Browser.
2. In the Central Test Data Editor, select:
Add new Data Set
from the context-sensitive menu or press »INSERT«.
3. In the dialog that appears, enter a name for the new data set and click "OK".
4. The new data set appears in the Central Test Data Editor. You can now add parameters to the data set (→ page 91)
5. You can rename the data set by pressing »F2« or selecting:
Rename
from the context sensitive menu.

6. You can categorize your central data sets using the *Add category* function from the context-sensitive menu (→ page 68) .

3.12.9.2 Deleting central test data sets

You can delete a central test data set if the data set has not yet been reused (referenced) in a Test Case (→ page 92) .

1. Select:
Delete
from the context-sensitive menu or press »DELETE«.
2. A dialog will appear if the data set has been reused and cannot be deleted.
3. You can use the search (→ page 179) to show where the data set has been used.

3.12.9.3 Adding and modifying parameters for central test data sets

Once you have created a central test data set (→ page 90) , you can add parameters to the data set using the *Edit Parameters* dialog.

1. Open the edit parameters dialog for the central data set by double-clicking on the data set in the Central Test Data Editor. You can also select **Edit parameters** from the context-sensitive menu.
2. In the *Edit Parameters* dialog, you can see any parameters you have already added for this data set, and what type of parameters they are.
3. Use the "Add" button to create a new parameter for the data set.
4. Enter a name for the parameter and select the type of parameter it should be (e.g. String, Integer, ...). The type of parameter it should be will depend on which actions are using it. A list of actions and their parameters (and types) is available in the reference manual (→ *Reference Manual* p. 21)).



Names for referenced parameters may only consist of letters, numbers and underscores. You cannot use spaces.

5. You can also change the order the parameters appear in, edit their types and names, and delete them completely using this dialog.

3.12.9.4 Entering data for central test data sets

Once you have created a central test data set (→ page 90) and have added parameters to the central test data set (→ page 91) then you can enter data for these parameters in the Data Sets View.

To enter data sets for a central test data set:

1. Open the Central Test Data Editor by clicking the "*Central Test Data Editor*" on the toolbar or selecting:
`Open with` → `Central Test Data Editor`
from the Test Suite Browser.
2. In the editor, single-click the central test data set you want to add data to.
3. In the Data Sets View, make sure the language in the combo box on the right is the right language for your data.
4. Select "*Add*" to add a row.
5. Enter the values for the parameters in the row.



You cannot add referenced parameters (i.e. reference names preceded by the equals sign) in the Data Sets View for a central test data set.

6. Use the buttons in the Data Sets View to add more rows, delete rows and insert rows above the currently selected row.

3.12.9.5 Reusing central test data sets in Test Cases

You can reuse a central test data set in a Test Case to provide the concrete data for the parameters required by the Test Case.

1. In the Properties View for the Test Case, enter the name of the central test data set you want to use in the *Central Test Data Set* field.

Press »CTRL+SPACE« to see a list of possible data sets for this Test Case. You will only be shown data sets that contain the correct parameters with the correct types.



2. When you have entered a central test data set, then the Properties View shows *central test data set* as the data type. You will see the data from the central test data set in read-only form in the Data Sets View.

If data is missing from the central test data set, you will receive the error that test data is incomplete for any Test Suites this Test Case is used in.



3. You can delete the central test data set used by removing it from the *central test data set* field. The data type reverts to *local data*. For more information on the data sources, see the earlier section (→ page 89) .

You can use central test data sets that contain more parameters than your Test Case. For example, if your Test Case requires the parameters NAME, ADDRESS and your central test data set contains NAME, ADDRESS, POSTCODE, you can still use the central test data set.



3.12.9.6 Importing Excel files as central test data

If you have existing Excel files that you want to convert into central test data, then you can import them via the Central Test Data Editor:

1. In the Central Test Data Editor, right-click and select **Import** from the context-sensitive menu.
2. In the dialog that appears, browse to the directory containing your Excel file(s).

3. Either select the whole directory (if it contains all Excel files) on the left, or select the individual Excel files on the right.
4. Click "*Finish*" to start the import.



Your Excel files must contain the correct amount and type of Project languages.

3.12.10 Using an Excel file as an external data source

In the Properties View, you can add Excel files as a data source to Test Cases which contain parameters referenced from the Test Cases or Test Steps they contain.



The addition of data via Excel is discouraged, because any problems with the data can only be found once the test is running. We recommend using the central data sets in Jubula to manage data within the Project (→ page 90) .

1. Navigate to the Properties View for the Test Case you want to add the Excel file to.
2. Enter the path to the Excel file in the *Excel data file* field.

The path to the Excel file can be absolute or relative (if you have specified a data files path (→ page 149)).



The Excel file must be configured in a specific way in order for Jubula to read it (→ page 95)

3. If you reuse this Test Case, the Excel file you enter will be reused along with the Test Case. When you reuse the Test Case, you can choose whether you leave this file or change it for another one.



If you store your Excel files in your workspace, you will be able to open these directly in Jubula from the navigator view using the in-place editor.

4. A Test Case with an Excel file as data is marked with a small Excel icon in the browsers to help you find it more easily later.

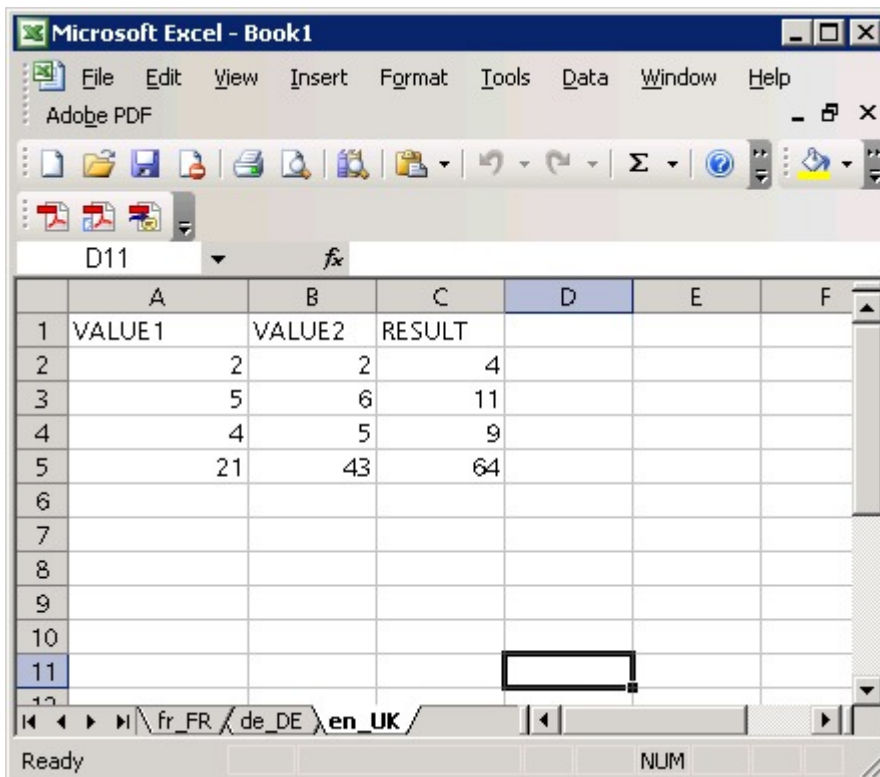
Please note that the Excel file is read at the start of the test execution. Any changes to the file after this will not affect the test data. For information on using the date function in Excel, see the section later (→ page 97) .



3.12.10.1 Configuring the Excel file

- The worksheet in Excel must be named with the language code for the language your data are in. See the Reference Manual ((→Reference Manual p. 383)) for a list of language codes.

For example, if the data of your first sheet are for French, then name the first sheet: `fr_FR` (Figure 3.18 → page 95).



	A	B	C	D	E	F
1	VALUE1	VALUE2	RESULT			
2	2	2	4			
3	5	6	11			
4	4	5	9			
5	21	43	64			
6						
7						
8						
9						
10						
11						
12						

Figure 3.18: Example Excel Table

You can create sheets for every language you need. Make sure that each sheet is named with the language code. Jubula will read the sheet which corresponds to the working language when the test is being executed.

- Name the top cell of each column with a parameter name from your Test Case.

For example, if you entered the reference `"=VALUE1"`, then you must enter `VALUE1` in the top cell of the column which will contain data for that parameter.



Jubula is case sensitive.

- Make sure that all the parameters for your Test Case have a column.
- Do not leave any gaps in the table.
- You must have an entry for each parameter used in the Test Case.
- You can then fill in the values or formulae you want to use for these parameters. Each row in the table represents one set of data for the parameters used.



We recommend that you format cells as text *before* adding the test data. This ensures that Excel's number formatting won't modify the test data in unexpected and undesirable ways. Especially for the boolean values *true* and *false*, make sure you format the column as text.

- If your Excel table contains data which change from day-to-day, then make sure you open the file before starting your test. Otherwise, the data from the last-opened state will be used.



If you store your Excel files in your workspace, you will be able to open these directly in Jubula from the navigator view using the in-place editor.

- See the section later on using the *today* function in Excel to get the current date (→ page 97) .

- Excel files may not contain autofilters in any of the worksheets to be used as data sources for Jubula. Remove any filters from all your Excel sheets before running a test.

3.12.10.2 Using the =TODAY() function in Excel

1. Because Excel stores the "`=today()`" function as a six-digit number, you must use a particular process to use this function to check a date as part of a test.
2. Enter the function `=today()` in a different sheet to the one you are using for your data sets. You can enter it in the same sheet if you want to, but make sure that it has its own column. It must not be in one of the columns you will use as a data set.
3. For example, your `=today()` function is in sheet one, cell G4.
4. You want your date to appear as dd.mm.yyyy.
5. In the column for your data set, enter the following formula:

`=text(Sheet1!G4, "dd.mm.yyyy")`

6. This will mean that the date will be treated as it appears with Excel.

If you are using the `=today()` function, don't forget to open the Excel file before starting your test. Otherwise, the data from the last-opened state will be used.



If you store your Excel files in your workspace, you will be able to open these directly in Jubula from the navigator view using the in-place editor.



3.12.11 Using the Data Sets View to enter data loops and to translate data

The Data Sets View lets you do three things:

- Enter multiple data sets for a parameter from a Test Case (→ page 98) .

- Enter data sets for a central test data set (→ page 92) .
- Translate your parameter values into other languages (→ page 99) .



You can also create central test data sets for your Project to reuse in Test Cases (→ page 90) .

3.12.11.1 Data Sets View: adding multiple data sets to a Test Case

If your Test Case has parameters which have been referenced from the Test Cases and Test Steps it contains, you can enter *data sets* in the Data Sets View.

This means that the Test Case will loop and be executed for each set of data you enter.

To enter data sets for a Test Case:

1. Open the Test Case Editor or Test Suite Editor by double-clicking on the Test Case or Test Suite you want to edit in the browser.
2. In the editor, single-click the Test Case you want to add data to.
3. In the Data Sets View, make sure the language in the combo box on the right is the right language for your data.
4. Select "Add" to add a row.
5. Enter the values for the parameters in the row.



You can also add references in the Data Sets View if you want to specify the concrete values for your data sets when you reuse this Test Case.

6. Use the buttons in the Data Sets View to add more rows, delete rows (if no row is selected, the last row is deleted) and insert rows above the currently selected row.

3.12.11.2 Data Sets View: translating test data

You can add data for other languages supported by your AUT by changing the language in the combo box in the Data Sets View.

You can add supported languages in the Project properties dialog (→ page 33) .



The other combo boxes are there to help you see the Data Sets View in different ways. You can see all the data for one parameter, for one data set or for one language.

3.12.12 Special parameters: empty strings and the escape character

Entering empty strings as parameters

If you want the parameter you enter to be an *empty string* (i.e. nothing), use two single quote marks: "

You can use this with the *equals*, *matches* or *simple match* operators.

You can also use '*^\$*' or '*^\s*\$*' with the operator *matches* to check that a text area is empty.

You can also use '*^\$*' with the operator *matches* to check that a text area is empty.

If a component looks empty, but entering an empty string doesn't work, it may be worth asking a developer what is actually in the component.



The escape character

Some symbols have a special meaning in Jubula. If you want to use the symbol without the special function, you have to *escape* it. The symbol to negate any special function of the following symbol is a backslash: (\).

See the Reference Manual (→*Reference Manual* p. 379)) for more details on special symbols and escaping them.



When you are using regular expressions, you will also need to think about which symbols need neutralising. Sometimes more than one backslash is necessary.

3.12.13 Overwriting data for Test Cases and Test Suites

When you reuse Test Cases, you reuse them with any concrete values they contain, and with any default values that you have entered for their referenced parameters.

Data which has been entered for referenced parameters can be overwritten when you reuse a Test Case.

Reusing Test Cases happens in two ways:

- by adding the Test Case to another Test Case (→ page 60) . The Test Case is then *nested* in the Test Case.
 - by adding the Test Case to a Test Suite (→ page 60) . The Test Case is then *nested* in the Test Suite.
1. Single-click the reused Test Case in the editor for the Test Case or Test Suite where you have reused it.
 2. In the Properties View, you can see the parameters you referenced from in this Test Case. The data source for this Test Case is shown as *referenced Test Case* to denote that the data have not been changed after reusing the Test Case.



A grey diamond next to the *parameter value* field means that the values in it were entered in the original specification of the Test Case.

3. You can enter parameter values here or reference the parameters again. They would then become parameters of the *parent* Test Case.



If you enter values here, you can see a yellow diamond next to the *parameter value* field. This means that the original data have been overwritten. The data source changes to *local Test Case*

If you add references here, you will be able to enter or overwrite data when you reuse the parent Test Case.

Once you change the parameter values of a reused Test Case, any changes to the parameters in the original specification of that Test Case will not affect your new values. You can reset any local changes to the data of the Test Case by removing all Excel files or Central Test Data Sets and then selecting *Referenced Test Case* from the data source combo box.



3.13 Working with component names

Component names are the names you use to refer to the user interface components you want your actions to be executed on. These component names are collected and assigned to real components when you carry out object mapping (→ page 122) .



A component is an element of the user interface you can execute an action on, e.g. a button, a text field.

Each component you are planning on testing requires a component name. We recommend using well-thought out names so that object mapping and test creation / maintenance are easier (→ page 229) .

3.13.1 Creating new component names

For information on creating new component names by *re-assigning* old component names (in the Component Names View for example, see the section on reassigning component names (→ page 103) .

You can create new component names in the Component Name Browser and in the Object Mapping Editor (tree view) via the context menu:

New Component Name



You can also use the toolbar button to create a new component name when you are in the Object Mapping Editor.

A dialog will appear to allow you to enter a component name. We recommend having good naming conventions for component names, as this will help you when mapping later (→ page 229) .

The component name you enter will be created and will appear in the *unassigned component names* category in the Object Mapping Editor and in the *unused component names* category in the Component Name Browser.

Newly created component names have the type *graphics component* until they are made more specific, either by mapping them to a technical name or by entering this new name in the Component Names View for a more concrete component type.



3.13.2 Entering and reassigning component names in the Component Names View

The following section deals with entering component names in the Component Names View when you have added Test Cases from the library. For information on component names in Test Steps, see the Test Steps section (→ page 117) .



1. Once you have added a Test Case from the library of Test Cases (→ page 73) , you will need to enter a component name for the component you want to test.
2. In the Component Names View, you will see the *old name* for this component, which was given in the Test Step contained in the library Test Case. The old name is just a placeholder and should be *reassigned* (Figure 3.19 → page 104).

When you enter a component name in the Component Names View, you create a new reference to a component in the AUT. This component name appears in the Object Mapping Editor to be mapped to a component in your AUT. A Test Case containing component names to be reassigned can therefore be used at a variety of different places to test different actual components. If you want to rename a component name, you can do so in the Component Name Browser (→ page 106) .

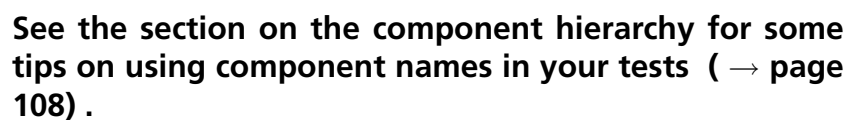


3. In the *new name* field, enter your name for this component, e.g. `MainWindow_NavigationTree_tre`.

We recommend using naming conventions for component names (→ page 229) .



6. Every time you want to carry out an action on this component, use this name.



Select the checkbox on the left of the Component Names View to be able to see and edit (reassign) this component

name when you reuse the parent of the Test Case you are editing.

This is useful for creating flexible keywords that can be used for the same workflow on different components. Depending on what you want the keyword to do when you reuse it, you can reassign the name to test different components in the AUT.

3.13.2.2 Changing component names

If you newly reassign component names in your test hierarchy that were themselves later reassigned, you will be shown a message in the Component Names View.

- The message only appears at places where you have reused the Test Case containing that component, and where you have overwritten the component name.
- For example:
 1. Your component was originally called *LoginDialog_nn_btn*.
 2. You reused the Test Case containing this component, and renamed the component *LoginDialog_OK_btn*.
 3. You then change the component name in the original Test Case to "*LoginDialog_AnyButton_btn*".
- In the Component Names View for the reused Test Case, you will see a warning message that the component name has no type.
- You will also see the new name for the component.

The warning field is not editable.

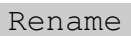


- If you want to keep the same overwritten name, enter the name into the "*new name*" field in the row for the new component name.
- You can also change the overwritten name, or not overwrite it at all. In this case you will need to adapt your object mapping, if this component has been used in a Test Suite.
- Once you have saved the Component Names View, the warning message will disappear.

3.13.3 Renaming component names

Use the rename function to change the name of a component name in your test specification. Whereas *reassigning* component names results in a new component name being created (→ page 103) , renaming just changes the name at all places where it is used.

You can rename component names in the Component Name Browser and in the Object Mapping Editor.

1. Select the component name you want to rename.
2. In the context-menu, select:

3. In the dialog that appears, enter a new component name to replace the old name. We recommend using naming conventions for component names (→ page 229) .
4. Press "OK". The component name you entered will replace the old name throughout your test.



You cannot rename a component name with a name that already exists. To merge two or more component names to the same name, see the section on merging (→ page 106) .

3.13.4 Merging component names

You can merge two or more component names to one unifying name in the Component Name Browser. This makes sense if you have accidentally created two component names for the same component.



The component types must be the same to be able to merge the component names, and the component names must not have already been mapped to different technical names from the AUT. The component names must all come from the current Project, not from any of the reused Projects.

To merge component names:

1. Select the component names you want to merge. Use »CTRL« to select multiple names.
2. From the context-menu, select:
Merge Component Names
3. A dialog will appear in which you can choose which of the selected names you want to merge all the names to. Select the name you want and click "OK".
4. The component names you selected will be merged to the name you specified throughout your test.

3.13.5 Deleting unused component names

When you reassign component names in the Component Names View, you automatically create new component names if the name you enter did not previously exist in the Project.

Larger Projects can often end up with a number of component names that are no longer used. You can see and delete these names in the Component Name Browser. The names in the *Unused Component Names* category (Figure 3.20 → page 107) are not used anywhere in this Project – they are not present in any Test Cases and they have not been mapped to any technical names from the AUT in the Object Mapping Editor.

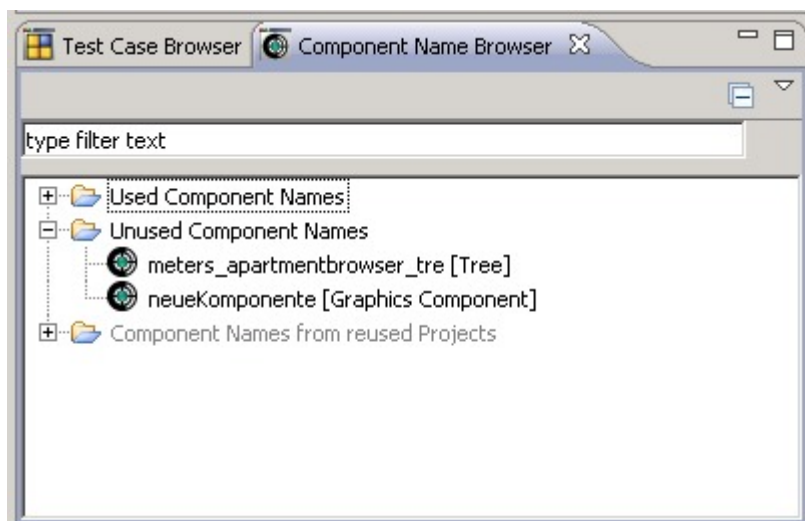


Figure 3.20: The Component Name Browser

To delete unused component names, select the name you want to delete and select:

Delete

from the context-menu.



To cleanup component names that are only used in mappings, or that are no longer used for a specific AUT, use the function *Cleanup unused component names* in the Object Mapping Editor (→ page 129) .

3.13.6 Understanding the component hierarchy

The component hierarchy in Jubula is designed to allow flexible test specification and robust tests. For a graphical overview of the component hierarchy, see the reference manual (→*Reference Manual* p. 375).

Abstract components

Jubula lets you write tests very abstractly at the beginning, only adding detail later. You will notice that the library contains categories such as *Component with Text Input* and *Graphics Component*.

These are *abstract components* – actions in these categories can be used on a wide range of actual components in the AUT. You can use a *Click* action on the *Graphics Component* to click any component in the AUT. You just need to enter different component names for it in the Component Names View. **Using the same component name for different**

component types

What happens if you want to specify a test that clicks in a table and then selects a cell in the table?

The click action is on the *Graphics Component* and the select cell action is on the *Table* component – but you don't want to have two different component names.

This isn't a problem for Jubula. You can use the same component name for different components as long as these are compatible. So, in this case, the *Graphics Component* and the *Table* component can both use the component name e.g. *TableView_MainTable_tbl*.

You cannot use the same component name for incompatible types, for example, trees and tables.



An overview of the component hierarchy can be seen in:
(→*Reference Manual* p. 375).

3.14 Working with Test Suites

3.14.1 Creating a Test Suite

Test Suites are used to organize and prepare Test Cases to be executed.

To create a Test Suite, the Test Suite Browser must be visible. If it is not visible, change to the Functional Test Specification Perspective and select:

Window → Show View → Test Suite Browser.

1. Select:

New → New Test Suite

from the context-sensitive menu in the Test Suite Browser.

2. Name the new Test Suite when prompted with a meaningful name. It is a good idea to have naming conventions (→ page 229) for Test Suites.

3. Click "OK".

4. The Test Suite you created will appear in the Test Suite Browser under the category *Test Suites*.

5. Once you have created a Test Suite, you can:

- Add Test Cases to it to be executed (→ page 60) .
- Configure the Test Suite in the Properties View (→ page 111) .
- Rename the Test Suite (→ page 58) .
- Delete Test Cases from it (→ page 61) .
- Add the Test Suite to a Test Job to be executed along with other Test Suites (→ page 113) .



You can filter in the Test Suite Browser using the field at the top. Use star * as a wild card.



You can combine multiple Test Suites to create a Test Job (→ page 113) . This is useful for testing multiple AUT's (or instances of the same AUT in one test.

3.14.2 Configuring Test Suites in the Properties View

To configure a Test Suite, you must first create one (→ page 110) .

If you are editing the Test Suite, the working language (which is specified via the globe button on the toolbar) must be set to a language supported by the chosen AUT for the Test Suite. Otherwise the Test Suite will be uneditable.



1. Open the Test Suite Editor by double-clicking on the Test Suite you want to configure.
2. In the Properties View, you can:
 - A Change the Test Suite name by entering a new name in the *Test Suite name* field.
 - B Add a comment to the Test Suite (→ page 63) .
 - C Enter a value in the *step delay* field.

The step delay is the time Jubula leaves between each Test Step during test execution. The default is 0 milliseconds (→ page 140) .



- D Select the AUT for this Test Suite. To be able to select an AUT (and object map, and execute your test) you must have added at least one AUT to the Project (→ page 45) .

You don't have to choose an AUT for a Test Suite as soon as you have created it, but you will have to choose one before object mapping, for example.



- E Choose a default reentry type for each of the four error types in Jubula from the combo-boxes.
Event Handlers are Test Cases used to deal with errors during test execution. When an error occurs, the current Test Case is searched for an Event Handler for that error type. If none is found, the parent Test Case is searched, and so on. If no Event Handler for the test is found,

then a default Event Handler (specified in the Test Suite properties) is activated.

As a general rule, you should avoid default Event Handlers being executed. See the sections on Event Handlers for information on the event types (→ page 147), reentry types (→ page 148) and creating your own Event Handlers.

F Save the changes in the Test Suite Editor.

3.15 Working with Test Jobs to test multiple AUT's

3.15.1 Combining Test Suites into a Test Job

You can create a Test Job when you want to execute a collection of Test Suites after each other.

The Test Suites can test different instances of the same AUT, or different AUT's entirely.

You do not have to create a Test Job if you only want to run tests on one instance of one AUT. To do this, you can use a Test Suite (→ page 110) . A Test Job is most useful when you want to switch between two AUT's (either the same AUT or a different one) in one test run.

3.15.2 Testing different AUT's in one test run

You can use Jubula to test multiple AUT's in one test run.

The AUT's can be the same actual AUT which has been started multiple times (to test refresh aspects, for example).

You can test AUT's that were started independently, or AUT's that are launched by other AUT's.

3.15.2.1 Testing independently started AUT's

To be able to test multiple AUT's that are *not* started by each other, the following criteria must be met:

- The AUT's are either written with the same toolkit (e.g. Swing) or,
- you have specified your Project at the *concrete* level, and will only be testing areas of the AUT's that can be tested with the actions that are valid for all AUT types (i.e. no RCP-specific components are involved in the test) (→ page 30) .
- The AUT's are all defined in the same Project.
- The first AUT can either be started using the *autrun* command (→ page 52) or via an AUT configuration (→ page 48) . Any other AUT's required for the Test Job must have been started with the *autrun* command.



To run Test Jobs from the test executor, all AUT's for the test run must already be started when the test execution begins. For unattended build and test processes, this will mean that the AUT's must be started with the *autrun* command.

3.15.2.2 Testing AUT's that are launched by other AUT's

If your AUT starts other AUT's which you also want to test, then the following criteria must be met:

- The AUT Agent must be running in *lenient* or *non-strict* mode (→ page 22) .
- The AUT's must be written with the same toolkit (e.g. Swing) (→ page 30) .
- The AUT's have all been defined for this Project (→ page 45) .
- The order in which the launched AUT's will appear and be tested must be known.



When an AUT is launched by another AUT, the AUT ID for the new AUT is formed as $AUT\ ID + 1$. The next AUT to be started receives the ID $AUT\ ID + 2$, and so on. You can enter these AUT IDs in the Test Suites in the Test Job.

Behavior of AUT's when being started by other AUT's

RCP starting RCP:[The newly started RCP AUT receives the ID $ID+1$.]

Swing Jar or main class starting Swing Jar or main class:[This is currently not possible.]

Swing Jar or main class starting Swing executable:[This is currently not possible.]

Swing executable starting Swing Jar or main class:[The newly started Swing AUT receives the ID $ID+1$.]

Swing executable starting Swing executable:[The newly started Swing AUT receives the ID ID+1.]

If the AUT Agent is not running in lenient mode, then the newly started AUT will shut down.



3.15.3 Creating a new Test Job

To create a Test Job, the Test Suite Browser must be visible. If it is not visible, change to the Functional Test Specification Perspective and select:

`Window` → `Show View` → `Test Suite Browser`.

1. Select:
`New` → `New Test Job`
from the context-sensitive menu in the Test Suite Browser.
2. Name the new Test Job when prompted with a meaningful name. It is a good idea to have naming conventions for Test Jobs.
3. Click "OK".
4. The Test Job you created will appear in the Test Suite Browser under the category *Test Jobs*.
5. Once you have created a Test Job, you can:
 - Add Test Suites to it (→ page 60) .
 - Specify which AUT the Test Suites should test (→ page 115) .
 - Rename the Test Job (→ page 58) .
 - Delete Test Suites from it (→ page 61) .

3.15.4 Specifying which AUT to test in a Test Job

Once you have added a Test Suite to a Test Job (→ page 60) , you must select the AUT ID that you wish to run this Test Suite on from the Properties View.

This enables you to differentiate between multiple AUT's (or instances of the same AUT in your tests).



You must have defined all AUT's to be tested during a Test Job in the AUT definition dialog (→ page 45) .

3.16 Information on Test Steps

Test Steps are the smallest unit in Jubula. They correspond to one action on one component.

Instead of creating Test Steps, we recommend using the library of Test Cases (→ page 73) . Each Test Case in the library contains one Test Step, with all the parameters already referenced. Using the library makes test creation much easier!



3.16.1 Specifying Test Steps

You must have created a Test Case (→ page 71) in order to create Test Steps.



1. Open the Test Case Editor by double-clicking on the Test Case you want to edit in the Test Case Browser.
2. Create a Test Step from the context-sensitive menu in the Test Case Editor:
Add → New Test Step.
(This adds the Test Step after any other items in the Test Case Editor.)
3. Alternatively, use the option in the context-sensitive menu to insert a Test Step. This inserts the new Test Step above the currently selected item.
4. The *New Test Step* (Figure 3.21 → page 118) dialog will appear.

Filling in the Test Step dialog

Test Step name

1. In the *Test Step Name* field, enter a name for the Test Step. This name is just for recognition purposes, but should be easily identifiable at a later point.

Component type

2. Select the component you want to test from the combo box.

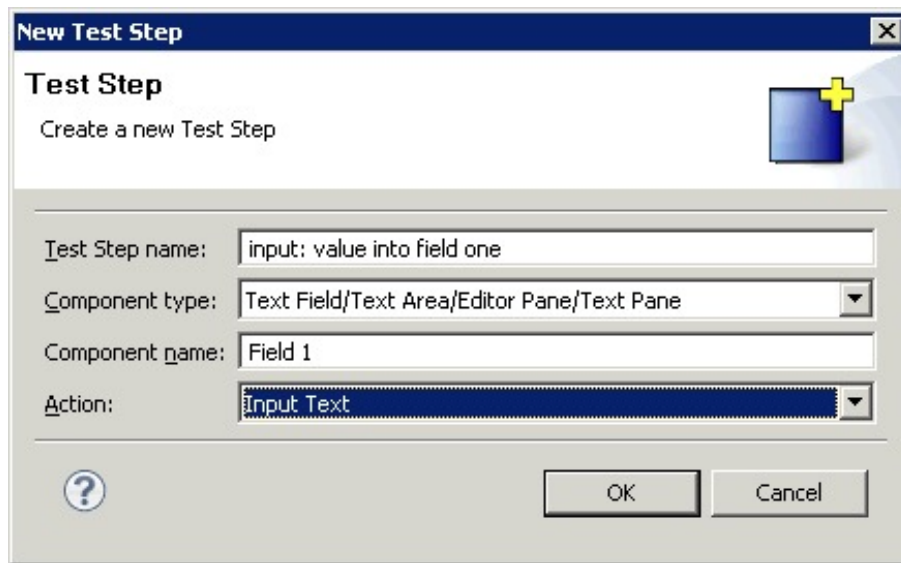


Figure 3.21: New Test Step Dialog



A component is an element of the user interface you can execute an action on, e.g. a button, a text field.

Component name

3. Enter a component name. This is your user-defined name that you will use to refer to this particular component. It will later be mapped to the real component in the AUT. Again, we suggest using conventions to name components (→ page 229) .

4. When you are entering component names, a combo box appears with a list of names you have already used.

If any of the names are grayed out, this means they have been used for other component types and cannot be used for the currently selected component type.

You can change, or *reassign* component names later when you reuse this Test Case. Doing this means that you can execute the same actions on different components in the interface. The section on the Component Names View (→ page 103) contains information on reassigning component names.

Action

5. From the combo box, choose the action you want to execute on this component.

Refer to the chapter on Components, Actions, and Parameters ((→*Reference Manual* p. 21)) for information on components, the actions they support, and their parameters.



6. Click "OK".
7. The Test Step will appear as nested in the Test Case.

You can change the position of Test Steps using drag-and-drop.



You can see and edit the details of any selected Test Step in the Properties View.

The next step is to add parameter values (data) for Test Steps in the Properties View (→ page 80) .

3.16.2 Editing Test Steps

1. Open the Test Case Editor by double-clicking the Test Case you want to edit in the Test Case Browser.
2. Single-click the Test Step you want to edit.
3. In the Properties View, you can:
 - edit the Test Step name
 - add a comment
 - edit the component, component name and action for the Test Step
 - edit the parameter values for the Test Step (→ page 80)
4. You can also move the Test Step within the Test Case using drag-and-drop.
5. To delete the Test Step, select:
Delete
from the context-sensitive menu.



Select the component or action in the Properties View and press »F1« to see context-sensitive help about this component or action from the reference manual.

3.17 Working with manual Test Cases

3.17.1 Creating manual tests

Jubula also lets you create manual tests to be executed and analyzed.

A Test Case containing manual Test Steps is the same as an automated Test Case for all intents and purposes. You can add data, reuse it in other Test Cases, and add it to Test Suites to be executed. The only difference is during execution: manual tests are executed in an interactive mode (→ page 121) .

To create a manual test:

1. Create a Test Case (→ page 71)
2. Add the module for *manual Test Case* from the library of Test Cases (category: Manual Test Step).
3. In the Properties View, you can enter three pieces of data:

The action(s) to perform: Enter a short description which should appear when the test is being executed. You can also enter referenced parameters (→ page 81) or variables (→ page 83) here to specify different data for the test.

The expected behaviour: Enter the description (any data) of the expected outcome of the Test Step.

Timeout: Enter how long the test execution should wait before receiving the information that the Test Step passed or failed. We recommend setting this timeout value high enough for a manual tester to be able to complete the Test Step and enter any comments.

4. Once your manual Test Case is finished, then you can add it to a Test Suite (→ page 60) to be executed (→ page 121) .

We recommend keeping manual Test Cases in a separate category to automated Test Cases. Although it is possible to combine manual and automated Test Cases in one test, make sure you know which Test Suites can be run completely automatically for your unattended tests!



3.17.2 Executing and analyzing manual tests

Once you have created a manual Test Case (→ page 120), you can execute it.

1. Start the AUT (→ page 136) .
2. Start the Test Suite containing the manual Test Steps (→ page 137) .
3. For each manual Test Step in the test, a dialog will appear with the descriptions and data you entered for the *action to perform* and the *expected behavior*.
4. Execute the Test Step as described, enter a comment if necessary, then click either "*passed*" or "*failed*" in the dialog to move to the next Test Step.

If the Test Step fails, then an automatic screenshot will be taken (→ page 153) , and any Event Handlers Chapter "Dealing with errors in tests: Event Handlers" (→ page 146) for this Test Case will be activated.



5. Once the dialog has been closed, the next Test Step dialog will appear.
6. At the end of the test, you can see the test result report like any other automated test (→ page 142) .

3.18 Object mapping

3.18.1 Object mapping

Object mapping is the joining of the component names you have specified in Test Steps to the real components in the AUT. Each component name you have specified in your Test Cases and used in a Test Suite will be shown in the Object Mapping Editor to be mapped.



New component names you have created in the Object Mapping Editor but have not yet used in Test Cases will also be shown in the Object Mapping Editor.

To be able to carry out object mapping, you must have:

- started and connected to the AUT Agent (→ page 21)
- created a Test Case containing Test Steps (→ page 71)
- added Test Cases to a Test Suite (→ page 110)
- specified an AUT (→ page 45)
- configured an AUT if you want to start the AUT via Jubula (→ page 48)
- chosen an AUT for the Test Suite (→ page 111)
- checked that your current working language is supported by the chosen AUT.
- connected to the AUT Agent (→ page 23)
- started the AUT to be mapped (→ page 136)
- opened the Object Mapping Editor (→ page 123)

3.18.2 Working with the Object Mapping Editor

3.18.2.1 Opening the Object Mapping Editor

1. Open the Object Mapping Editor by selecting the Test Suite whose components you want to map and selecting:
`Open with` → `Jubula Object Mapping Editor`.

The Object Mapping Editor also opens automatically when you start the Object Mapping Mode via the toolbar.



The Object Mapping Editor for the AUT used by this Test Suite will appear (see Figure 3.23 → page 135).

You can see the name of the AUT you are mapping in the tab of the editor



3.18.2.2 The different views in the Object Mapping Editor

You have three options to view the Object Mapping Editor - a split view, tree view and a table view. Each view displays the same information, but in a different format. Depending on the view you choose, you can carry out different actions.

The split view

This view is automatically shown when the Object Mapping Editor is opened. It contains three panels for the three main categories. In the split view, you can:

- Assign (map) technical names to component names (→ page 133) by dragging component names onto technical names (either assigned or unassigned).

The split view is especially useful if you have many mappings – it lets you find component names and technical names separately to map them, without having to scroll through the whole tree.



-
- Create new component names for your tests (→ page 102) .
 - Rename component names (→ page 106) .
 - Create categories and map into them (→ page 124) .

The tree view

In the tree view, you can:

- Assign (map) technical names to component names (→ page 133) by dragging component names onto technical names (either assigned or unassigned).
- Create new component names for your tests (→ page 102) .
- Rename component names (→ page 106) .
- Create categories and map into them (→ page 124) .

The table view

In the table view, you can:

- Sort your object mapping information according to the technical name, component name, category or component type by clicking on the relevant table headers.
- Rename component names by entering a new name into the component name field. This new name replaces the old name for this component name anywhere you have used it in your test specification.

3.18.2.3 Working with categories in the Object Mapping Editor

Default categories in the Object Mapping Editor

The Object Mapping Editor tree view displays the following categories by default:

Unassigned component names: these are the names you have used in your Test Cases or component names that you have created (→ page 102) . They are unassigned because they have not yet been mapped to a technical name.

Unassigned technical names: these are the names that you have collected from the AUT (→ page 131) , but not yet assigned to component names.

Assigned names: there are pairs of names that have been mapped to each other. Each technical name can be mapped to one or more component names. This mapping tells Jubula which actual components you are referring to in your Test Cases.

Creating categories in the Object Mapping Editor

We recommend creating categories in the Object Mapping Editor to make your mapping work easier (→ page 233) .

- You can create categories and subcategories in the Object Mapping Editor (tree view) by:
 1. Selecting the category you want your new category to appear in (e.g. in "assigned names").
 2. Selecting "create category" from the context-sensitive menu.
 3. Entering a name for the category

You can't have two categories with the same name the same level.



- When you are mapping, you can choose which category to map into. See the next section (→ page 125) for details.

It is a good idea to create categories in the Object Mapping Editor. See the section on best practices (→ page 233) for more details.



Mapping into categories in the Object Mapping Editor

Once you have created categories in the Object Mapping Editor, you can choose to map technical names collected from the AUT directly into a category. This can help if you have created a category for each dialog/window, and you want to map all of the components from it into one category.

1. When you are in the Object Mapping Mode, right-click on the category you want to map into and select:

Map components into this category

to make the technical names you collect from the AUT appear in this subcategory.

If you have already mapped the technical name, the name will be shown in the Object Mapping Editor, but not moved into the category.



The status bar displays which category you are mapping into.



3.18.2.4 The configuration view in the Object Mapping Editor

In the configuration view in the Object Mapping Editor you can alter the object recognition for the test execution.

Understanding object recognition

Object recognition in Jubula is based on a calculation which takes various factors into account. For some applications, some factors may be more important than others, and you can change their weighting accordingly.

The factors Jubula uses in the calculation are:

Name: The name of the object within the AUT code, as given by the developer (if a name was given).

Path: The route through the AUT hierarchy to get to this component.

Context: The components in the vicinity of this component.

Threshold: This determines what percentage value a component in the AUT must have in order to be considered as the originally mapped component. Components with a value under the threshold are not considered. The component with the highest value above the threshold is chosen during execution.

Options for object recognition You can change the profile used during test execution using the combo box:

Standard: This is the default profile. It has a high value for name weight and lower values for context and path. The threshold is 85%.

Strict: In this profile, the values for name, path and context are the same as in the standard profile. The threshold is at 100%. This means that a component must exactly correspond to the originally mapped component.

Given names: In this profile, only the component name is considered. A component will only be selected if it has the same name as the originally mapped component. This profile can be used when you are sure that all of the components in the AUT have unique names.

Custom: This profile lets you move the sliders yourself. You can lock sliders to stop them being affected by other sliders.

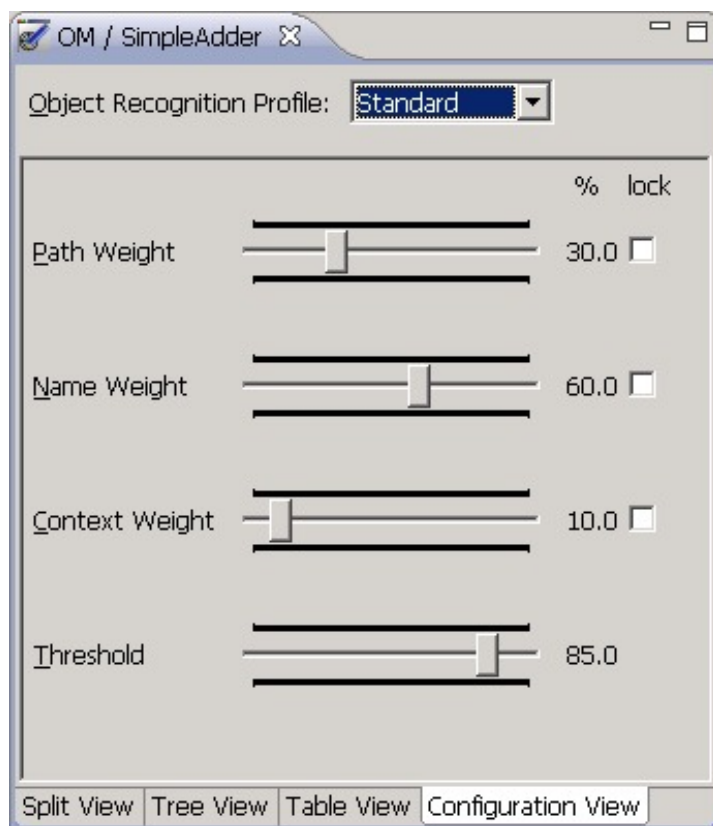


Figure 3.22: Object Mapping Configuration



Take care when manually customizing the object mapping settings. You may have test execution problems if you have set the values too strictly, or not strictly enough.

3.18.2.5 Refreshing the Object Mapping Editor

1. Select *refresh* from the context-sensitive menu in the Object Mapping Editor or press »F5« to refresh the Object Mapping Editor.
2. Confirm that you want to want to refresh the editor.

When you refresh the Object Mapping Editor, any new component names which have been added to Test Suites for this AUT are collected.

3.18.2.6 Finding components in the AUT via the Object Mapping Editor: highlight in AUT

In the Object Mapping Editor, you can search for:

- All the places where a component name has been used in the test (→ page 178)
- The specific place where a particular component name comes from in the test (→ page 178)
- The technical component in the AUT (see below).

Highlighting a component in the AUT

To search for components in the AUT, the Object Mapping Editor must be open (→ page 123) , and the AUT must be running (→ page 136) in the Object Mapping Mode (→ page 131) .

1. In the Object Mapping Editor, right-click on the technical name whose component you want to find in the AUT and select:

Highlight in AUT

from the context-sensitive menu.

2. The component whose technical name you selected will be highlighted with a green border in the AUT.

You can only highlight components that are currently visible in the AUT.



3.18.3 Deleting from the Object Mapping Editor

1. You can delete items from the Object Mapping Editor using the "*delete*" option from the context-sensitive menu.
2. You can also select the item you want to delete and press »DELETE«.
3. If you delete a component name that is still used in a Test Case for this Test Suite it will reappear when you select "*refresh*" from the context-sensitive menu.

If you want to remove all unused mappings from the Object Mapping Editor, then use the *Cleanup* option (→ page 129) .



3.18.3.1 Removing unused component names from the Object Mapping Editor

Over time, your Project will change, and these changes may mean that component names you had previously used in Test Suites, then mapped to technical names via the Object Mapping, are no longer used. This results in object mappings in which the component name is no longer used in the test for this AUT. Such component names can be recognized by the fact that *Show corresponding specification* in the Object Mapping Editor (→ page 178) returns no search results in the search result view (i.e. the component name is no longer used in the test), or if the search result view shows only Test Cases that are used in other AUT's. If you perform *Show where used* in the Component Name Browser (→ page 178) and the only search result is a mapping, then this is also a component name that is no longer used in the test.

You can delete any component names from the Object Mapping Editor that are unused in the Project for this AUT:

1. In the Object Mapping Editor, select:
`Cleanup` → `unused component names`
from the context-sensitive menu.
2. The cleanup operation starts, and a progress dialog is shown while unused names are being searched for. Depending on the size of your Project, this search may take some time.
3. If the Object Mapping Editor contains component names that are not used in any of the Test Suites for this AUT (either component names that are already mapped, or are still unassigned), then they are shown in a *Delete unused component names* dialog.



If no component names are found, then you will see a message informing you of this.

4. In the dialog to delete unused component names, you can see which component names have been identified for deletion. You can deselect individual component names to opt not to delete them. Once you have specified which component names should be deleted, click "OK".
5. The component names you specify will be deleted from the Object Mapping Editor. Save the editor to save the changes.



Removing component names from the Object Mapping Editor does not mean that the component names are deleted from the Project. However, if the deletion from the object map means that the component name is now used nowhere in the Project, then it will appear under the *Unused Component Names* category in the Component Name Browser and can be deleted from the whole Project (→ page 107)

3.18.4 Collecting components (technical names) from the AUT

Once your test specification is ready, you can *collect* components (technical names) from the AUT to *map* (assign) to the component names you used in your tests.

To collect components from the AUT, you must have:

- started and connected to the AUT Agent (→ page 21)
 - created a Test Case containing Test Steps (→ page 71)
 - added Test Cases to a Test Suite (→ page 110)
 - specified an AUT (→ page 45)
 - configured an AUT (→ page 48) if you want to start the AUT via Jubula
 - chosen an AUT for the Test Suite (→ page 111)
 - checked that your current working language is supported by the chosen AUT.
 - started the AUT to be mapped (→ page 136)
 - opened the object mapping editor (→ page 123)
1. Start the Object Mapping Mode by clicking the arrow next to the *Start Object Mapping Mode* on the toolbar and selecting which AUT (based on the AUT ID) you want to map.

 start Object Mapping Mode

If you have the same AUT running more than once, you will only be able to collect components from the AUT whose ID you chose. The object map for AUT's that are the same is, however, identical.



The status bar will show that the Object Mapping Mode is active.

2. Bring the AUT into focus by clicking on its titlebar.

For Java AUT's:

- In the AUT for which the Object Mapping Mode was started, move the cursor over components. They will be highlighted with a green border (see Figure 3.24 → page 135).

- To collect a technical name for a component, hover the cursor over the component whose name you want to collect.
- Press »CTRL+SHIFT+Q«.



You can change the key combination for the object mapping in the object mapping preferences (→ page 152) . This is a good idea if the current key combination has a specific meaning in your AUT. You can also set the object mapping combination to a mouse click if your AUT does not accept key combinations.

- If no component is collected, then you may need to extend Jubula to recognize and test the component. More information on extending Jubula is available in the Extension Manual.

For HTML AUT's:

- While the Object Mapping Mode is active, the AUT cannot be used.
 - To collect a technical name for a component, click the component whose name you want to collect.
3. In the Object Mapping Editor, the technical name for this component will appear in the *unassigned technical names* category.



If you have already mapped this component, it will be highlighted in the Object Mapping Editor.

4. When you collect a technical name, it is displayed with a colored dot in the Object Mapping Editor. The color of the dot indicates the strength of the component recognition for this component, *in the current state of the AUT* (→ page 133) .
5. Collect all the names you need from the AUT and then click the *Stop Object Mapping Mode* button on the toolbar.
6. You can now map (assign) the component names you used in your Test Cases to the technical names you have collected from the AUT (→ page 133) .



stop Object Mapping Mode

3.18.4.1 Understanding the colored dots when collecting component names in the Object Mapping Editor

When technical names are collected from the AUT, they appear in the Object Mapping Editor with a colored dot. The color of the dot corresponds to the strength of the component recognition for this component *at the time of collecting*.

A green dot signifies that the component can be found as an exact match, and that only this component was above the threshold (→ page 126) (i.e. only this component was considered as possible).

A yellow dot means that the component can be found as an exact match, but that other components were also above the threshold, i.e. this was not the only component considered possible.

A red dot means that the component can not currently be found if a test is executed. The recognition value for the component is below the current threshold.

You can use this information to identify components that will not be recognized in the current state of the AUT before running the test.

The colored dot disappears after saving. It is not a measurement of the component state over time, but only at the moment when the component was collected.



3.18.5 Mapping (assigning) collected technical names to component names

Once you have collected the components from the AUT that you will use during the test (→ page 131), you can then map them to the component names you used in your Test Cases.

1. In the Object Mapping Editor, in the split view or the tree view (→ page 123), you will see the component names you have created (→ page 102) and/or used in your Test Cases, as well as the names you have collected from the AUT. The names are grouped into categories of unassigned and assigned names (→ page 124).

2. To map a component name to a technical name, you must drag the *unassigned component name* onto the corresponding *unassigned technical name* that you want this component name to refer to.
3. When a component name has been assigned to a technical name, the joined names appear in the *assigned names* category. The component type for the component name is adjusted so that it reflects the type of component it has been mapped to (→ page 108) .



You can also "unassign" component names from technical names by dragging the component name back into the *unassigned component names* category

4. Save the changes in the editor.



You will only be able to map component names to technical names if their types are compatible (→ page 108)

3.18.6 Object mapping and AUT changes

If your AUT has changed from one version to the next, your tests may still run, depending on the nature and extent of the changes.



Please bear in mind that changing the look and feel of your AUT may lead to different technical names for the components in your AUT if you have not named the components.

If your test does not run, a few simple steps will update it. If you get a *component not found* error, or Jubula selects the wrong component during the test, remap the new component from the AUT and assign it to your component name. This updates this component for the whole test.

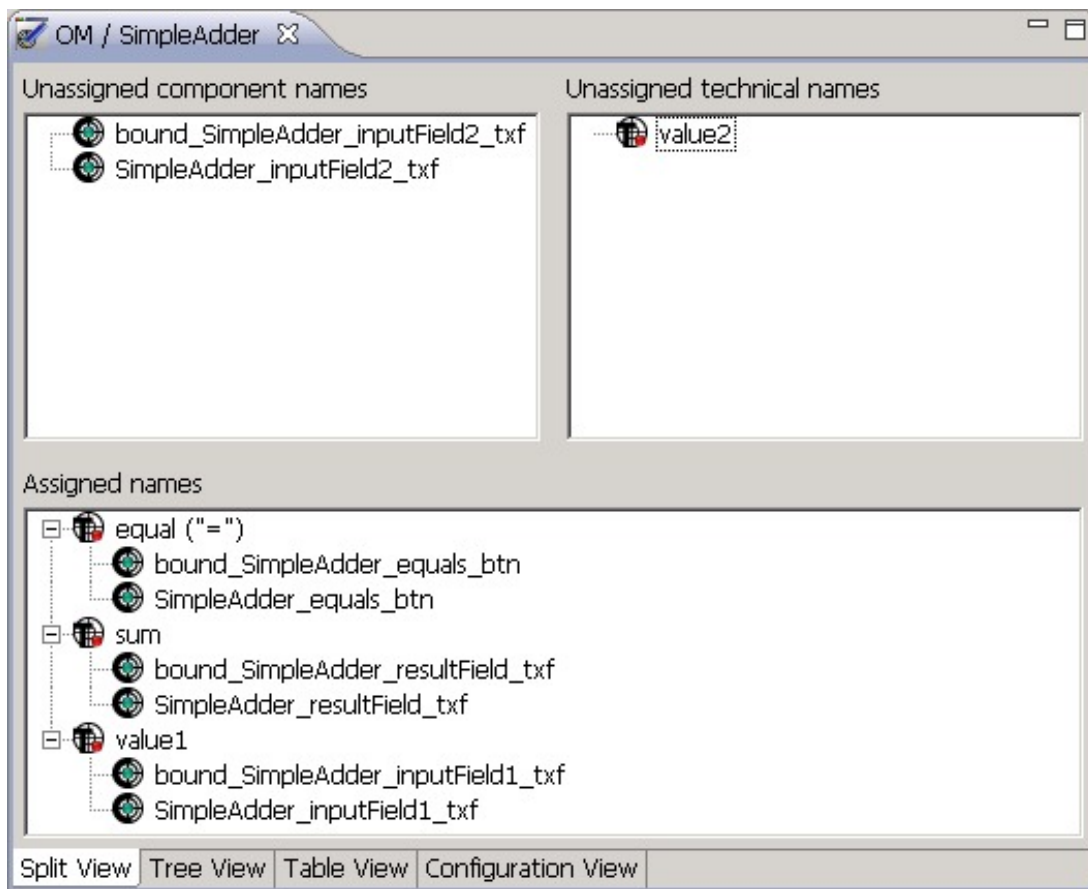


Figure 3.23: Object Mapping Editor

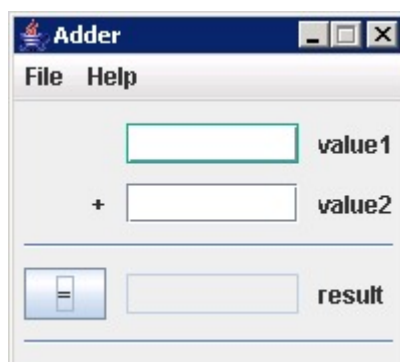


Figure 3.24: Green Border around Supported Component

3.19 Test execution

3.19.1 Prerequisites for test execution

To be able to start a test you must:

1. Create a Test Suite (→ page 110) .
2. Add Test Cases to be executed to the Test Suite (→ page 60)
3. Define an AUT (→ page 45) .
4. Configure an AUT (→ page 48) if you want to start the AUT via Jubula.
5. Choose an AUT for the Test Suite (→ page 111) .
6. Set the working language to the language you want to execute the test in.
7. Connect to the AUT Agent (→ page 23) .
8. Start the AUT (→ page 136) (either via Jubula or using the *autrun* command).
9. Complete object mapping (→ page 122) .
10. Make sure that your test data is complete in your Test Cases.



It is a good idea to set your preferences for test results in the preferences dialog (→ page 149) before test execution.

3.19.2 Starting the AUT

To start the AUT, you must:

- have defined an AUT (→ page 45)
- have connected to the AUT Agent (→ page 23)
- if you want to start your AUT via Jubula, and not using the *autrun* option, then you must have configured an AUT (→ page 48)

You can only start AUT's via Jubula which support the current working language. Change the working language if necessary by selecting a language from the list next to the language button on the toolbar.



1. On the toolbar, click on the arrow next to the "Start AUT" button.
2. Select an AUT and the configuration to start it with from the list.
3. The AUT will be started in the current working language.
4. Clicking directly on the "Start AUT" button will start the most recently started AUT (marked with a tick in the list of AUT's and configurations).



Once the AUT has been started, it will appear in the running AUT's view.

3.19.3 Starting, stopping and pausing Test Suites and Test Jobs

3.19.3.1 Starting a Test Suite or a Test Job

To start a Test Suite or a Test Job, you must:

- have started the AUT (→ page 136)
- have completed object mapping for the Test Suite or Test Suites (→ page 122)
- have entered complete test data for your Test Suites, for the language you want the test to run in.

Before starting a Test Suite or a Test Job, it is a good idea to review your preferences for the test results (→ page 153) .



Starting a Test Suite: You can start a Test Suite from the Test Suite Browser or from the Running AUT's View. Click on the arrow next to the "Start test execution" button and select a Test Suite to start from the list.

Starting a Test Job: You can start a Test Job from the Test Suite Browser. Click on the arrow next to the *"start test execution"* button and select a Test Job to start from the list.

 start test



The amount of Test Suites or Test Jobs in the list for the *"start test"* button depends on your selection. In the Running AUT's View, only Test Suites for the currently selected AUT will be shown. In the Test Suite Browser, if you have selected a Test Suite or a Test Job, only the selected item will be startable.


When a test is running, you can view the progress of the test in the *Progress View*. This is especially useful if the ITE and the AUT are running on different machines: you can see how much longer the test will run without having to switch to the machine the AUT is running on.

3.19.3.2 Stopping a Test Suite or Test Job

 stop test

To stop the test at any point, click on the *"stop test execution"* button on the toolbar. The test stops automatically when it finishes or when an error occurs whose Event Handler has *exit* as a reentry type.

3.19.3.3 Pausing a Test Suite or Test Job

 pause test

1. To pause a test, click on the *"pause test execution"* button on the main toolbar.
2. To continue with the test, press the *"Pause test execution"* button again.



You can opt to automatically pause the test execution when an error occurs via the toolbar (→ page 139) .



If the client and AUT Agent are installed on the same machine, moving the mouse will disrupt the test.

3.19.4 Interactive test analysis

If you are executing tests interactively, you can use Jubula to help you analyze any errors that occur.

3.19.4.1 Pause on Error

On the toolbar in the Jubula client, click the *"Pause on Test Execution Errors"* button or press »F9« to cause the test to pause when it encounters an error.

Failed Test Steps that have a retry Event Handler will not be paused unless the final retry fails.



3.19.4.2 Continuing after an error

Once you have analyzed the error in the AUT or in the test specification, you have two options:

Continue with the Event Handler: Press the *"Pause test execution"* button on the toolbar (→ page 138) to continue with the test. This will result in a Event Handler (either a default Event Handler or one of your own) being activated.

Continue without Event Handler: Press the *"Continue without Event Handler"* button or »F8« to continue with the test *as if the failed Test Step had been successful*. No Event Handler is activated. This option is useful if the Event Handler that would be activated would restart the AUT or exit the test, but you are either able to ignore the error for the moment or make changes in the AUT to ensure that the test will be able to continue.

Any changes you make to the test specification while the test is paused will not become valid until the next test execution start.



3.19.5 Altering the speed of test execution

You can slow down the speed of test execution to be able to watch your tests by entering a *step delay* in the Properties View for the Test Suite (→ page 111) .

3.20 Working with test results

3.20.1 The Test Result View

The Test Result View Figure 3.25 → page 142 tracks the test progress for each Test Suite, marking each Test Step as successful or unsuccessful.

When the test begins, the Test Steps to be executed appear in the Test Result View. As each Test Step is executed, it is marked with a question mark. If the Test Step is successful, it is marked with a green tick. Failed Test Steps are marked with red crosses. Test Steps which succeeded after retrying (→ page 146) are marked differently to show that they were successful after retrying.

Navigating in the Test Result View

- Single click on an item in the Test Result View to see details for that item in the Properties View.
- Double-click on an item in the Test Result View to be taken to this item in the Test Suite.
- You can clear the Test Result View by clicking the "clear" button in the tab of the view.
- Use the arrows on the toolbar to spring to the next or previous error in the Test Result View (you can also use the combinations »ALT+UP« and »ALT+DOWN«).
- Each node in the Test Result View shows the amount of time taken to execute this node. You can turn this option off in the preferences (→ page 155) .

You can also see screenshots of errors that were taken by Jubula by selecting the screenshot from the tree in the test result tree. You will need to have the Image View open to see the screenshot.



For preferences about the Test Result View, see the section on the preference pages (→ page 153) .

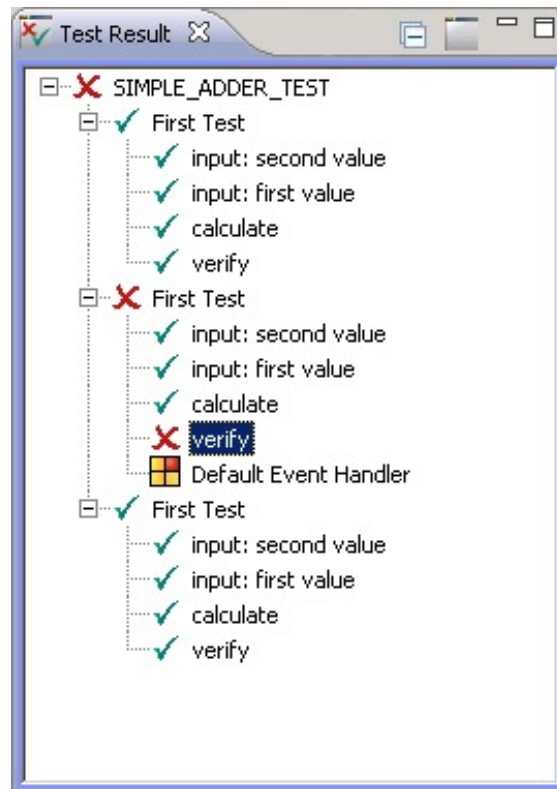


Figure 3.25: Test Result View

3.20.2 XML and HTML reports

You can create XML and HTML reports for your tests in one of two ways:

1. In the Test Result Summary View (→ page 145) .
 2. You can specify that XML and HTML reports should be created when a test runs in the preferences (→ page 153) .
- We recommend using the option to export test results from the Test Result Summary View, as this does not involve defining any external files for your test environment.

3.20.3 Working with the Test Result Summary View

Jubula stores a test result summary and full details about the whole test for each test run. Individual Test Suites within a Test Job are counted as one test run each.

You can see the test result summaries in the Functional Test Reporting Perspective, in the Test Result Summary View.

3.20.3.1 Re-opening the test result view for a test run

1. If the full details for a test run are still available (the column *Details* shows *true*), you can re-open the full test result by double-clicking on the entry in the Test Result Summary View or by clicking the button to open the report in the top right-hand corner of the view.
2. In the test result, you can navigate through the test results using the mouse or using the toolbar buttons to spring to the next or previous error. In this way, you can view any screenshots in the Image View and details about errors in the Properties View.
3. Each node in the Test Result View shows the amount of time taken to execute this node and any parameter values that were used at this node. You can turn these options off in the preferences (→ page 155) .

In the Project properties, you can specify how often the full reports should be automatically deleted from the database (→ page 33) .



3.20.3.2 Filtering and sorting in the Test Result Summary View

The Test Result Summary View shows a summary of each Test Suite that has run.

Sorting entries in the Test Result Summary View

Click on a column header to sort the table according to the values in this column.

Showing and hiding columns in the Test Result Summary View

In the context-menu on the column headers in the Test Result Summary View, you can select which columns you want to show in the view. Your settings for this view are saved in your workspace.

Filtering in the Test Result Summary View

At the top of the Test Result Summary View there is a filter. Choose which column you want to filter, and enter the value you want to filter by (use * as a wildcard).

3.20.3.3 Changing the amount of result summaries shown in the Test Result Summary View

In the test result preferences (→ page 153) you can alter the amount of days that a test result summary is shown. The top of the Test Result Summary View shows the amount of days that test result summaries are being shown for. Any unshown summaries are still present in the database and can be shown by increasing the amount of days.

3.20.3.4 Changing the relevance of a test run

You can change the relevance of a test run (i.e. whether it is exported with the Project or not) in the Test Result Summary View.

1. Select the test run whose relevance you want to change.
2. Select:

Toggle relevance
from the context-sensitive menu.



You can only change the relevance for one test run at a time!

3.20.3.5 Refreshing the Test Result Summary View



Use the "Refresh" button at the top right hand corner of the Test Result Summary View to refresh the view after tests have been run.

3.20.3.6 Deleting test runs from the Test Result Summary View



Select one or more rows in the Test Result Summary View and use the "Delete" button in the top right hand corner of the view to delete these test runs.

3.20.3.7 Exporting test results from the Test Result Summary View as HTML and XML reports

- Select one or more test reports and select **Export** from the context-sensitive menu or click the "Export" button on the toolbar of the Test Result Summary View.
- In the dialog that opens, enter or browse to a directory where the test results should be saved.
- When you click "Finish", the result reports for the selected test runs will be exported to the directory you entered.
- For each test run, an XML and an HTML file are created.

In order to export test results from the Test Result Summary View, details for these reports must still be in the database. This will be shown as *true* in the *Details* column in the Test Result Summary View.



3.20.3.8 Entering comments for test runs in the Test Result Summary View

You can add a comment to a test run in the Test Result Summary View by:

- Selecting a test run in the Test Result Summary View and then selecting **Edit comment** from the context-sensitive menu.
- In the dialog that opens, enter a title for the comment and a longer description in the details area.
- When you click "OK", the title of the comment is shown in the *Comment* column.
- You can view the details for a comment by reselecting **Edit comment** from the context-sensitive menu.

3.21 Dealing with errors in tests: Event Handlers

Event Handlers are Test Cases used to deal with deviations during test execution. When an error occurs, the current Test Case is searched for an Event Handler for that error type. If none is found, the parent Test Case is searched, and so on. If no Event Handler for the test is found, then a default Event Handler is activated. Default Event Handlers are specified in the Test Suite properties (→ page 111) .

The advantage of adding your own Event Handlers is that you can define specific behavior for certain errors in each Test Case. An Event Handler can be an empty Test Case, or can contain Test Cases to be executed when the error occurs.

For each Event Handler, you must specify:

an event type to define what sort of error should activate this Test Case (→ page 147) .

a reentry type to define how the test should continue once the Event Handler has been executed (→ page 148) .

3.21.1 Adding Event Handlers to a Test Case

This section deals with adding Event Handlers to Test Cases. For information on using Event Handlers meaningfully to ensure that your test execution is as robust as possible, see the section on Event Handlers in the Best Practices section (→ page 234) .

1. Open the Test Case Editor by double-clicking on the Test Case you want to edit in the Test Case Browser.
2. Select
`Add → Test Case as Event Handler`
from the context-sensitive menu in the editor or use »CTRL+ENTER«.



You can also drag the Test Case you want to use straight into the Event Handlers area of at the bottom of the Test Case Editor.

3. Choose a Test Case to add from the dialog that appears.

You can't add a Test Case to itself as an Event Handler, because this would create an infinite loop.



4. Select an event type from the combo box (→ page 147) .
5. Select a reentry type from the combo box (→ page 148) .

For each Test Case you can only add one Event Handler for each event type.



6. Click "OK".
7. The Event Handler appears in the lower half of the Test Case Editor.
8. Save the changes in the editor.



Event Handler

3.21.2 Event types

The event type is the type of error the Event Handler will react to. There are four events in Jubula. They correspond to common errors which happen during test execution.

Component not found If a component in the AUT cannot be found, an error occurs. The component might not be found if it has been changed, is not (yet) visible or has been deleted.

Check failed For some actions, you can specify an expected value to check. If the value you specify is not true, then a *check failed* error occurs.

Action error Action errors occur when an action cannot be carried out on a component, when the data is invalid (e.g. an invalid menu path) or when an action takes too long and there is a timeout.

Configuration error Configuration errors occur when there is an internal problem, for example a changed implementation class in the XML file.

3.21.3 Reentry types

The reentry type specifies how the test should continue once the Event Handler Test Case has been executed.

Break The test execution leaves the Test Case in which the error occurred and continues at the next Test Case or Test Step.

Continue The test execution carries on at the next Test Case or Test Step. This is a good option when the error is relatively unimportant and does not affect the following Test Steps.

Exit The test execution is stopped. Use this when the error is so severe that the test cannot be continued.

Return The test execution leaves the Test Case in which the activated Event Handler is nested. This could be the current Test Case or one higher up in the hierarchy. This option is useful if you have a use case, which contains Test Cases to test a particular area or function. You can decide to leave this part of the test, and carry on at the next use case. *Return* will continue at the next Test Suite in a Test Job if it is activated as a default Event Handler or if there are no further steps in the current Test Suite.

Pause The test execution is paused. To restart the test, click the *pause* button in the toolbar in the Jubula client.

Retry The failed Test Step is repeated as many times as you specify in the Properties View. If the Test Step is successful on repeating, it is marked as successful after retrying. If the Test Step fails on all retries, the error type is passed on to the next parent Test Case and this Test Step (and therefore the test) is marked as failed.



See the section on best practices to learn how to use Event Handlers to document errors and ensure that the test continues running (→ page 234) .

3.22 Preferences

Open the preferences dialog (Figure 3.26 → page 149) by selecting:

Window → Preferences .

3.22.1 Test preferences

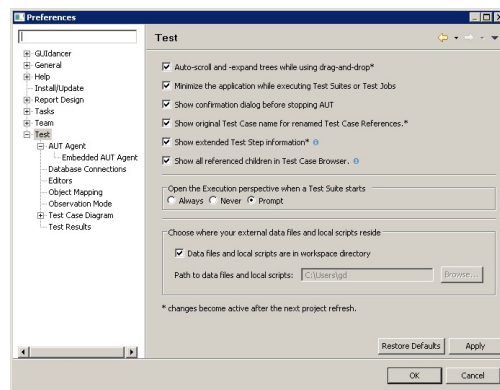


Figure 3.26: Jubula Preference Dialog

In the preferences dialog, select *Test* from the tree on the left hand side.

From this page, you can configure your preferences for:

Auto-scrolling and -expanding: When the checkbox is marked, the views and browsers in Jubula automatically scroll in the direction you move the mouse when you are dragging and dropping. Trees will also be automatically expanded when you hover over them while dragging items.

Minimizing the ITE: When the checkbox is marked, Jubula is automatically minimized when test execution begins. This is useful if you are letting tests run on the same machine you are specifying on.

AUT confirmation dialog: When the checkbox is marked, a dialog appears to check if you are sure when you click the "stop AUT" button.

Original Test Case name: When the checkbox is activated, you can see the original name of a referenced Test Case in brackets behind a new name you enter for the Test Case. This can help you to see and search for Test Cases you have reused.

Test Step information: When this checkbox is activated, you see the details about the Test Step (the component name and type, and the action) in square brackets behind the Test Step name. If you do not want to see these details, you can deactivate this checkbox.

Show referenced children: When this option is not active, you can only see referenced parent Test Cases in the Test Case Browser. The referenced Test Cases contained in these Test Cases are not displayed. This action can be useful if you want to improve the speed of working with the ITE.

Switching to the Functional Test Execution Perspective:

When the test begins, Jubula can automatically change to the Functional Test Execution Perspective. You can choose to always be asked, to always change, or to never change.

Data files location: You can specify a location where external data files (e.g. Excel files) are held, or use the workspace directory as the base location.



The advantage of using the workspace as a location for your data files is that you can view these in the navigator view directly in the ITE. Windows users can even open Excel files from the ITE using the In-Place editor.

3.22.2 AUT Agent preferences

You can find the AUT Agent preference page under:
Test-AUT Agent
in the preferences dialog.

1. In the AUT Agent preference page, you can add, edit and delete AUT Agents and port numbers.
2. The hostnames and port numbers you enter here are displayed in the drop-down list for the "*Connect to AUT Agent*" button on the toolbar. You will be able to connect to these hostnames on these port numbers when you have started an AUT Agent on them (→ page 22) .

The AUT Agent preference page can also be accessed from the AUT configuration dialog which you can open from the Project properties.



3.22.3 Embedded AUT Agent preferences

You can find the preference page for the embedded AUT Agent under:

Test-AUT Agent-Embedded AUT Agent
in the preferences dialog.

On this page, you can enter the port number that the embedded AUT Agent (→ page 23) should use when you connect to it (→ page 23) . The default is 60000.

3.22.4 database preferences

You can find the database connection preference page under:

Test-database Connections
in the preferences dialog.

When you start Jubula for the first time, it is automatically configured to use an embedded database. If this is the only database configured, then the database login is performed automatically.

In this preference page, you can configure other database connections for your productive database.

We do not recommend using the embedded database for productive use of Jubula.



3.22.4.1 Adding, editing and removing database configurations

To add a new database for use with Jubula:

1. In the database Connections preference page, click "Add".
2. In the dialog that appears, enter a meaningful configuration name.

3. Select the type of database that this configuration is for.



Jubula is thoroughly tested with Oracle. PostgreSQL and MySQL have been successfully tried out with Jubula, but we cannot guarantee their usage in productive environments.

4. Based on the database type you choose, you must enter more details about the database connection. Speak to your database administrator if you are unsure of the connection details.

You can edit and delete database configurations using the buttons in the database Connections preference page.

3.22.5 Editor preferences

In the *Test - Editors* preferences, you can choose whether you want items (Test Cases and Test Steps) to be placed at the bottom of the tree or after the selected item when you use the *add* option.

3.22.6 Object mapping preferences

You can access the object mapping preferences from *Test - Object Mapping* in the preferences dialog.

1. Select whether you want Jubula to display how many nodes are contained in each category in the Object Mapping Editor.
2. Choose which keystrokes or mouse buttons you want to use to collect components in the Object Mapping Mode.



If your AUT does not accept keystrokes, set the object collection preference to a mouse button combined with a modifier key.

3.22.7 Observation mode preferences



The observation mode can only be used for Java AUT's.

Start/stop check mode: Choose which keystroke you want to use to start and stop the check mode when you are observing Test Cases in Java AUT's.

Check component: Choose which keystroke you want to use to show the check dialog when you are in check mode. The check dialog lets you choose what property of the selected component you want to check.

Show console: The console provides information on the actions that have been observed. This is especially useful if your AUT is running on a different machine to your client.

Trigger for replace text: Replace text actions are observed when the focus moves from the text component to another component. Depending on your application, this may not happen automatically in some situations. Enter triggers here which you can use to tell Jubula that you wish to observe a replace text action, even if the focus does not move from the text component.

3.22.8 Test result preferences

You can open the test result preference page by selecting *Test - Test Results* from the preference dialog.

1. Choose whether you want the Test Result View to open when testing begins.

The Test Result View follows the test execution and shows which Test Steps were successful and which failed.

2. Choose whether you want to track the test execution progress in the Test Result View. When this option is activated, the Test Result View scrolls to follow the test progress.
3. Choose whether you want Jubula to automatically create a screenshot when a test encounters an error. Screenshots are automatically saved in the database and can be viewed



using the Image View when you select the failed Test Step in the Test Result View.

Screenshots are only taken of the primary monitor. Multiple monitors are not supported for screenshots.

4. Choose whether you want to create HTML and XML reports for each execution. When you activate this choice, decide whether you want full reports or just the errors to be shown. Enter a directory for the reports.
5. Decide whether you want all test runs or no test runs to be marked as relevant, or whether you want to be prompted each time. Non-relevant test results are not exported with the Project.
6. Enter the amount of days that test result summaries should be displayed for. This gives you a better overview of the most recent results in the Test Result Summary View.

3.22.9 Importing and exporting database preferences

You can export and import your database preferences for Jubula. This makes migrating to new versions easier, as you do not have to reenter any database preferences.

To export your database preferences:

1. Select:
`File → Export`
from the main menu in the ITE.
2. In the dialog that appears, select: *General/Preferences* and click "Next".
3. Select the *Database Connections* option and browse to a place in the file system where you want the preferences to be saved.
4. Click "Finish".

To import your database preferences:

1. Select:
`File → Import`
from the main menu in the ITE.

2. In the dialog that appears, select: *General/Preferences* and click "Next".
3. Browse to the preference file you wish to import.
4. Select the *Database Connections* option and click "Finish".

3.22.10 Label decoration preferences

You can open the label decoration preferences by selecting *General - Appearance - Label Decorations* from the preference dialog.

From this preference page, you can alter your preferences for:

Completeness Check Decorator: We recommend leaving this decorator active, as it shows you red crosses and tooltips in the Test Suite Browser when Test Suites and Test Jobs are incomplete.

Test Result Duration Decorator: Deactivate this option to not see the execution time required for each node in the Test Result View. The times shown are for the execution of each node, including its children. You can use this information to see whether specific Test Cases are taking longer than you expect them to.

Test Result Parameter Decorator: Deactivate this option to not see the parameter values used at this node (i.e. either entered at this level, or actually used at this place) in the Test Result View. You can use this information to easily see which data were used for your Test Cases, without having to click through each individual node in the Test Result View. This can be especially useful if you have one Test Case that runs multiple times with different datasets.

3.22.11 General/Keys preferences

In the *General* preferences, you can select the *Keys* option to be able to edit the key binding for shortcuts in Jubula. Here, you can create your own shortcuts or alter the current shortcuts.

3.22.12 Help preferences

1. Choose whether you want to have help displayed in an external browser. This lets you see more information at once.
2. For context-sensitive help for windows and dialogs, choose how you want the help to appear.
3. Choose how many search results should be shown when searching through the help system.

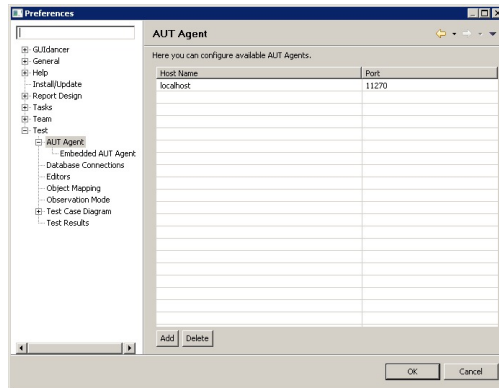


Figure 3.27: AUT Agent Configuration Dialog

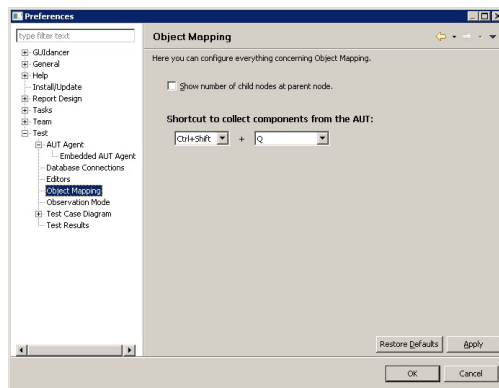


Figure 3.28: Object Mapping Preference Dialog

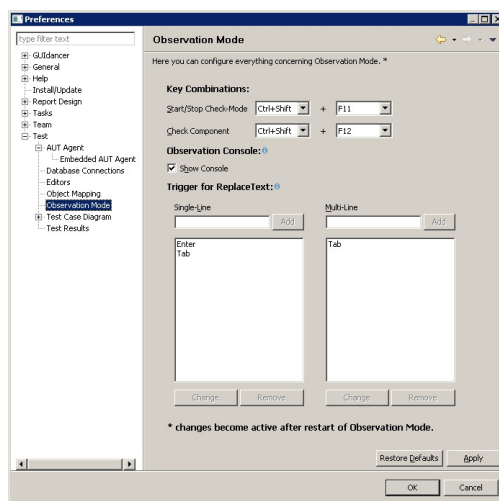


Figure 3.29: Observation Mode Preference Dialog

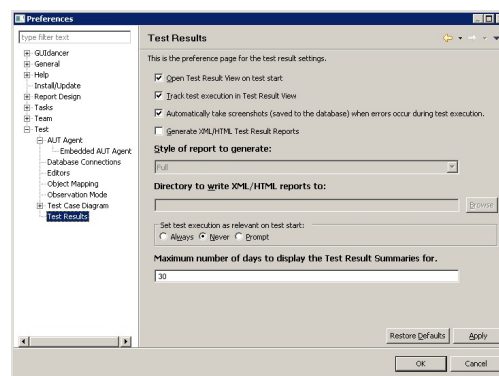


Figure 3.30: Test Result Preference Dialog

3.23 Observing Test Cases

3.23.1 Tips and tricks for using the observation mode

We have designed the observation mode to help you get started with your tests and to help you understand what sort of user actions correspond to Jubula actions.

Jubula is first and foremost a keyword-driven tool, and the library of Test Cases installed with Jubula means that you have various benefits over simple recording:

- You can create tests without needing the AUT - you don't have to wait for each version of the AUT to begin test specification.
- Your tests aren't so dependent on the actual implementation of the AUT, so they can be a lot more general in terms of data, component implementations etc. and therefore a lot more robust and maintainable.
- Using keywords encourages you to think about your test structure a lot more, which also helps maintenance later.

Nevertheless, we understand that you may want to observe some Test Cases. Here are some tips that might help you:

- Think about your test structure at the beginning: what modules will you want to reuse? Start by observing small Test Cases, e.g. a Test Case to login, a Test Case to open a dialog etc.
- Check while you are observing that the actions you carry out are observed in the way you meant.
- Supplement your observed Test Cases with other Test Cases from the library of Test Cases in Jubula.
- Refactor as you go! If you have recorded a Test Case with specific data, but you will want to use it for any type of data, then add a reference for the data.

3.23.2 Starting observing

To be able to observe Test Steps, you must:



- define an AUT (→ page 45)

Only Java AUT's can use the observation mode.

- configure an AUT (→ page 48) if you want to start it via Jubula.
- set the working language to the language you want to observe in.
- start the AUT you want to observe from (→ page 136) (either via a configuration (→ page 48) or using the *autrun* command (→ page 52)).



start observation

Once you have completed these steps, you can select the "*observe Test Case*" button on the main toolbar.

1. When asked, enter a name for the observed Test Case.
2. The status bar will show that you are in the observation mode.
3. A Test Case Editor for the observed Test Case will appear.
4. Switch to the AUT and activate it by clicking once in the title bar.
5. You can now observe Test Steps.



If you have a Test Case open in the Test Case Editor, the Test Steps will be observed into this open Test Case.

3.23.3 Observing tests in Java AUT's



If you have not already done so, we recommend reading the tips section for the observation mode before beginning observing (→ page 159) .

1. In Java AUT's (Swing and SWT/RCP) the observation mode will automatically record your actions in the user interface. Each action is created as a Test Step in the Test Case Editor for this observed Test Case.

See the section later on performing check actions in the observation mode (→ page 162) .



2. You can also see which actions have been recorded in the console (Figure 3.31 → page 161).

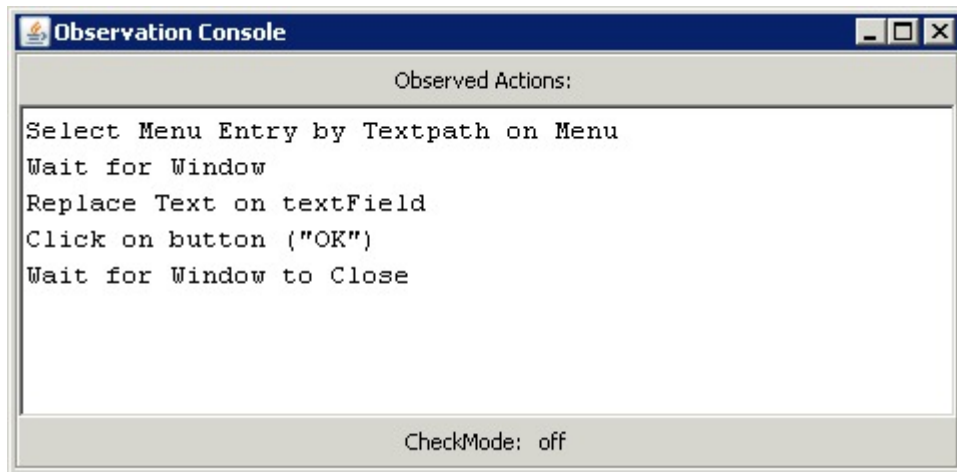



Figure 3.31: The observation console

If you are creating tests for SWT and RCP AUT's, check that you have set the keyboard layout correctly in the Project properties (→ page 50) and that you have defined the right toolkit for the Project (→ page 34) .



3. Component names for your components are automatically generated and assigned to the technical names from the AUT when you observe Test Steps. If Jubula notices that you have already created and mapped a component name for a technical component, it will use this name instead of creating a new one.
4. Once you have recorded the actions you need, stop the observation mode by clicking on the "stop observing Test Case" button on the main toolbar.
5. Save the Test Case editor containing the Test Steps you have just observed.
6. Check the Test Steps and their parameter values which have been recorded. You will notice that any text that

 stop observation

contains non-alphanumeric characters is enclosed in single quotes. Single quotes are used by Jubula to cancel any meaning of the characters within the quotes.



Run the test that you have just recorded to see if it works as you intended. If not, you may need to make some changes to the parameter values, or you may have to supplement the Test Case with Test Cases from the library (→ page 73) .

3.23.3.1 Actions that cannot be recorded

A few actions cannot be recorded in the current version. These include:

- Key combinations that are used as shortcuts in SWT applications.
- Click counts on trees. The select actions are correctly recorded, but the click count is set to 0 and must be manually adjusted.
- Components that contain texts that are too long (more than 3999 characters).
- Actions on native dialogs e.g. file choosers.

3.23.3.2 Performing checks in the Java observation mode

You can perform checks in the observation mode by taking the following steps:

1. Start the check mode by pressing »CTRL+SHIFT+F11«. This key combination can be changed in the preferences (→ page 153) .
2. In the observation console, the check mode will be marked as *on*.
3. In the AUT, components will be highlighted with a red border (Figure 3.32 → page 163) .



While the check mode is active, no other actions will be recorded.

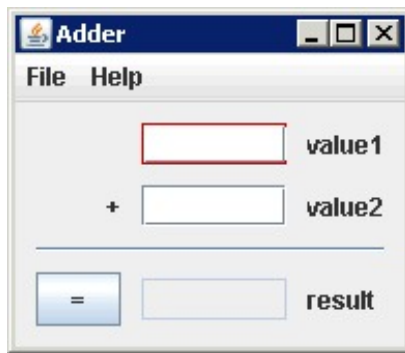


Figure 3.32: Red borders in the check mode

4. Hover over the component you want to execute a check on and press »CTRL+SHIFT+F12«. This key combination can be changed in the preferences (→ page 153) .
5. A dialog will appear showing the type of component you are performing the check on.
6. From the dialog, select the check action you want to perform and enter any parameters the check action needs. Many check actions have predefined parameters based on the state of the AUT.
7. When you have specified your check action, choose whether you want to close the dialog and continue in the check mode (*check on*) or whether you want to stop the check mode when the dialog closes (*stop checking*).

You can manually stop the check mode using the same key combination as you used to start the check mode (»CTRL+SHIFT+F11« by default).



8. The check action you specify will be added to the Test Case Editor.


3.24 Working with the Problem View

3.24.1 The Problem View

The Problem View (Figure 3.33 → page 164) is in the middle of the specification perspective, at the bottom, by default. It shows three types of message:

 information

 error

 warning

- Information
- Errors
- Warnings

Right-clicking on a message and selecting *Quick Fix* will either solve the problem or take you to the place where you can solve the problem, if this is possible.

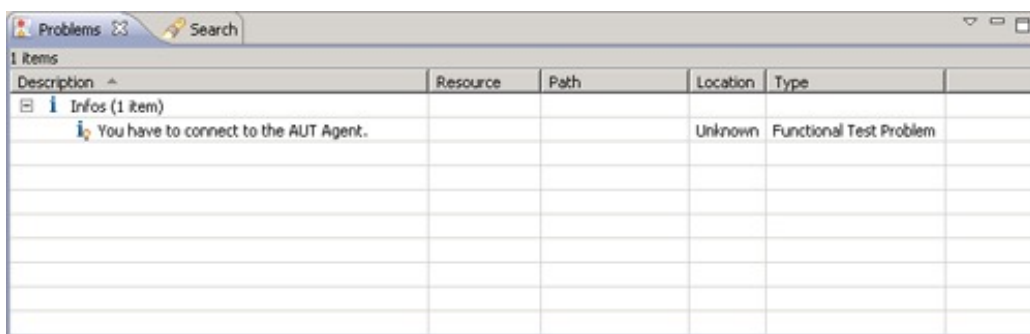


Figure 3.33: Problem View

3.25 Working with the Teststyle guidelines

The Teststyle function in Jubula provides assistance when writing tests. The Teststyle guidelines in Jubula are a set of standard rules dealing with the creation of Test Suites.

3.25.1 Activating Teststyle for a Project

You can activate Teststyle for a Project in the Project properties:

1. Open the Project properties via:
`Test → Properties`
2. Select *Teststyle* from the left.
3. Using the checkbox at the top of the dialog (Figure 3.34 → page 166), you can centrally activate or deactivate Teststyle for the whole Project.

By default, Teststyle is activated for all Projects.



4. Click "OK" in the Project properties to save the changes.
5. If your current Project flouts any of the guidelines that are set, then you will see entries in the Problem View notifying you of the places where guidelines are being flouted. For more information on working with the Problem View to fix tests, see the section later (→ page 169) .

The Teststyle settings are central for the whole Project and will be seen by all users of the Project.



3.25.2 Configuring Teststyle for a Project

When Teststyle has been activated for a Project (→ page 165) , you can configure:

- which guidelines should be used (→ page 167)

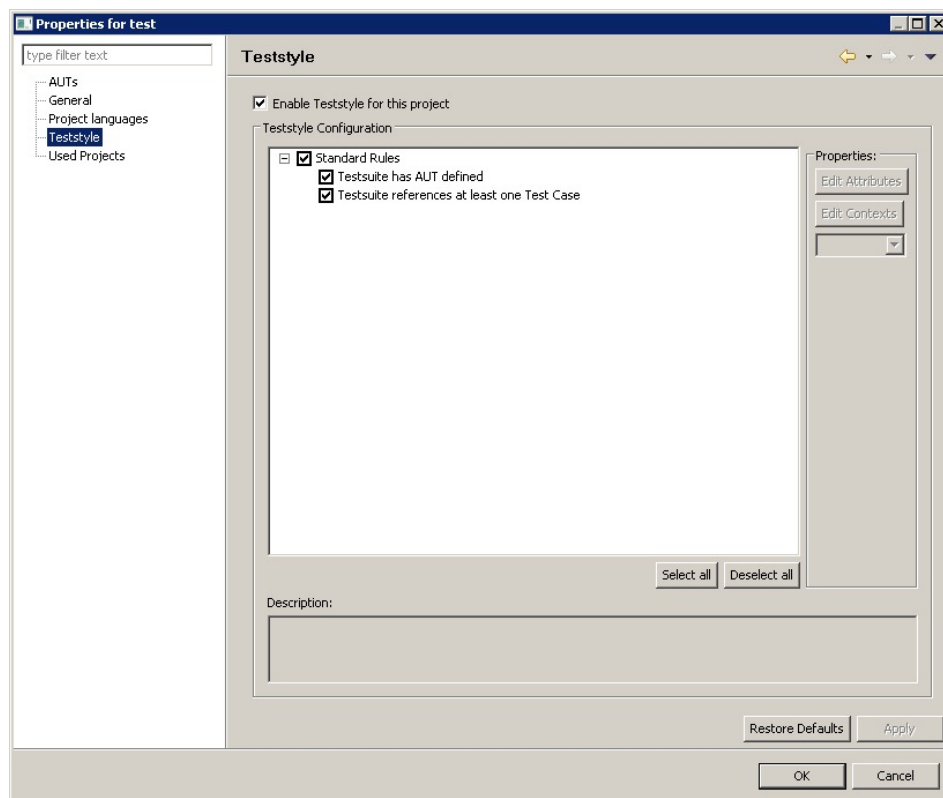


Figure 3.34: Teststyle

- what kind of a message a flouted guideline should show (information, warning or error) (→ page 167)
- the attributes for a guideline (→ page 168)
- the contexts a guideline should be active for (→ page 169)

3.25.2.1 Activating and deactivating individual guidelines

In the Project properties, under the *Teststyle* section, you can activate and deactivate guidelines for use in the Project:

1. When Teststyle is enabled for the Project (→ page 165) , select the checkbox for an individual guideline to activate it for the Project.
2. Deselect a checkbox to deactivate the guideline.

When you select a guideline, the description text gives you more information on what the guideline is for.



3. You can (de)select all the guidelines in a category by activating or deactivating the checkbox for the category name.
4. You can also use the buttons "*Select All*" and "*Deselect All*" to (de)select all the guidelines.
5. Click "*OK*" in the Project properties to save the changes.
6. If your current Project flouts any of the guidelines that are set, then you will see entries in the Problem View notifying you of the places where guidelines are being flouted. For more information on working with the Problem View to fix tests, see the section later (→ page 169) .

3.25.2.2 Setting the message level for guidelines

For each guideline in the Project properties, you can decide how Jubula should notify you that the guideline is not being upheld.

The choices available are:

Information: Information is shown in the Problem View using a blue icon. Elements in the test are not decorated with information messages.

Warning: Warnings are shown in the Problem View using a yellow icon. Warnings are also shown on elements in the test (e.g. Test Cases).

Errors: Errors are shown in the Problem View using a red icon. Warnings are also shown on elements in the test (e.g. Test Cases, Test Suites. If a Test Suite contains a Teststyle error, it will not be executable via the ITE.

To set the message level for a guideline:

1. In the Project properties, under the section *Teststyle*, select the guideline you want to configure.
2. In the combo-box on the right-hand side, select the level of message you want to receive if this guideline is not upheld.
3. Click "OK" in the Project properties to save the changes.

3.25.2.3 Configuring the attributes for guidelines

For many of the guidelines in the Project properties, you can define the values or quantities for the aspects it checks. For example, you can edit the attribute *Maximum number of parameters* for the guideline that specifies the maximum amount of parameters on a Test Case.

To edit the attributes for a guideline:

1. In the Project properties, under the section *Teststyle*, select the guideline you want to configure.
2. Click "*Edit attributes*" to open the attributes dialog.
3. In the dialog (Figure 3.35 → page 170), you can see and edit the values for any editable attributes for this guideline.
4. Confirm your changes in this dialog using "OK".
5. Click "OK" in the Project properties to save the changes.



If you enter an invalid value, the default value will be used in its place.

3.25.2.4 Configuring the contexts for guidelines

For many of the guidelines in the Project properties, you can set the contexts they should be valid for. Some of the available contexts include Test Cases, Test Suites, or the whole Project. To edit the context for a guideline:

1. In the Project properties, under the section *Teststyle*, select the guideline you want to configure.
2. Click "*Edit context*" to open the context dialog.
3. In the dialog (Figure 3.36 → page 170), you can see and edit any available contexts for this guideline.
4. To activate a context, select the checkbox next to it. To deactivate a context, deselect its checkbox.
5. Confirm your changes in this dialog using "*OK*".
6. Click "*OK*" in the Project properties to save the changes.

If you enter an invalid value, the default value will be used in its place.



3.25.3 Working with the Problem View to view and fix Teststyle problems

Once you have activated (→ page 165) and configured (→ page 165) Teststyle for a Project, you will be informed when your chosen guidelines are not being upheld. The central place for viewing your Teststyle information, warnings and errors is the Problem View.

For each Teststyle message, you will see an entry in the Problem View, with the information on where the guideline is not being upheld. You can use the Problem View to view the rule that has been flouted and also to fix the problem using the Quick Fix option.

3.25.3.1 Viewing the broken Teststyle rule from the Problem View

1. Select the entry in the Problem View whose rule you want to view.

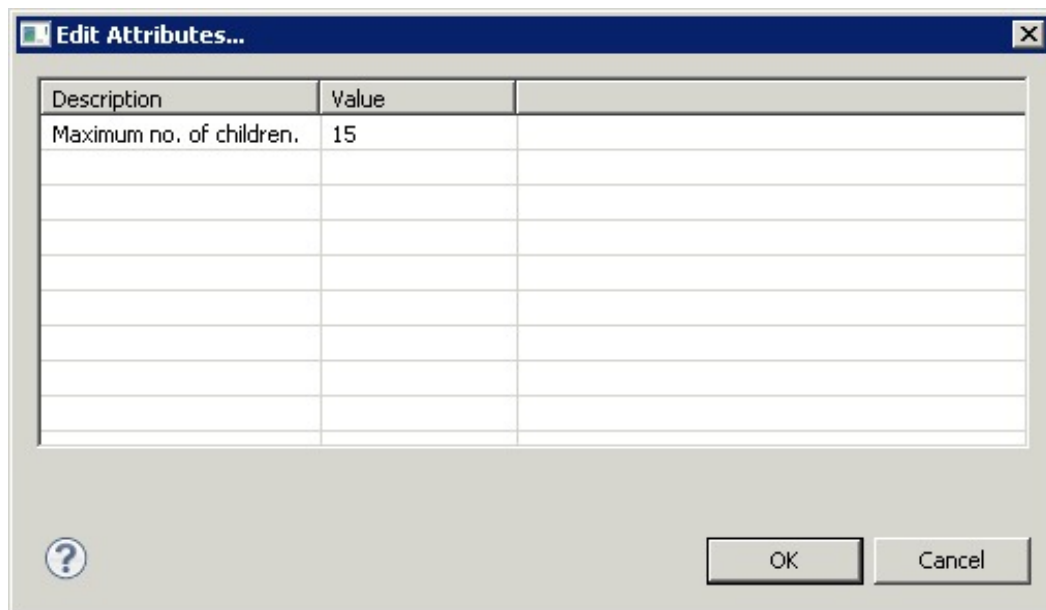


Figure 3.35: Edit Attributes

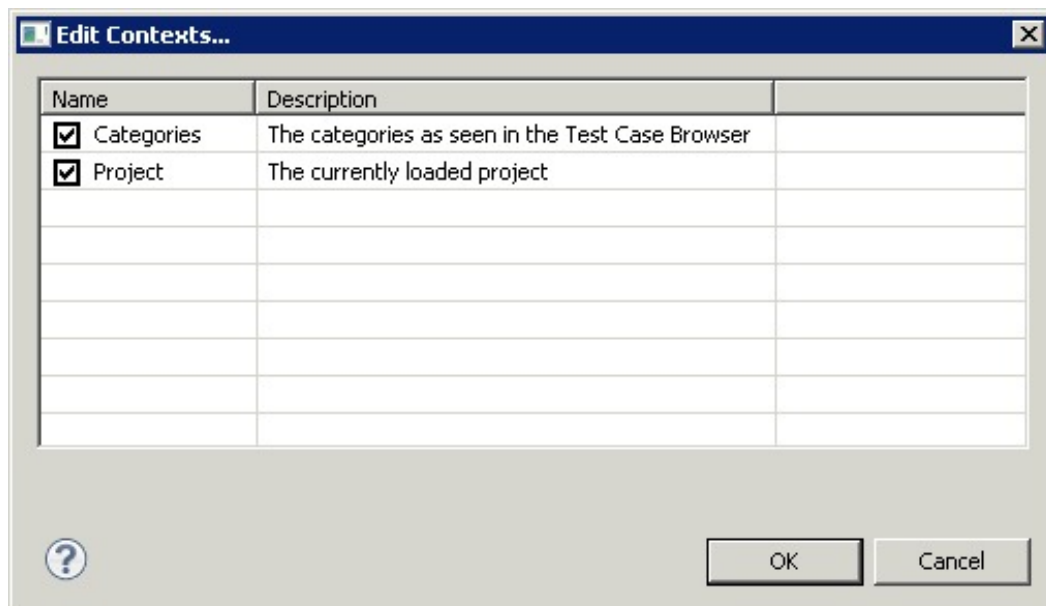


Figure 3.36: Edit Contexts

2. Right-click in the Problem View and select:
`Show Teststyle Rule`
from the context-sensitive menu.
3. The Project properties dialog appears, opened at the Teststyle page. The guideline that resulted in the entry in the Problem View is selected.
4. From here you can adapt the guideline if required (→ page 165) .

3.25.3.2 Using Quick Fix to fix the problem

1. Select the entry or entries you want to work on from the Problem View.
2. Right-click in the Problem View and select:
`Quick Fix`
from the context-sensitive menu.
3. The Quick Fix dialog Figure 3.37 → page 172 will appear with the options available to fix the problem. The options may range from opening an editor to automatically fixing the problem.
4. Select the option(s) you want to carry out and click "OK".
5. The option(s) you chose will be executed.

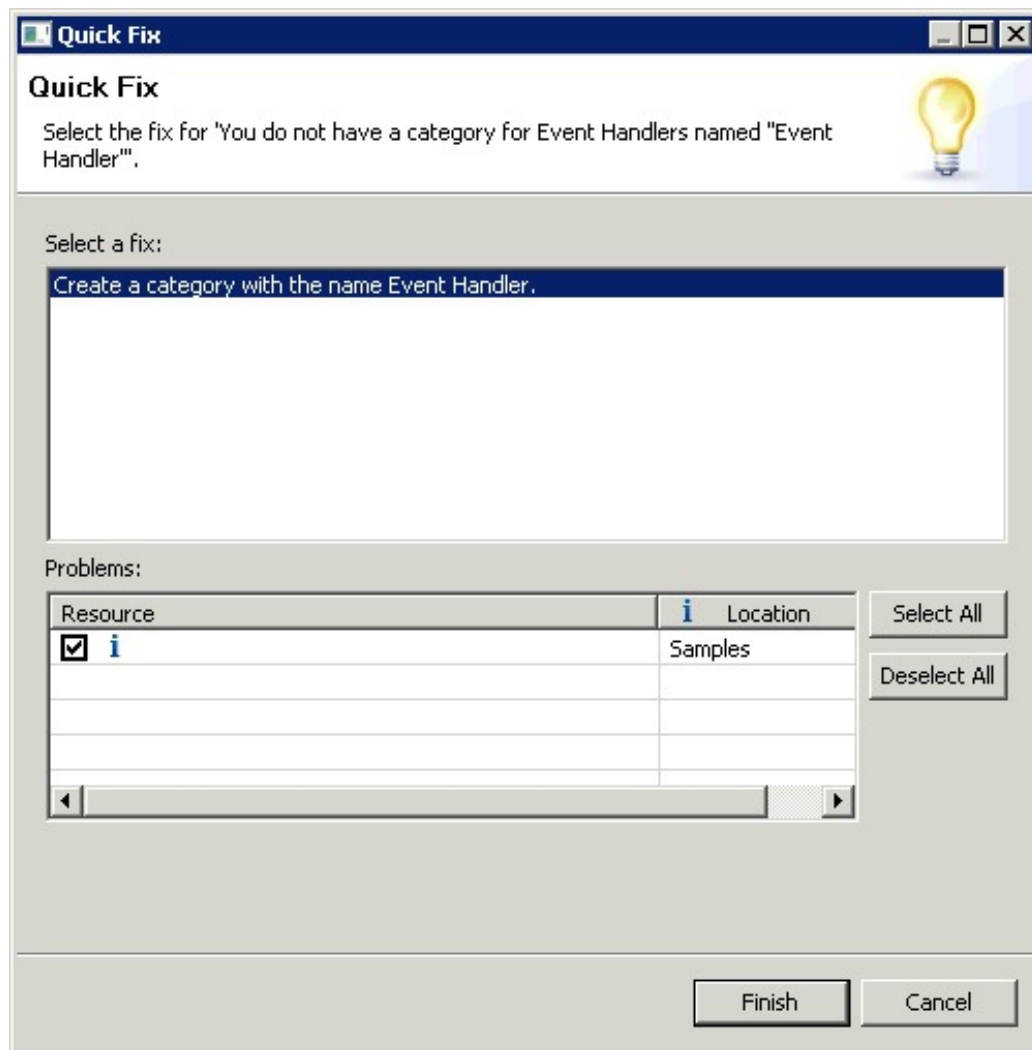


Figure 3.37: Quick Fix

3.26 Working with Metrics in Jubula

The Analysis framework in Jubula lets you run metrics on your Project. In the current version, three example metrics are included which can be run on the whole Project or on parts of it.

The following metrics and analyses are available:

3.26.0.1 Numeric Project Element Counter

This metric is in the category *Numerical Metrics*.

This metric counts the amount of items in a Project according to the following rules:

Test Cases: How many individual Test Cases have been created.

Test Suites: How many individual Test Suites have been created.

Test Jobs: How many individual Test Jobs have been created.

Test Steps: How many individual Test Steps have been created, plus how many Test Steps have been used from any reused Projects. Test Steps are only counted once for the place they are specified (or, in the case of Test Steps from reused Projects, reused). They are not counted transitively (i.e. if a Project contains a Test Case with one Test Step, and this Test Case is reused four times, there is still only one Test Step).

Referenced Test Cases: How many separate (not transitive) Test Case references there are in the current scope. If a Test Case *TC1* is reused twice in a Test Case *TC2*, then the number of reuses for *TC1* is two, regardless of how many times *TC2* is reused.

Event Handlers: How many Event Handlers are used in the current scope. The same rules for non-transitivity apply - if *TC1* contains an Event Handler, then this Event Handler is only counted once, regardless how many times *TC1* is reused.

Categories: How many categories are created in the current scope. Only categories in the Test Case Browser and Test Suite Browser are currently counted.

The results are shown in the Search Result View.

3.26.0.2 Ratio general : specific

This metric is in the category *Numerical Metrics*.

This metric calculates the amount of Test Cases or Test Steps used that are generally available for all *toolkits* (the "concrete" actions). It then counts the amount of actions used that are specific to one or more particular toolkits (e.g. specific to the RCP toolkit or the HTML toolkit). The results are presented as percentages in the Search Result View.

Wherever possible, it is preferable to work with more general actions than specific. This metric may help you to gauge how well you are keeping to this guideline.

3.26.0.3 Empty chains analysis

This analysis is in the category *Analysis*.

This analysis examines the current scope for test hierarchies wherein a Test Case is reused in one place – alone in another Test Case. This Test Case is then also reused in one place – again alone which is then reused in one place alone in another Test Case and so on. Such hierarchies can often be redundant: the Test Case at the bottom of the hierarchy could just as well have been reused directly in the Test Case at the top of the hierarchy.

Such empty chains are collected during the analysis (which may take longer on larger Projects) and displayed in the Search Result View. They are shown in order of their length, starting with the longest chains.

The analyses and metrics can either be run for the whole Project, or for specific contexts within the Project.

To run a metric or analysis for the whole Project, select the metric or analysis you want to perform from the toolbar button.

To run a metric or analysis for a specific context, select the context you require (e.g. the top node in the Test Case Browser to perform the metric or analysis just for items within this browser) and then select:

Analyze

from the context menu, selecting the required option to run.

Once a metric or analysis has run, the results are shown in the Search Result View.

3.27 Adapting the user interface

You can individualize the Jubula user interface in various ways.

3.27.1 Moving Browsers, Views and Editors

You can move items in the user interface in two ways:

- Drag-and-drop views or browsers. While the mouse button is held, the target location is marked by a gray rectangle.
- Right mouse-click on the tab area and select *"Move"* and then either *"View"* or *"Tab Group"*. Single-click to drop the item.

3.27.2 Resizing in the user interface

To change the size of the views and browsers:

- Double-click on the tab of a view, browser or editor to maximize it. Double-click again to minimize it.
- Use the buttons in the top right-hand corner of the view or browser to minimize, maximize or restore it.
- Drag the borders of the view or browser to enlarge or reduce it.
- Right mouse-click on the name of the view or browser and select *"Size"* and then the side to be adjusted. The chosen side appears as a blue line which can be dragged and dropped.
- To turn the view or browser into a separate window, select *"Detached"* from the context-sensitive menu of the tab area.

3.27.3 Restoring user interface defaults

1. You can restore the default perspective at any time:

`Window` → `Reset Perspective`.

2. To show or restore individual views or browsers, choose either the *"restore"* icon in the tab for the view or select:

`Window` → `Show View`

and then choose which view to display.

3.27.4 Changing perspectives

1. To change between perspectives, select:
`Window` → `Open Perspective`.
2. Select which perspective to open.
3. Alternatively, you can use the icons in the top left-hand corner of the current perspective to toggle between perspectives.

3.27.4.1 Automatically changing perspective

By selecting:

`Window` → `Preferences`,

you can choose to automatically change to the Functional Test Execution Perspective when test execution starts, or to be asked each time a Test Suite is started.

3.28 Searching in Jubula

3.28.1 Searching for and opening the original specification of a Test Case or Test Suite

To find the the original specification of any reused Test Case or Test Suite:

1. Single-click the Test Case or Test Suite whose specification you want to find.
2. Right-click on the Test Case or Test Suite and select "*Show specification*" from the context-sensitive menu. You can also press »F6« to show the specification.
3. The originally specified Test Case or Test Suite will be highlighted in the Test Case Browser or Test Suite Browser.
4. You can open the specified Test Case or Test Suite automatically by selecting "*Open specification*" or pressing »CTRL+F6«.

Test Cases and Test Suites that are not visible due to active filters in the browser are not displayed when show specification is used. You can, however, use *open specification* to directly open a filtered Test Case or Test Suite.



3.28.2 Searching for places where a Test Case has been used

To find the places where you have reused a Test Case:

1. In a browser or editor, right-click on the Test Case whose places of reuse you want to find.
2. Right-click on the Test Case and select "*Show where used*" from the context-sensitive menu. You can also press »F7« to show the places where a Test Case has been reused.
3. In the "*Search Result View*", a list of places will appear. These are the places where the Test Case has been reused.
4. Double-click on a Test Case icon in the view to highlight the place where the Test Case is reused and to open the editor for this Test Case.

3.28.3 Searching for places where a component name has been used

You can find the places where you have used a component name from within the Object Mapping Editor and the Component Name Browser.

In the Component Name Browser:

1. Select the component name you want to search for and select:

`Show where used`

from the context-sensitive menu. You can also press »F7« to show the places where the name has been used.

2. All the places where you have used this component name in this Project will be shown in the Search Result View (→ page 181) . Both Test Cases and object mappings where the component name was used will be shown.

In the Object Mapping Editor: Show where used:

1. Select the component name you want to search for and select:

`Show where used`

from the context-sensitive menu. You can also press »F7« to show the places where the name has been used.

2. All the places where you have used this component name in this Project will be shown in the Search Result View (→ page 181) .

In the Object Mapping Editor: Show corresponding specification:

Use this option when a component name appears in the Object Mapping Editor that should have been overwritten, but hasn't been.

1. Select the component name you want to search for and select:

`Show corresponding specification`

from the context-sensitive menu.

2. The Search Result View only shows places which could have lead to this component name appearing in the Object Mapping Editor, not all places in the test where it has been used.

3.28.4 Searching for places where a central test data set has been used

You can find the places where you have used a central test data set from within the Central Test Data Editor:

1. Select the central data set you want to search for and select:

Show where used

from the context-sensitive menu. You can also press »F7« to show the places where the data set has been used.

2. All the places where you have used this data set in this Project will be shown in the "Search Result View" (→ page 181) .

3.28.5 Using the search dialog

You can open the search dialog from the toolbar, by pressing »CTRL+H« or via the menu:

Search → **Search**

The search dialog allows you to search the **current Project in the current working language** for:

- Keywords (→ page 179) (Test Cases, Test Steps, Test Suites, Test Jobs and categories).
- Test data used in Test Cases or central test data sets (→ page 180) .
- Files in the workspace (→ page 181) .

3.28.5.1 Searching for keywords (Test Cases, Test Steps, Test Suites, Test Jobs and categories) throughout the Project

On the first tab of the search dialog, you can search for the following based on their name:

- Test Steps
- Test Cases – either the original specification or places where a Test Case has been reused (referenced)

- Test Suites – either the original specification or places where a Test Suite has been reused (referenced)
- Test Jobs
- Categories

Enter the term you wish to search for and select whether you want the search term to be case sensitive or whether you will be using regular expressions.

Click "*Search*" to start the search. The results are shown in the Search Result View (→ page 181) .



Only names from within the current Project and in the current working language are displayed. Names from reused Projects are not found.

3.28.5.2 Searching for test data

On the second tab of the search dialog, you can search for test data in the following:

- Central test data sets
- Test Cases – either either the original specification or places where a Test Case has been reused (referenced)
- Test Steps



The search does not consider the *name* of the central test data set, for example, only the data contained in it.

Enter the term you wish to search for and select whether you want the search term to be case sensitive or whether you will be using regular expressions.

Click "*Search*" to start the search. The results are shown in the Search Result View (→ page 181) .



Only test data from within the current Project and in the current working language is displayed. Data from reused Projects is not found.

3.28.5.3 Searching for files in the workspace

On the third tab of the search dialog, you can search for and in files contained in your workspace.

Enter the term you wish to search for and click "Search" to start the search. The results are shown in the Search Result View (→ page 181) .

3.28.6 Using the search result view

The Search Result View shows any items found during a search – either via the search dialog or the "Show where used" action.

From the Search Result View, you can:

- Double-click on an item to open its editor.
- Re-run the search using the button in the top right-hand corner of the view.
- Show a list of previous searches, whose results can be reshown when they are selected.

3.28.7 Searching for items in editors and browsers

Jubula offers a search function in most editors and views.

1. Start the search function either via:

Edit → **Find**

or using the context-sensitive menu in the editors and browsers.

2. In the find dialog, you can enter the following criteria:

- The text you are searching for.
- The direction you want to search in.
- Whether the search is case sensitive or not.
- Whether you are using regular expressions (→ *Reference Manual* p. 15).

Regular expression searches are case-sensitive.



- Whether the search should be wrapped. This will carry on the search from the top of the browser once you reach the bottom of the browser.
3. Items which correspond to the search terms are highlighted in gray.

3.28.8 Using filters in Jubula

3.28.8.1 Text filters

The Test Case Browser, the Test Suite Browser, the Component Name Browser and the Object Mapping Editor (in the tree view) all contain a filter area at the top. You can enter a filter text into this area to show matches to the text in the browser or editor. Use star (*) as a wildcard.

In the Test Case Browser, the Component Name Browser and the Test Suite Browser, the names of the items in the browser are considered for the filter.

In the Object Mapping Editor, the names and *types* of the components are considered for the filter.

3.28.9 Other filter options

Some browsers offer pre-defined filters in the top right-hand corner.

In the Component Name Browser, you have the following options:

- Sort by name or by type
- Hide component names from reused Projects.

In the Test Case Browser, you can also show any unused Test Cases.



We recommend using this filter to refactor or clear up your test.

3.29 Troubleshooting

3.29.1 General help

Always check in the Problem View first for any help with your tests. The messages there should help you with most of the problems. Jubula also highlights problems in the browsers and views with red crosses. Hover the mouse over the item to see a tool tip with more information.

In some text fields you can only enter certain characters. A red background in a text field means that you have entered something incorrect in that field. A yellow background means that the information entered is incomplete.

Looking in the log files may help to resolve some problems. Logs for the Jubula client and AUT Agent can be found via:

Help → Show Client/AUT Agent Log

The ITE can only access an AUT Agent log file if it is currently connected to a AUT Agent.



This section contains information on some of the more common problems:

- Starting the AUT Agent (→ page 183) .
- Connecting to the AUT Agent (→ page 184) .
- Starting the AUT (→ page 184) .
- Mapping components from the AUT (→ page 185) .
- Executing Test Suites (→ page 186) .
- Failed Test Suites (→ page 187) .
- Locked Test Cases (→ page 188) .

For other problems, look in the relevant section of the user manual or the reference manual. There are also FAQs on the Jubula website.

3.29.2 I can't start the AUT Agent

Check:

- That you have used the right port number and that it is not being used by any other program.
- Whether the computer on which the AUT Agent is installed is switched on.
- That there are no network problems, and that all necessary hardware is connected.

For information on starting the AUT Agent, refer to the section earlier in this chapter (→ page 21) .

3.29.3 I can't connect to the AUT Agent

Check:

- That the AUT Agent you want to connect to has been started and is running (→ page 22) .



If you have not started an AUT Agent, you can work with the embedded AUT Agent (→ page 23) .

- That you are connecting the AUT Agent to the right port number and AUT Agent hostname.
- That you are connecting to the right version of the AUT Agent.

For more information on connecting to the AUT Agent, see the section earlier in this chapter (→ page 23) .

3.29.4 I can't start the AUT

If the start AUT button is disabled

If you cannot start a particular AUT, this could be because:

- You have not connected to the AUT Agent (→ page 23) .
- You have not defined an AUT (→ page 45) .
- You have not configured an AUT (→ page 48) .
- The AUT does not support the current working language.

Errors starting the AUT

If there is an error starting the AUT, this could be because:

The AUT cannot be found: Make sure that the JAR or executable file for the AUT is correct, or that the classpaths are correct (→ page 48) .

The main class has not been found: Make sure that the JAR file or the classpaths you have entered contain the main class (→ page 48) .

The JAR given as a classpath is not valid: Check that you have entered the right JAR file, and that the path to it is correct from this computer. If you have entered a relative path, make sure that it is relative to the AUT working directory, if there is one, or to the AUT Agent directory if you have not specified a AUT working directory.

The JAR given as a classpath does not contain a distinct main class:

Check that you have entered a JAR which contains a main class.

If the AUT does not appear in the Running AUT's View

If the AUT does not appear in the Running AUT's View and you are testing an RCP application, make sure that the RCP Remote Control plugin has been installed (→ page 210) .

Otherwise, check that the toolkit you specified in the AUT configuration (→ page 48) is the right toolkit for this AUT.

3.29.5 I can't map components in the Object Mapping Mode

Green border does not appear in Java AUT's

If you are in the Object Mapping Mode and cannot see a green border around components, then this could mean the following:

- The border cannot be drawn – try collecting the component anyway and see if the technical name appears in the Object Mapping Editor.
- The component is not supported – find out whether the component in your AUT is a standard component or a custom component. If it is a custom component, you may need to extend Jubula to be able to test it.

Green border is shown but component cannot be mapped

If you can see the green border in the Object Mapping Mode

but cannot collect the component, this could mean the following:

- The key combination you are using is being blocked by the AUT – try changing the combination (→ page 152) .
- The component is already mapped. Check whether a technical name in the *unassigned technical names* or *assigned names* category is being highlighted.
- The component is not supported – find out whether the component in your AUT is a standard component or a custom component. If it is a custom component, you may need to extend Jubula to be able to test it.

Problems with GTK

The GTK graphics toolkit (often used with Linux) can make object mapping with the default Jubula settings impossible for certain components. If the AUT that you are mapping is running under Linux, please change the object mapping shortcut key combination to anything *other than* »CTRL+SHIFT+[0-9 OR A-F]«.

3.29.6 I can't execute my Test Suite

If you cannot start a Test Suite, check that:

- The Test Suite has complete data for the current language.



No parameter values can be empty in a Test Suite. If you no longer need the parameter, delete it.

- The object mapping is complete (→ page 122) .
- The Object Mapping Mode and the observation mode have been switched off.
- You have connected to the AUT Agent (→ page 23) .
- You have started the AUT (→ page 136) .



You can use the Test Suite Browser to click through the Test Suite to see the exact places where data and object maps are missing.

3.29.7 My Test Suite failed

If your test fails, you can find exact details about the problem by looking at the Test Result View (→ page 142) . A screenshot of the AUT is automatically taken when an error occurs and can be seen in the Image View.

For other problems with test execution, check:

- That the AUT is in focus and is visible. You can automatically activate the AUT in the AUT properties (→ page 48) or you can use a Test Case to activate the AUT. You can also set your preferences to minimize Jubula when test execution begins (→ page 149) .

Bear in mind that different platforms have different focus behavior. You may need to specifically bring the AUT into focus by clicking into it as a part of the test.



- That the AUT is ready when the test begins. To ensure this, you can insert a Test Case at the beginning of each test which waits for the window or for the first component you want to test. This is especially important when you are running tests from the command line with the Test Executor.

Waiting for a window to appear is generally a good idea whenever a new window or dialog is opened during the test. Waiting for the window or the component you want to test will make sure that your tests run even under different timing conditions.



- Make sure that the Test Cases and Test Steps are in the right order.
- Ensure that the data you have entered is correct.
- Check your object mapping configuration (→ page 126) . We recommend that you set the recognition to "standard".
- If the AUT is an RCP application, make sure that the keyboard layout is correct (→ page 48) . Otherwise, make sure you have the current version of the RCP Remote Control plugin (start your application with -clean to be sure).

- Check that the Test Steps preceding the failed Test Step were successful by looking at the state of the application.



To help you analyze the reasons for Test Suite failure, you can use the *pause on error* option (→ page 139) .

3.29.8 My Test Suite failed when using *rdesktop*

- Under a very specific configuration involving *rdesktop 1.5*, Jubula is known to frequently fail tests with an action error: 'Timeout received before confirming the posted event'.
- This occurs when *rdesktop 1.5* is used to log into a user account used for the ITE as well as a separate account for the AUT Agent.
- The problem does not occur under *rdesktop 1.4.1*.

3.29.9 I can't save my editor

If you are working in a multi-user environment, or if you are working on multiple editors at once, Jubula may lock the editors. This is to secure the editors so that conflicting changes do not take place. You will be able to save changes in your editor once the changes by the other tester or in the other editor have been saved.

After an irregular shutdown, you may find that some editors may still be locked. This is temporary and will pass.

3.29.10 Creating a support information package

If you have found a problem in Jubula, you can create a support information package from within the ITE. You can choose which information to include in the package, which can be uploaded to a bug-tracking system as a .zip file.

1. Select:

Help → Create Support Information Package

2. In the dialog which appears, select what to add to the package:

Configuration information: This includes information about how you have configured Jubula, which other plugins you are using, and information about your environment.

The AUT Agent log This lets us see the messages from the AUT Agent. If you are not connected to the AUT Agent, this option will be grayed out.

The client log This lets us see the messages from the ITE.

The Project and depending Projects The currently open Project and any Projects it reuses.

3. Browse to the destination to save the .zip file.
4. Click "OK".
5. The prepared package will be saved to the destination you specified.

3.29.11 Log file locations

The logs for Jubula can be found in your home directory under:

.jubula/logs

3.30 Finishing up

3.30.1 Stopping the AUT

1. When you stop the AUT Agent, any AUT's that were connected to this AUT Agent are closed.
2. AUT's which were started from a configuration are automatically closed when the test execution via the test executor stops.
3. To stop the AUT manually, select the AUT you want to stop from the Running AUT's View and then click the Stop AUT button.
4. A dialog will appear to ask you if you are sure you want to stop the AUT. Click "yes".
5. You can opt to not see this dialog in the preferences (→ page 149) .

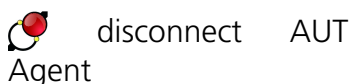


Stopping an AUT in a test

Stopping an AUT as part of a test is generally not possible. The AUT will be stopped automatically at the end of testing according to the points mentioned above.

You can use the *restart* action to restart the AUT during the test. You may be able to use the action *External Key Combination* to send an unconfirmed keystroke to the AUT (i.e. »ENTER« on a prompt dialog for closing the AUT) to close the AUT at the end of your test, however, the success of this is dependent on the AUT and environment as well as timing and synchronization issues.

3.30.2 Disconnecting from the AUT Agent



To disconnect from the AUT Agent, select the "*Disconnect from AUT Agent*" button on the toolbar.

3.30.3 Closing Jubula and stopping the AUT Agent

Once the Test Suite, AUT and AUT Agent have been stopped or disconnected, make sure any changes are saved and select:

File → **Exit** .

3.30.4 Stopping the AUT Agent

- Stop the AUT Agent by selecting:
`Start` → `Programs` → `Jubula`
and then "*Stop AUT Agent*" from the Windows Menu.
- Under Linux, use the script:
`stopautagent`.

You can also stop the AUT Agent from the system tray.



- You can use the parameter `-p <port number>` to specify which port number should be stopped.
- You can also enter the hostname directly after the `stopautagent` command to specify which hostname should be stopped.

Stopping the AUT Agent will stop any AUT's that are connected to this AUT Agent.



3.31 Using the test executor for testing from the command line

You can also run tests using the test executor. From the command line interface, you can select and start tests.



It is especially important to ensure that your test contains a suitable startup module to wait for the AUT, perform the login if necessary and reset the AUT to a defined state when you are running tests from the command line.

3.31.1 Starting the test executor

1. In a command line interface, locate the Jubula installation directory and open the Jubula directory within it.
2. Enter `testexec.exe`.
This will run the executable `testexec.exe`.
3. Do not press »ENTER« until you have entered the necessary parameters. See the next section for details on the parameters.



Be sure to start the AUT Agent before executing a test

3.31.2 Parameters for the test executor

1. Once you have browsed to the Jubula installation directory and entered `testexec.exe`, you can enter the parameters for the test execution.
2. The test executor has various parameters Figure 2 →
page 192 :



A list of language codes is available in the reference manual ((→*Reference Manual* p. 383)).

Detail	Parameter
Help	-h Gives parameter help
Project name	-project <project name> e.g. -project "ExampleProject"
Project version	-version <project version> e.g. -version "1.3"
AUT configuration name	-autconfig <configuration name> e.g. -autconfig "localconfiguration" Use when starting an AUT via a configuration.
AUT ID	-autid <ID> e.g. -autid "SimpleAdder1" Use when AUT was started via <i>autrun</i> .
Configuration file	-c <path to configuration file> e.g. -c "C:/Program Files/guidancer/config" Use this instead of entering all arguments via the command line.
Workspace	-data <path to workspace> e.g. -data "C:/Users/Test" The Jubula workspace with the preference settings for the database connection
Database scheme	-dbscheme <scheme> e.g. -dbscheme "embedded"
Database username	-dbuser <username> e.g. -dbuser "username"
Database password	-dbpw <password> e.g. -dbpw "password"
Server	-server <AUT Agent hostname> e.g. -server "localhost"
Port number	-port <port number> e.g. -port "60000"
Language	-language <language code> e.g. -language "en_US"
Test Suite	-testsuite <testsuite name> e.g. -testsuite "suite1" Use to start a Test Suite Only one Test Suite or Test Job can be started
Test Job	-testjob <testjob name> e.g. -testjob "job1" Use to start a Test Job

Detail	Parameter
	Only one Test Suite or Test Job can be started
Data directory	-datadir <path to external test data directory> e.g. -datadir "C:/Program Files/Test/Data"
Result directory	-resultdir <path to directory> e.g. -resultdir "C:/Program Files/Test/results"
No run option (optional)	-n e.g. -n Checks if the Test Suite is startable.
Quiet option (optional)	-q e.g. -q Does not give out test information.
Database URL (optional)	-dburl <URL> e.g. -dburl "db.example.de" If no URL is given, the default will be used.
Timeout (optional)	-timeout <timeout in seconds> e.g. -timeout "3600" Enter an optional timeout for the command line client.
No screenshot (optional)	-s
Test results not relevant (optional)	-r Flags the test results as not relevant in the test result summary (→ page 153) .

- You can either enter a Test Suite to be executed or a Test Job. Only one of these two commands is accepted for the test executor.



If you are starting a Test Suite, Jubula will start your AUT from its configuration if you have entered one. If you are starting a Test Job, you must make sure that the first AUT you require is already started with the *autrun* command. Other AUT's required during the test must either also be started already, or started as a part of the test itself.

4. If the AUT you want to test was started with the *autrun* command, you must enter an AUT ID. If your AUT will be started with an AUT configuration, then enter the configuration name.

The test executor also accepts arguments to pass on to the Java Virtual Machine. This means that you can, for example, increase the initial and maximum amount of system memory allocated to the JVM with the parameters `-Xms<memory_size>` and `-Xmx<memory_size>`, respectively. For example, the parameter `-Xmx128M` would make a maximum of 128 MB of system memory available to the test executor. When entering the standard parameters for the test executor, you may add `-J<JVM_parameter>` for each JVM parameter you wish to set. For example, `-J-Xmx128M`. Multiple parameters, like standard parameters, are separated by spaces. Here is an example of defining multiple JVM parameters: `-J-Xmx128M -JXms128M`.



5. Using the no run option will check the completeness of the Project, test data and the validity of the database connection. The AUT Agent connection is not checked.
6. Once you have entered all the necessary parameters, press »ENTER«.
7. The client will connect to the AUT Agent, connect to the database, open the Project, start the AUT (for Test Suites) or connect to it (for Test Jobs) and then execute the Test Suite or Test Job you specified.
8. Once the test has finished, the client will show an exit code.
 - "Exit code: 0" indicates that the test was successful.
 - "Exit code: 1" indicates that the test contained an error.

To stop the test execution, use »CTRL+C«



3.31.3 Using the test executor with the embedded database

If you are using the default embedded database, you will need to enter the following information as parameters:

-dbscheme Default Embedded (H2)

-dbuser sa

-dbpw <empty> ("")

If you have changed the name of the default database scheme in the database preferences (→ page 151) , you should alter the parameter for the *-dbscheme* accordingly.

3.31.4 Using a configuration file

1. You can also enter test executor parameters in an XML configuration file.



The workspace parameter cannot currently be entered in the configuration file for the test executor, it must be entered directly into the command line interface.

2. This saves you typing in the same details each time you start a test.
3. To enter the configuration file path, enter the following command:
`-c <path to file>`
For example: `-c "C:/My Documents/config1.xml"`
4. You can also overwrite parameters contained in the configuration file by entering another parameter directly into the interface.
5. For example, if your configuration file contains the Test Suite "*Suite1*", but you enter `-testsuite "Suite2"` in the command line interface, the Test Suite called "*Suite2*" will be started, not "*Suite1*".
6. To create an XML file:
A Open a text editor of your choice.

B Enter the desired parameters as shown in the example:

```
<configuration>
  <project>ExampleProject</project>
  <version>1.0</version>
  <autconfig>local configuration</autconfig>
    OR
  <autid>Adder</autid>
  <dbscheme>embedded</dbscheme>
  <dbuser>sa</dbuser>
  <dbpw></dbpw>
  <server>localhost</server>
  <port>60000</port>
  <language>EN_US</language>
  <testsuite>Suite1</testsuite>
    OR
  <testjob>Job1</testjob>
  <datadir>C:/Program Files/guidancer/data</datadir>
  <resultdir>C:/Program Files/guidancer/results</resultdir>
</configuration>
```

C Save the file as an XML file.

D This file will be used when you enter the "-c <path to file>" parameter.

The no-run and quiet mode parameters can only be entered in the command line interface. They cannot be entered in the configuration file.



Java Virtual Machine parameters can only be entered in the command line interface. They cannot be entered in the configuration file.



7. A template configuration file was installed with Jubula. It can be found under:
InstallationDirectory/examples/scripts

3.32 Using the dbtool client to import, delete and export from the command line

You can use the dbtool to import, export and delete Projects from the database in without using the ITE.

This is particularly useful for automated build and test processes, where you may want to automatically export and import Projects out of and into version control to run your tests.

3.32.1 Starting the dbtool

1. In a command line interface, locate the Jubula installation directory and open the Jubula directory within it.
2. Enter `dbtool.exe`.
This will run the executable *dbtool.exe*.
3. Do not press »ENTER« until you have entered the necessary parameters. See the next section for details on the parameters.

3.32.2 Parameters for the dbtool

1. Once you have browsed to the Jubula installation directory and entered `dbtool`, you can enter the parameters for the database actions.
2. The dbtool uses the parameters described in the table Figure 2
→ page 199
:
3. You can use the parameter `-keepsummary` to specify that the test result summaries should not be deleted when the Project or Projects are deleted. This is useful for continuous integration processes, where the test results over time should be kept, but the Projects are reimported into the database (for example from the version control system) each night. If you do not enter this parameter, the summaries will be deleted with the Projects.



If you are using the embedded database, see the section on using the embedded database with the Test Executor for information on which username and password to use (→ page 196)

4. Once you have entered all the necessary parameters, press »ENTER«.

Detail	Parameter
Help	-h Gives parameter help
Delete Project	-delete <project-name project-version> e.g. -delete "ExampleProject" 1.0
Delete All	-deleteall e.g. -deleteall
Keep test result summaries	-keepsummary (optional) e.g. -keepsummary
Directory	-directory <directory path> e.g. -directory "D:/Test/Projects/" The directory for imports and/or exports The directory must already exist
Export Project	-export <project-name project-version> e.g. -export "ExampleProject" "1.0" Existing files with the same name will be overwritten
Export All	-exportall e.g. -exportall The directory for the export all must be empty The directory must already exist
Import Project	-import <import-file> e.g. -import <ExampleProject.xml>
Workspace	-data <path to workspace> e.g. -data "C:/Users/Test" The Jubula workspace with the preference settings for the database connection
Database scheme	-dbscheme <scheme> e.g. -dbscheme "Oracle"
Database username	-dbuser <username> e.g. -dbuser "myusername" Use sa (without quotes) for the embedded database.
Database password	-dbpw <password> e.g. -dbpw "mypassword" Use <empty> ("") for the embedded database
Database URL (optional)	-dburl <URL> e.g. -dburl "db.example.de" If no URL is given, the default will be used.

Table 3.2: Parameters for the dbtool

3.33 Using Chronon in Jubula

Jubula supports Chronon (2.0) as a monitoring agent to record information about the use of Jubula itself as well the behavior of your AUT during a test. The information gathered by Chronon can be used to help analyze any errors found in the program.



Jubula users: the Chronon plugins are only available in the standalone version.

3.33.1 Using Chronon in Jubula

The embedded support for Chronon in Jubula lets you record debug information while you are working with Jubula. Should an error occur, the Chronon report can be added to any bug reports or support conversations to help our team analyse the problem.

3.33.1.1 Turning on the recording

1. First, you must make the Chronon embedded recorder available for Jubula. To do this, you must open the Jubula.ini file and remove the comment symbol from the last two lines. Then you must (re)start Jubula.
2. You can activate the Chronon recording in Jubula by selecting:
`Help → Start Chronon Recorder`
3. The Progress View will open and you will see the status of the recording. The progress bar continues to run once the recorder has started. The status changes from *starting* to *running*. When the recorder is running, you will be able to see the job running in the Progress View and also in the status bar. Do not stop the job – this will stop the recording.
4. Once the recorder is running, your actions in Jubula are written to the recording file.

Bear in mind that any data contained within Jubula may be visible by our development team when analysing any Chronon files



We do not recommend running the Chronon recorder constantly, as the collection of additional information can lead to detractions in performance in Jubula. Instead, we would recommend turning Chronon on when you specifically want to reproduce a problem, and turning it off afterwards. Additionally, you should ensure that you have increased the heap space to avoid running into memory problems.



3.33.1.2 Turning off the recording

You can deactivate the Chronon recording by opening the Progress View and clicking the "Stop" button.

Restarting or closing Jubula will also stop the recording.

3.33.1.3 Accessing the Chronon report

The Chronon reports are saved to your home directory under:
.jubula/Chronon

When working with Chronon in Jubula itself, a new folder is created for each recording session (one session = starting - recording - stopping). The relevant folder for a bug report can be zipped and attached to help our analysis.

3.33.1.4 Increasing the heap space to improve working with Chronon

Having the Chronon recorder running is very memory-intensive. You will almost certainly have to increase the heap space allocated to Jubula. You can do this in the .ini file for Jubula.

3.33.2 Using Chronon when testing your AUT

You can configure your AUT to run with Chronon recording so that you can analyze any errors that occur during your automated test runs. See the section later (→ page 204) on analysing the reports for information on the tooling required to do this.



Jubula users: The support for Chronon recording in AUT's is currently only available in the standalone software.

3.33.2.1 Adding Chronon information in the AUT configuration

You can select Chronon as a monitoring agent and configure it in the Expert Settings in the AUT configuration.

1. Open the AUT configuration dialog from the Project properties (→ page 33) .
2. Select the *Expert* configuration.
3. Select Chronon as the monitoring agent.
4. You can then enter the configuration details for the monitoring.

Package patterns: Enter a comma-separated list of packages that you want to be covered by the monitoring. The packages must adhere to the patterns as defined in the Chronon documentation, for example `com.myorg.**` selects the whole `com.myorg` namespace. `com.myorg.*` selects only classes in the `com.myorg` package. The documentation for the patterns is located at

<https://chronon.onconfluence.com/display/DOC/Include+and+Exclude+patterns>.

If you enter no patterns, the recording file will be empty.

Target directory: Enter the path to a directory where the recording files should be written. The AUT must have write permissions to this place. Any non-existing folders will be automatically created. This target directory and all items it contains are deleted when the AUT is started or restarted. Once the AUT is started, the target

directory is recreated ready for the next run. This means you must ensure you have copied any relevant files before starting or restarting the AUT. When the AUT is stopped after a test, the recording file is written to this target directory. The target directory can be relative to the AUT working directory, if you have entered one.

Operating System: Enter `win`, `lin`, `sol` or `mac` (without quotes) to specify your operating system.

JRE Architecture: Enter `32` or `64` (without quotes) to specify your architecture. If you have entered `mac` as an operating system, you can only enter `64`.

3.33.2.2 Adapting tests to improve data collection

You should bear the following in mind when using Chronon for recording information in automated tests.

Performance in the AUT may be affected

- The recordings that Chronon performs are very memory-intensive. For this reason, you may notice performance differences in your AUT. It may also be necessary to increase the step delay for your tests, and / or add extra synchronization to compensate for the performance differences when Chronon is running.
- For these reasons, we do not recommend having Chronon configured as a part of your standard AUT configuration. Instead, we suggest running tests with Chronon monitoring when needed.
- We also strongly suggest ensuring that your AUT and the machine it is running on have sufficient memory to cope with the increased monitoring activity.

Stopping or restarting the AUT will cause the previously recorded information to be lost

- The recording files are written when the AUT is stopped. This means either stopping the AUT by hand, using the "Stop AUT" button in Jubula or when you use the *restart* action.
- Because of this, we recommend executing individual Test Cases (use cases) in Test Suites that you want to analyze with Chronon. You should ensure that any Event Handlers in the Test Suite will not cause the AUT to restart.

3.33.2.3 Analyzing the generated reports

To analyze the reports generated, you will require the Chronon Time Travelling Debugger from Chronon Systems. Jubula users can download a trial version of this tool. The link to the trial version is provided in the expert AUT configuration. Open source projects may contact Chronon Systems for free licenses.

3.34 Launch Configurations

3.34.1 Intro

Jubula allows you to start AUT's from within Eclipse, similar to the way in which you would debug an application using Eclipse. This section describes how to start your AUT directly from the IDE.

3.34.2 Requirements

The following Features need to be installed in order to start Swing AUT's:


- Jubula Launch Support for Java / Swing
- Java Developer Tools (JDT)

The following Features need to be installed in order to start RCP AUT's:

- Jubula Launch Support for Eclipse RCP
- Java Developer Tools (JDT)
- Eclipse Plug-in Development Environment (PDE)

3.34.3 Customizing the Perspective

In most cases, the button to start AUT's is not included in the main toolbar. The following steps describe how to add this button to a Perspective's toolbar:

1. Select:

from the main menu.
2. In the dialog that appears, select the *Command Groups Availability* tab.
3. Ensure that the entry *Start AUT* in the section *Available command groups* is checked.
4. Close the dialog by pressing the OK button.

Once you have completed the steps listed above, the *Start AUT* button should be visible in the main toolbar of the active Perspective.



3.34.4 Starting the AUT



The *Start AUT* button can be used, just like the *Run* or *Debug* buttons, to launch an application or configure application launches. Applications started in this way are always started in debug mode. Furthermore, *Start AUT* adds additional information to the launch process, allowing the started application to be tested by Jubula.

Some of the additional information that makes the started application testable can be configured from the *Test* tab of the *Start AUT Configurations* dialog.

The **AUT ID** can be assigned or left blank. If left blank, the configuration name (ex. *New_configuration*) will be used.

3.34.5 AUT Agent

Although Jubula's automatic AUT Agent selection generally selects the correct AUT Agent, it can be useful to know what criteria go into the selection process. Jubula follows these rules when deciding with which AUT Agent the starting AUT should be registered:

1. If Jubula is currently connected to an AUT Agent, then that AUT Agent will be used.
2. If an embedded AUT Agent is running, then Jubula will connect to the embedded AUT Agent and use that for AUT registration.
3. Jubula will start an embedded AUT Agent, connect to it, and use that for AUT registration.

3.34.6 Additional information for RCP AUT's

3.34.6.1 Keyboard Layout

The Keyboard Layout for the AUT can be defined in the *Start AUT Configurations* dialog. The entered value must represent a locale and must represent a defined keyboard layout.

3.34.6.2 RCP Remote Control Plug-in

In order to start an RCP AUT, the RCP Remote Control plug-in (`org.eclipse.jubula.rc.rcp`) must be included in the launch

configuration. The list of plug-ins for the launch configuration can be viewed and modified from the *Plug-ins* tab of the *Start AUT Configurations* dialog when an *Eclipse Application* launch configuration is selected.

3.34.7 Common Pitfalls

- Breakpoints may alter the timing of tests. This can have consequences such as timeouts, tests failing when they should succeed, and tests succeeding when they should fail.

Chapter 4

Toolkit-specific information

This chapter contains information testing AUT's written in the various toolkits supported by Jubula.

4.1 Testing Swing AUT's

You can write tests for Swing AUT's when you select the toolkit *concrete* or *Swing* in the Project properties. The Swing and concrete toolkits actually contain the same actions as each other.

When you select Swing as the Project toolkit, the library Project *unbound_modules_concrete* is automatically reused in your Project. The actions in this library are described in the reference manual (→ *Reference Manual* p. 21).

4.1.1 Supported Swing AUT's

Jubula supports AUT's written with the Swing GUI toolkit according to the following points:

- The AUT is written using Java 1.4 or higher.

Swing AUT's written in Java 1.4 cannot be started using an executable file (→ page 48) and cannot be started using the *autrun* option (→ page 52) . For both of these options Java 1.5 or higher is required.



- It uses a Java SE (Standard Edition) JRE.



AUT's based on the NetBeans framework, and NetBeans itself, are currently not supported.

4.1.2 Design for testability in Swing

Although components in the AUT can be recognized even when they are not named by the developers, using the *setName* method for the current Swing component class certainly makes it easier to test AUT's. Even if a whole area of the AUT has changed, the component will still be found based on this unique name.

4.2 Testing RCP AUT's

You can write tests for RCP AUT's by selecting the toolkit *concrete* or *RCP* from the Project properties. If you will need to test RCP-specific components (such as toolbars with drop-down menus, or tree-tables), you will need to select *RCP* as the toolkit for the Project.

When you select RCP as the Project toolkit, the library Projects *unbound_modules_concrete*, *unbound_modules_swt* and *unbound_modules_rcp* are automatically reused in your Project. The actions in these libraries are described in the reference manual (→*Reference Manual* p. 21).



The GEF toolkit is a subset of the RCP toolkit and is described in a later section (→ page 213) .

4.2.1 Supported RCP AUT's

Jubula supports AUT's written with the RCP GUI toolkit according to the following points:

- The AUT is written using Java 1.5 or higher.
- It uses a Java SE (Standard Edition) JRE.
- The AUT uses SWT version 3.1 or higher.

4.2.2 Setting up an RCP AUT for testing

If you want to test a *Rich Client Platform* application, you must first unzip our "*RCP Remote Control*" plugin into your AUT. This can be done as follows:

1. Locate the Jubula installation directory.
2. Extract the content of the *rcp-support.zip* folder into the *plugins* directory for your RCP AUT.

When you install a new version of Jubula, you must repeat these steps with the new RCP remote control plugin. We recommend starting your AUT once with `-clean` to ensure that the new remote control plugin is used.



3. RCP applications generally have a *configuration/config.ini* file which contains the parameter *osgi.bundles*. This parameter may need to be modified to allow the RCP remote control plugin to load on AUT startup. The *org.eclipse.update.configurator* plugin automatically loads all plugins found in the *plugins* directory, which means that the RCP remote control plugin should start with the AUT if *org.eclipse.update.configurator@3:start* is already defined in the *osgi.bundles* parameter. Otherwise, you may need to add *org.eclipse.jubula.rc.rcp* to the end of the *osgi.bundles* parameter.

If you do not follow the above steps, the AUT Agent will not be able to communicate with your AUT!

4.2.3 Keyboard Layouts

For RCP AUT's, a keyboard layout must be entered in the AUT configuration (→ page 50) . Jubula installs German (DE) and English (US) as standard keyboard layouts.

If you require a different keyboard layout, you must create a mapping file for the new language (→*Reference Manual* p. 387) and place it in the *server/resources* directory. You also have to put this mapping file into the RCP Remote Control plug-in of your AUT:

```
<RCPAUT>/plugins/org.eclipse.jubula.rc.rcp_<Version>/resources
```

4.2.4 Design for testability in RCP

Although Jubula can relocate components in the AUT even when they are not named by the developers, naming components is nevertheless a good idea. In SWT and RCP there is no method like the Swing *setName* method to name components in the program code. However, you can improve the testability of your application by using the following method in your SWT or RCP code for the current component class: *setData(String key, String ComponentName)*. For the key, use *TEST_COMP_NAME*.

Even if you do not name components, you can choose to have Jubula generate unique names for your components in the AUT in the AUT dialog (→ page 45) .

4.2.5 Component name generation in RCP

RCP AUT's often use wizards and standard dialogs. The components in these dialogs are not often named by developers, and are in different places on Windows, Linux and Mac systems.

Jubula lets you decide if unique names should be generated for these components in your AUT, if no name has been given. You can configure this in the AUT settings (→ page 45) . We recommend selecting this option, as it makes your tests more robust to any changes and also makes platform-independent testing possible in RCP AUT's.

4.2.6 Best practices for testing RCP AUT's

Perspective layout reset

One of the features of RCP AUT's is that they generally remember the state of the AUT (position of views, which perspective was open) when the AUT is closed. In order to make tests as robust as possible, we recommend starting each use case with a module to reset the perspective to its defaults, and testing with this default perspective.

Workspace choice

RCP AUT's use a workspace to save user-specific preferences. The choice of workspace is usually offered before the AUT starts. This dialog is not currently testable by Jubula, so we recommend adding the desired workspace as an AUT argument in the AUT configuration (→ page 48) . The parameter used to specify the workspace is `-data "WORKSPACE"`.

4.3 Testing GEF AUT's

The GEF toolkit is a sub-toolkit of RCP. To be able to test AUT's that contain GEF components, you must select *RCP* as the toolkit in the Project properties.

The same points about the RCP Remote Control plugin and the keyboard files mentioned in the section on RCP AUT's (→ page 210) are also valid for testing AUT's with GEF components.

Testing GEF components is a little different to testing other components in Jubula, because we highly recommend that the software developers use our accessibility plug-in in the AUT to make the testing of GEF components more robust.

GEF testing is supported from GEF version 3.0 up to but not including 4.0. We test our GEF support on GEF version 3.7.2.



4.3.1 Testing GEF components

Like other Jubula tests, tests for GEF components are based on the component, action, parameter principle. However, it is important to note that individual figures are not addressed using component names. The only component that is mapped in GEF is the *figure canvas*. The figures on the canvas are tested by referring to their textpath.

4.3.2 Using the GEF inspector

The GEF Inspector View in the ITE allows you to collect information about the textpath to an item from within your RCP AUT. For many of the actions on GEF figures, you will need to know the textpath. Using the Jubula accessibility plug-in will mean that the textpaths will be easier to determine without the inspector (→ page 214).

To use the inspector, you must have started the AUT (→ page 136).

1. Open the GEF Inspector View if it is not already open via:

Window → Show View → Other

2. In the Inspector View, select the AUT you want to inspect in from the drop-down menu next to the *activate inspector* button.
3. In your AUT, you will see a blue marking over items you can inspect in the canvas.
4. Click on an item to inspect it.



Make sure that the AUT is in focus before inspecting the figure.

5. In the GEF Inspector View, the path to the item will be shown. If the code for your AUT uses the accessibility code from Jubula, then the textpath will be less complex and more readable.
6. You can copy the textpath from the Inspector View via the context-menu and paste it as a parameter in your Test Cases.



You will need to activate the inspector for each item you want to inspect

4.3.3 Adding GEF accessibility to your AUT

Although Jubula supports identification and testing of GEF components out-of-the-box, it is highly recommend that application developers enhance this support by implementing an accessibility plug-in for their application. Such a plug-in makes identification of GEF components more robust and allows Jubula tests to reference additional components, such as connection anchors. The following sections explain how to develop an accessibility plug-in.

4.3.3.1 Setting up your Workspace

Although you can use the IDE of your choice to develop your accessibility plug-in, this guide will use Eclipse (specifically, Eclipse 3.4.1 with PDE plug-ins).

1. Before setting up your workspace, it is important to have the ECP Remote Control plug-in available, as this contains

- libraries and interfaces that are required in order to develop your accessibility plug-in. The archive file *rcp-support.zip*, available from the Jubula installation directory, contains the RCP Remote Control plug-in. Once you have extracted the RCP Remote Control plug-in, you can continue setting up your workspace.
2. In order to cleanly separate test code from application code, we recommend using separate workspaces for application and accessibility development. Create a new Eclipse workspace for accessibility plug-in development.
 3. Once you have created your new workspace, change that workspace's Plug-in Development Target Platform to include the RCP Remote Control plug-in, as well as all plug-ins that make up your AUT. Information on configuring the Target Platform plug-ins can be found in Eclipse's on-line Help under:
Plug-in Development Environment Guide > Tools > Preferences > Target Plug-ins.
 4. Create a new Plug-in Project. The default values proposed by the New Plug-in Project wizard are acceptable, so simply enter a name for your new Project and complete the wizard without additional modifications.
 5. You now have a workspace with a properly configured Target Platform as well as a new Plug-in Project. You can now begin developing your accessibility plug-in.

4.3.3.2 Walkthrough

The first step to implementing your accessibility plug-in is to create Identifiers for each type of GEF component that you wish to make accessible for Jubula tests. An Identifier is a Java class that implements *org.eclipse.jubula.rc.rcp.gef.identifier.IEditPartIdentifier* and provides Jubula with additional and/or more precise information about a specific *org.eclipse.gef.EditPart*. The granularity of your Identifier classes will depend on the class hierarchy of the EditParts in the AUT. For example, if many of the EditParts share a common superclass, then you can write a single Identifier for that superclass that will be able to provide accessibility for all EditParts that inherit from that superclass. See (→ page 217) for a sample implementation.

The next step is to create an Adapter Factory. This extension will provide Jubula with information regarding which Identifier to use for each Edit Part.

1. Open the plugin.xml file for your accessibility plug-in and select the 'Extension' tab (Figure 4.1 → page 216).



Figure 4.1: Plug-in Editor with Extensions tab selected

2. Add an instance of the `org.eclipse.core.runtime.adapters` extension.
3. Add a 'factory' to the new extension for each type of GEF component for which you wish to provide accessibility. Each factory must provide adapters from the GEF component that implements `org.eclipse.gef.EditPart` to `org.eclipse.jubula.rc.rcp.gef.identifier.IEditPartIdentifier` (Figure 4.2 → page 217).
4. Once you have defined your Adapter Factory, you will need to implement it. Your Adapter Factory, which must implement `org.eclipse.core.runtime.IAdapaterFactory`, provides appropriate instances of your created Identifiers for a given Edit Part. See (→ page 217) for a sample implementation.
5. Once you have created your Identifiers and Adapter Factories, you can export your plug-in and copy it to the 'plugins' directory of your AUT (Figure 4.3 → page 218).

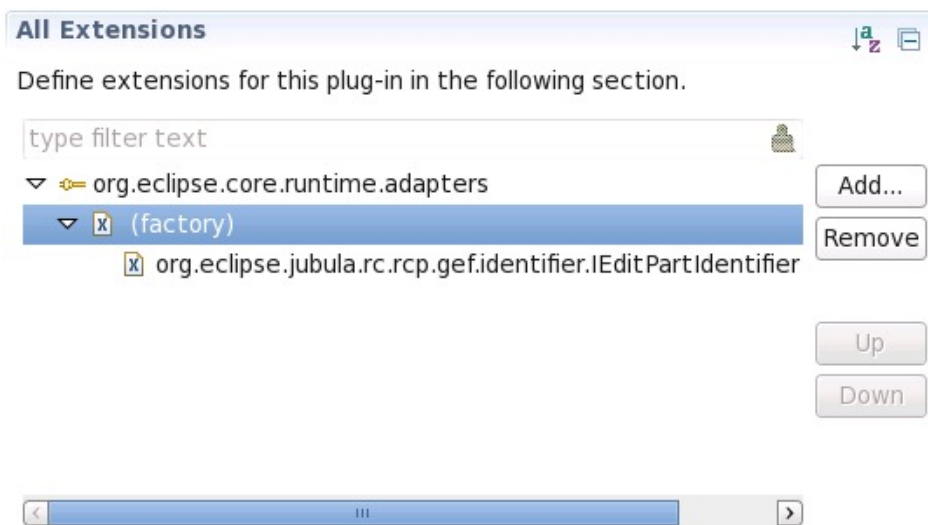


Figure 4.2: Plug-in Editor with defined Adapter Factory

When starting your AUT after adding or replacing your accessibility plug-in, it is recommended that the AUT be started with the -clean parameter.



4.3.3.3 References

A sample accessibility plug-in for certain elements of the Logic Diagram sample plug-in project (contributed from the GEF plug-ins to Eclipse's New Project Wizard) are included in the installation of Jubula. You can import the sample accessibility plug-in into an Eclipse workspace in order to examine the general structure of such an accessibility plug-in (Figure 4.4 → page 219)

The javadocs containing information about `org.eclipse.jubula.rc.rcp.gef.identifier.IEditPartIdentifier` and default implementations can be found in the Jubula installation directory under `documentation/gefapi/javadoc.zip`.

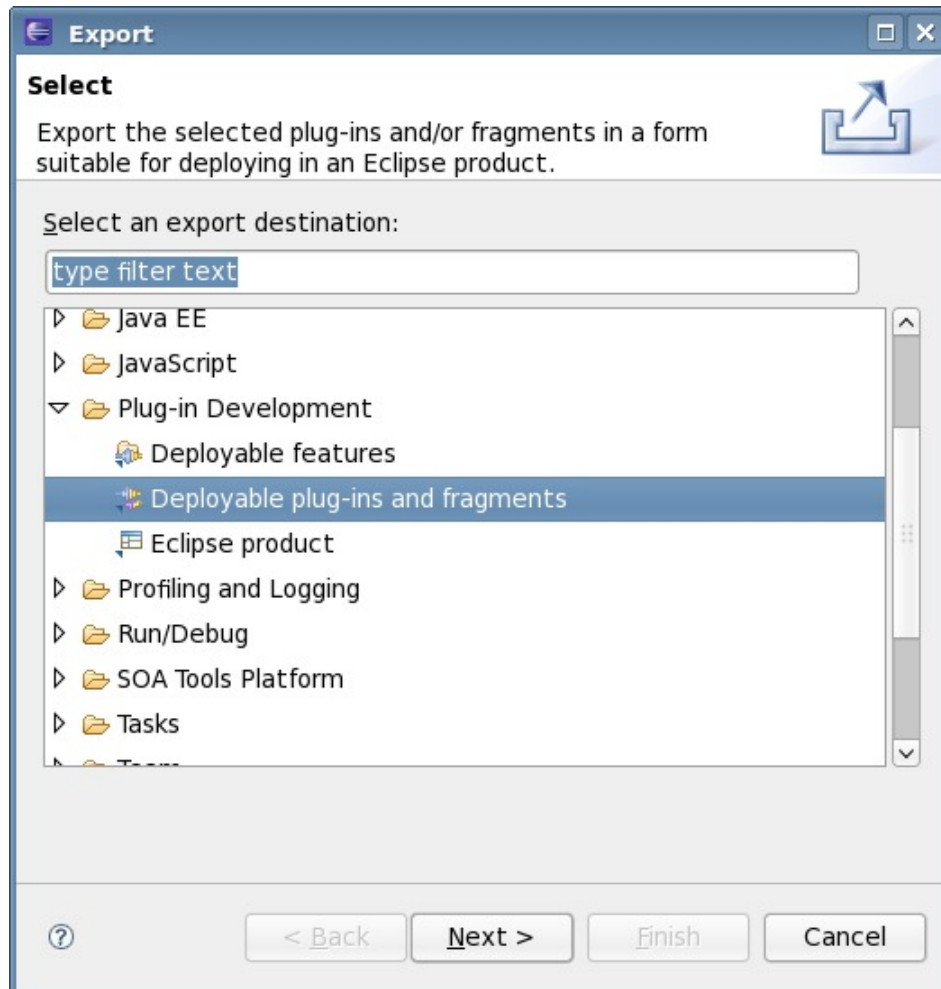


Figure 4.3: Exporting the Project to a Plug-in

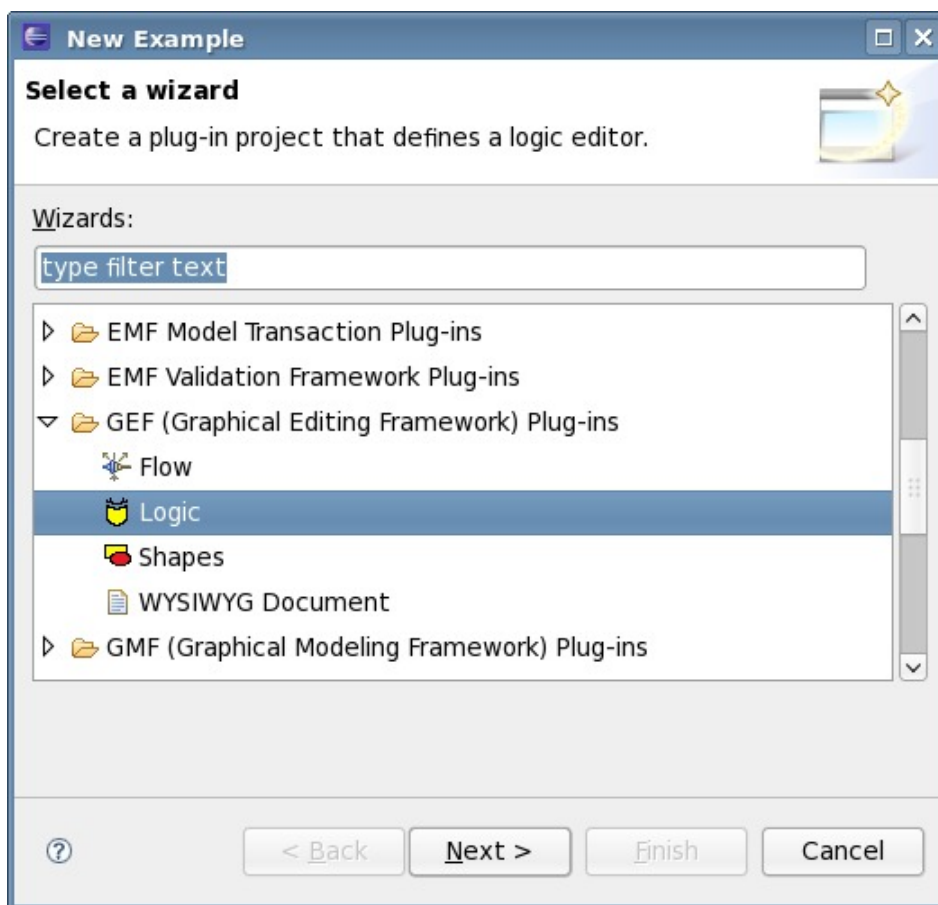


Figure 4.4: Creating the 'Logic' Project

4.4 Testing HTML AUT's

Jubula supports testing of HTML AUT's. To be able to test HTML applications, select the *HTML* toolkit in the Project properties.

When you select HTML as the Project toolkit, the library Projects *unbound_modules_concrete* and *unbound_modules_html* are automatically reused in your Project. The actions in these libraries are described in the reference manual (→ *Reference Manual* p. 21).

4.4.1 Supported HTML AUT's

Jubula supports AUT's written with the HTML GUI toolkit according to the following points:

- AUT's can be run in Internet Explorer (versions 7 and 8), Firefox (3.0 to 3.5) and Safari (version 5).
- We strongly recommend writing HTML AUT's so that they conform to the W3C standard. You can check whether your AUT is W3C conform using an online validator: <http://validator.w3.org>
- Frames and IFrames are not supported.
- Some of the Jubula actions in the *concrete* toolkit (i.e. which are theoretically valid for all AUT types) may not (yet) be supported. In some cases, this is because the component doesn't exist as such in HTML AUT's (menu bars for example). In other cases, text components such as tables or lists do not have a concept for dealing with selection as they do in e.g. Swing.
- The *autrun* option to start AUT's (→ page 52) cannot be used for HTML AUT's.
- Unlike other toolkits supported by Jubula, real clicks and key events are not sent to HTML AUT's.

4.4.2 Design for testability in HTML AUT's

Although Jubula does not require you to name components in your AUT, you nevertheless make your AUT more testable by doing so.

- Each supported component in HTML AUT's can have its own attribute used to identify it during test execution.
- In your AUT configuration (→ page 50) , you can define an attribute name which should be used as an identifier for components.
- For example if your attribute name is `testid` (e.g. `<div testid="Username"></div>`) then you would enter *testid* in the AUT configuration.

Chapter 5

Best practices

This section deals with how to best structure and design your tests in Jubula for successful automation. The topics covered here are by no means comprehensive – we recommend contacting the Jubula team for training and consulting offers.

5.1 Keyword design – how to structure your tests

This section deals with how to structure your tests in terms of *keywords*. It covers the following topics:

- How to decide when to make a keyword and structuring Test Cases to make them maximally reusable as keywords (→ page 226) .
- Where to add data – when to reference parameters and when to enter concrete values (→ page 226) .
- How to structure Test Suites (→ page 227) .

5.1.1 Making Test Cases into reusable keywords

One of the main advantages of Jubula tests is their structure. Tests are made up of small, reusable modules called keywords. A keyword is simply a Test Case which carries out a certain action or certain actions in the AUT. The Test Case is named to reflect what it does and is designed to be reusable.

5.1.1.1 General guidelines for keywords

As a rule of thumb, we recommend making a keyword for any action or actions you execute more than once in your test. This ensures that maintenance is much easier later – you will only have one place to change in the test, instead of many.

So what is a keyword? A good example is *Login*. It is obvious from the name what this Test Case does. It contains three of the Jubula unbound modules from the library Project:

- `ub_cti_replaceText` (which has the parameter `TEXT`)
- `ub_cti_replaceText` (which has the parameter `TEXT`)
- `ub_grc_clickLeftSingle` (no parameters)

In this keyword, the two `TEXT` parameters from the replace text modules would be referenced as `=USERNAME` and `PASSWORD` (→ page 81) .

Any time you need to perform a login in your test, you can reuse this keyword and enter the specific use data each time. This has the advantage that your tests will be easy to read, and that any maintenance you need to do to this keyword can be done in one place.

Other things that lend themselves well to being made into keywords are:

- Synchronized clicks on buttons (e.g. wait for component, then check enablement, then click).
- Tab selections (e.g. Select Content Tab, Select Options Tab in Tools Dialog).
- Recurring workflows (e.g. carrying out a search).
- Anything that you use more than once.

Many of these keywords will use the same modules from the unbound modules Project. However, the advantage of making a keyword is that all or most of the data that you need, and the component names, can be encapsulated into a module which has a recognizable name.

Some actions cannot be entirely encapsulated into a keyword with no parameters. Actions to enter text or select items from trees often require a different text or textpath each time they are reused. In this case, it is probably not worth writing a separate keyword for each possible data set.

If you realise later on in your test that you will need a module again, use the refactor function (→ page 64) to extract a reusable Test Case.



5.1.1.2 The size of keywords

Most keywords are likely to be reasonably small, because they are easier to combine and read when they are smaller. Also, it is easier to reuse smaller, less specific modules than larger ones. However, sometimes it can be worth making larger modules, especially for filling in tables, tabbed panes and dialogs. If you have to fill in the search dialog quite often, you could make a keyword containing all the necessary sub-keywords to fill in this dialog. You can parameterize the data that will change and maybe even enter default values that can be overwritten when the Test Case is reused.

5.1.1.3 Utility modules for keywords

In designing your tests, it may sometimes occur to you that there is a level between the unbound modules and your finished keywords. A good example is a module to to a synchronized text entry on a component:

- `ub_grc_waitForComponent`
- `ub_grc_checkEnablement`
- `ub_grc_checkEditability`
- `ub_cti_replaceText`

It is very likely that a test will have to enter various texts. The only difference for each case will be the text to enter and the component to enter it into. All other things (amount of time to wait for the component, the fact that the component should be enabled and editable etc.) will remain the same. A utility module to perform a synchronized replace text can solve this problem. It can be reused in Test Cases to replace the text on any component (→ page 104) and only requires the text to enter as a parameter.

Generally, it is worth making a utility module when the unbound module contains one or more parameters whose values will remain constant in your test (typically things like file

paths, operators, path types etc.). The utility module can parameterize the data that will change (e.g. boolean values, text entries, what to check etc.) and make your test specification and maintenance quicker.

5.1.2 Data in keywords

The rule of thumb for dealing with data is, like with keyword design, not to do anything twice. This means that you should set data that will not change (→ page 80) and parameterize data that will change (→ page 81).

Keywords that enter text, select from trees and check boolean values are all likely to need referenced parameters so that you can specify each time you reuse the Test Case what text to enter, what to select, and whether you expect a true or false result.

Parameters that deal with operators, search/path types, timeouts and delays can generally be fixed for your keywords and do not need to be parameterized. This reduces the amount you have to enter each time you reuse a Test Case and also means that data that will remain the same can be managed in a central place.



Don't worry if you are not sure at the beginning what will or won't change. You can add parameter references later in the Properties View and can remove unnecessary references in the *edit parameters* dialog (→ page 82).

If your Test Case needs to refer to the same data more than once, it may be worth thinking about parameterizing the data to the parent Test Case, so that you only have to enter the concrete data once. So, if your test creates a new project and then checks to see if the project name appears correctly, you should probably have the project name as a parameter of the whole Test Case to avoid having to enter the name in two places.

Alternatively, storing the data in a variable or as a central test data set may be another way to avoid having to enter the same data more than once.

5.1.3 Structuring your use cases and Test Suites

When it comes to executing tests, there are two important things to bear in mind when working with Jubula:

- All the necessary keywords for a single *use case* should be grouped together in one Test Case, which is named after the use case it tests. The use case should be complete – it should not require any data or component names. This makes it much easier to add and remove whole use cases from Test Suites and gives you a good overview of which use cases you have tested.
- Each use case should assume (and create) a well-defined state of the AUT and leave the AUT in a well-defined state. This is important for error handling later. It is a good idea to create a keyword called *app_startup* which creates the right state for the AUT, and to use this keyword at the beginning of each use case.

The well-defined state of your AUT may be completely empty, or may contain test data. Deciding what data your test needs and what data it will create itself is an important part of an automation strategy.



Use cases are then grouped together into Test Suites. The Jubula team uses the following Test Suites in their tests:

- FULLTEST
- FULLTEST_BROKEN
- WORK_(INITIALS)

The FULLTEST Test Suite contains all of the executable *Use Cases* from the Project. There can also be Test Cases which reproduce errors from the bug-tracking system, called *ticket tests*. Each ticket Test Case can be linked to the bug-tracking system via its ID.

The FULLTEST_BROKEN Test Suite contains any use cases which are currently broken and which have been removed from the FULLTEST so as not to affect any test reports or statistics. Put tests in the broken Test Suite if they have been broken for some time. Tests that are critical, or which should be fixed in the next build, should be left in the FULLTEST Test Suite.

The WORK Test Suites can be used by testers as a sandbox to execute single use cases during specification, or to replay tests with the intention of watching the execution.

5.2 Naming conventions

Because Jubula is a keyword-driven tool, names are particularly important. Tests are easy to read if the Test Cases are well-named. The following sections introduce some naming conventions for Test Cases, component names and parameters as well as Test Suites.

5.2.1 Naming conventions for Test Cases

You should name your Test Cases so that it is clear from the name what the Test Case does. Instead of vague or ambiguous names (e.g. Select), try and write names that describe the Test Case in terms of the action and the component it deals with. A few good examples are:

- Close Search Dialog with OK Button
- Select Project Language from Combo Box
- Fill in Project Wizard, containing:
 - Enter Project Name in Wizard
 - Select Project Language from List
 - Select Default Language from Combo Box
 - Close Project Wizard with Finish Button

The Test Case names do not have to exhaustively describe the component, or where it is in the AUT – using categories for the different areas of your AUT saves you having to do this (→ page 231) . The aim with naming your Test Cases is to make your tests readable and easy to understand.

You do not have to write Test Case names in CamelCase or with underscores – Test Case names are not used in the test executor, so there is no problem if they have spaces in them.

5.2.2 Naming conventions for component names

Component names in Jubula are your link between the test specification and the object mapping. Naming components well means that your mapping will be easy to understand and maintain, as you will always be able to tell what components you are referring to from the names alone.

We suggest the following three-part structure for naming components:

<LOCATION>_<FUNCTION>_<TYPE>

So the components in a login dialog could be named:

- *LoginDialog_Username_cti*
- *LoginDialog_Password_cti*
- *LoginDialog_Language_cbx*
- *LoginDialog_OK_btn*
- *LoginDialog_Cancel_btn*

Using the *New Component Name* feature in Jubula, (→ page 102) , you can create component names for your AUT at the beginning to set conventions the whole team should use.



The table below gives some common abbreviations to distinguish between component types.

5.2.3 Naming conventions for Test Suites and Test Jobs

Test Suite and Test Job names should not contain any special characters or spaces. As Test Suites and Test Jobs are entered via the test executor, it is important to have names that are easy to enter into a command line under any system.

Abbreviation	Component
app	Application
btc	Button Component
btn	Button / RadioButton / CheckBox
cbx	ComboBox
grc	Graphics Component
hyl	HTML Hyperlink
ctx	Component with Text
cti	Component with Text Input
lbl	Label
lst	List
mbr	Menu Bar
tpn	Tabbed Pane
tbl	Table
tbi	Toolbar Item
txf	TextField / TextArea / EditorPane / TextPane
tre	Tree
trt	Tree Table
brw	Web Browser

Table 5.1: Component Naming

5.2.4 Naming conventions for referenced parameters

Entering meaningful reference names for parameters in a Test Case helps the readability of your tests. If you are creating a utility module which doesn't as yet have a set function (→ page 225) , then you should reference the parameter with a reasonably abstract name.

For example, if you are creating a Test Case that will select an entry from any combo box, you could name the parameter =ENTRY as the function of the Test Case is not set yet.

Once you have a keyword that executes a specific action, you should try to name the parameters according to what data they need.

If you have a keyword to select a language from the combo box in the login dialog, you could name the parameter =LANGUAGE. In combination with the Test Case name, this makes it easy to understand what a keyword does and what data it needs.

This is especially important in keywords which contain Test Cases with the same parameter names. If you have two

=*TEXTPATH* parameters in your Test Cases, for example, you could name one of them =*TEXTPATH_TREE* and one of them *TEXTPATH_CONTEXTMENU* so that you and other testers using your keywords know what to enter where.

Parameter names cannot contain spaces.



5.3 Using categories to structure your tests

The amount of keywords you have in a Jubula test and the amount of components your test deals with can grow very quickly. For this reason, it is important to think about structuring the Test Case Browser and the Object Mapping Editor to make finding Test Cases and component names easier.

5.3.1 Best practices for structuring the Test Case Browser

We recommend using categories (→ page 68) to structure the Test Case Browser. This makes your keywords easier to find when you are creating tests.

We recommend using two main categories to organize the Test Case Browser: executable tests and modules.

5.3.1.1 Executable tests

The *executable test* category contains the following:

The executable use cases as described in (→ page 227)

A subcategory called *Event Handlers* which is discussed later (→ page 234) .

Any tests which reproduce errors from the bug-tracking system, called *ticket tests*. Each ticket Test Case is named with the ticket number to link it to the bug-tracking system.

5.3.1.2 Modules

The *modules* category contains two subcategories: *AUT-bound modules* and *unbound modules*.

AUT-bound modules

This category contains keywords that you have created to perform specific actions in your AUT. The AUT-bound modules category is split up into further categories, representing the dialogs and windows of your AUT. A typical structure for the AUT-bound modules category might look like this:

- Dialogs
 - Error Dialogs
 - Load Project Dialog
 - New Project Dialog
 - * Configure Language Page
 - * Configure Project Page
 - Save Dialog
- Main Window
 - General Tab
 - Security Tab

In each category, the Test Cases that execute actions in this area are stored. In the *Main Window* category, keywords to open dialogs from the menu or the toolbar could be stored. In the *Main Window - General Tab* category, you would find the keyword to select the security tab, to save the changes in the security tab etc.

The AUT-bound modules are all linked to the AUT in some meaningful way. For example, they could contain one or more of the following:

- sequences of actions specific to this AUT
- data and/or references specific to this AUT
- component names specific to this AUT

Structuring the AUT-bound modules in this way (Figure 5.1 → page 234) makes it easy to navigate through the keywords you have created.

Unbound modules

The *unbound modules* category contains modules which do not fit into the AUT-bound modules category. Good candidates for the unbound modules are utility modules, as mentioned in the previous section (→ page 225) . Other unbound modules could be sequences of actions that other AUT's could also use. Unbound modules generally contain no data or component names specific to one AUT. They may, however, contain default data you expect to use throughout your tests (like the operator, the pre-ascend in trees etc). At some point, the unbound modules could be moved into an external Project to serve as a library Project for more than one AUT (→ page 78) .

The unbound module category (Figure 5.1 → page 234) can be structured in categories according to actions or components.

Think about how many “layers” you need in your hierarchy – sometimes it is well worth making an unbound module, sometimes an unbound module would add an unnecessary extra layer. If your unbound module is not significantly different from the *ub_Test Case* from the library Project, it is probably superfluous.



5.3.2 Best practices for structuring the Object Mapping Editor

Organizing the Object Mapping Editor into categories (Figure 5.2 → page 235) makes it easier to find components you have mapped later on – especially if you have to remap any of them. As a general rule, the categories in the Object Mapping Editor will reflect the categories in the Test Case Browser in the *AUT-bound modules* category – the AUT should be split up into areas and dialogs, so that you have a simple yet hierarchical overview of your AUT in the Object Mapping Editor (→ page 231) .

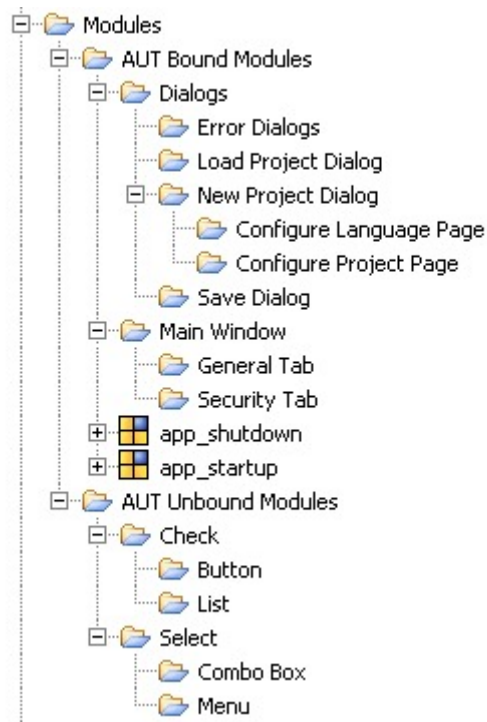


Figure 5.1: The Modules Category

5.4 Best practices for using Event Handlers to deal with errors

By using Event Handlers in the right way, you can ensure that your FULLTEST will run through all of its use cases, even if errors occur in the individual use cases.

You can also ensure that you get as much information as possible about the error, and where it happened.

The following sections deal with two uses of Event Handlers. First of all, an error handling strategy for the whole test is explained. This strategy can be used for any Project. The second section deals with Event Handlers on individual Test Cases to deal with local (expected) errors

5.4.1 Structuring global Event Handlers for your test

When automated tests are running, it is important to ensure that an error in the test will not result in the rest of the test not running. To do this, we recommend adding general Event

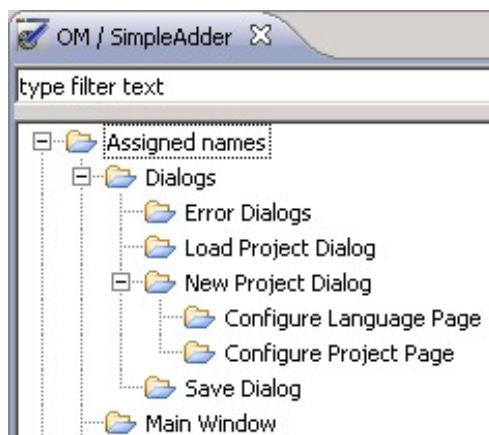


Figure 5.2: Categories in the Object Mapping Editor

Handlers to each use case, at the very top level of the use case.

The aim of adding Event Handlers here is:

1. To catch and deal with errors that are not known or expected during the test. For known/expected errors, Event Handlers can be used locally on individual Test Cases (→ page 237) .
2. To ensure that the rest of the test can continue – either in this use case, or in the next one.

A prerequisite for using Event Handlers in a global way is the structure of your tests. Each use case must be contained in a single Test Case, and the use cases must be independent from each other – starting in a well-defined state and leaving the AUT in a well-defined state (→ page 227) .

The following steps describe how to set up a global error handling concept in your tests. They assume that you have set up a category structure as described earlier (→ page 231) .

5.4.1.1 Creating general Event Handlers

The first thing to do is to create four Test Cases for the four event types:

Action Error: This Test Case contains the unbound module *restart application*

Check Failed: This Test Case is empty

Component Not Found: This Test Case contains the unbound module *restart application*

Configuration Error: This Test Case contains the unbound module *restart application*



We recommend placing any Event Handlers you write in a category *Executable Tests/Event Handlers*

5.4.1.2 Adding the Event Handlers to the use case

The next step is to add the Test Cases as Event Handlers to the use case so that they are activated when an error occurs.

1. Open a use case in the Test Case Editor. Remember that each use case should be independent, and entirely contained in one single top level Test Case (→ page 227) .
2. Add the *Action Error* Test Case to the use case as an Event Handler (→ page 146) . Specify *Action Error* as the error type, and *return* as the reentry type.
3. Add the *Check Failed* Test Case to the use case as an Event Handler. Specify *Check Failed* as the error type, and *continue* as the reentry type.
4. Add the *Component Not Found* Test Case to the use case as an Event Handler. Specify *Component Not Found* as the error type, and *return* as the reentry type.
5. Add the *Configuration Error* Test Case to the use case as an Event Handler. Specify *Configuration Error* as the error type, and *return* as the reentry type.

Now, any error in this use case (which is not handled by a local Event Handler within the use case) will result in one of these Event Handlers being activated. *Check failed* errors will mean that the test continues. *Action errors*, *Component not found* and *Configuration errors* will provoke a restart of the AUT. The test execution leaves the branch in which the Event Handler was nested (the use case) and carries on at the next Test Case (use case) or Test Suite.

The *app_startup* Test Case is important at the beginning of each use case in case the AUT has just been restarted (→ page 227) .



Carry out these steps for each use case as you create it.

In this way, unexpected errors can be dealt with in a way that allows them to be identified later on. The test does not stop at the error, but tries to carry on at the next relevant point. This is the reason why use cases should be independent from each other – to stop further errors occurring.

5.4.2 Using Event Handlers locally for specific Test Cases

Often, there are errors that occasionally occur at specific points in the test. A good example of this is a prompt dialog that sometimes occurs when saving, for example. For errors that can be expected in some way, you can add Event Handlers to the Test Cases that cause the error to deal with the error and carry on with the test without activating the global Event Handlers.

You can use local Event Handlers to specify a particular response to an error at this point – e.g. if you have a check for the existence of a project in the database and it fails, you may not want to continue with this test.

You can also use local Event Handlers to wait for, or create a state in the AUT that you require before the test can continue. In this case, you should structure your Test Case so that it checks for the status you require. If this isn't the case, the Event Handler is activated and takes the necessary steps to produce the status. Using the *retry* reentry type (→ page 148) , you can specify that the failed Test Step (the check) is retried after the Event Handler has been executed. An example of this is below.

5.4.2.1 Dealing with occasional dialogs

In the case of occasional dialogs, your Test Case in the test should contain an action which checks that the dialog does **not** exist (e.g. *Check Existence of Window* with the parameter *false*). If the dialog is not visible, the test will continue normally.

If the dialog is present, an error occurs. Add an Event Handler to the *Check Existence of Window* Test Case that clicks e.g. the cancel button in the window and waits for the window to close. The Event Handler should react to a *check failed* event, and should use *retry* as the reentry property. Once the window has closed, the failed Test Step (check existence) will be carried out again. This time, the check will succeed, and the test will continue. The Test Step will be marked as successful on retry.

Chapter 6

User interface

There are four types of area in the Jubula user interface:

Perspectives

A perspective is a collection of views, editors and browsers on the screen.

Browsers

Browsers let you see the Test Cases and Test Suites you have created in their hierarchical structures. There is also a browser for component names.

Editors

Editors let you modify, add and delete items. You can open editors by double-clicking on a node (e.g. Test Suite or Test Case) in a browser.

Views

Views show details about the currently selected item. If you single-click an item in a browser, the details in the views are read-only. If you select an item from within an editor, you can edit some of the values in the views.

You can change the way your user interface looks by moving areas, changing the colors, and hiding/showing views (→ page 175) .

To reset the default layout at any time, select:

Window → **Reset Perspective** .

6.1 Perspectives

6.1.1 The Functional Test Specification Perspective

The Functional Test Specification Perspective (Figure 6.1 → page 241) provides browsers, editors and views to let you create and edit tests.

It contains:

- The Test Suite Browser
- The Test Case Browser
- The Component Name Browser
- The editor area
- The Properties View
- The Data Sets View
- The Component Names View
- The Problem View
- The search result view (behind the Problem View)
- The Running AUT's View

The Component Names View, Data Sets View and Properties View show you details about the currently selected item in the browser or editor.

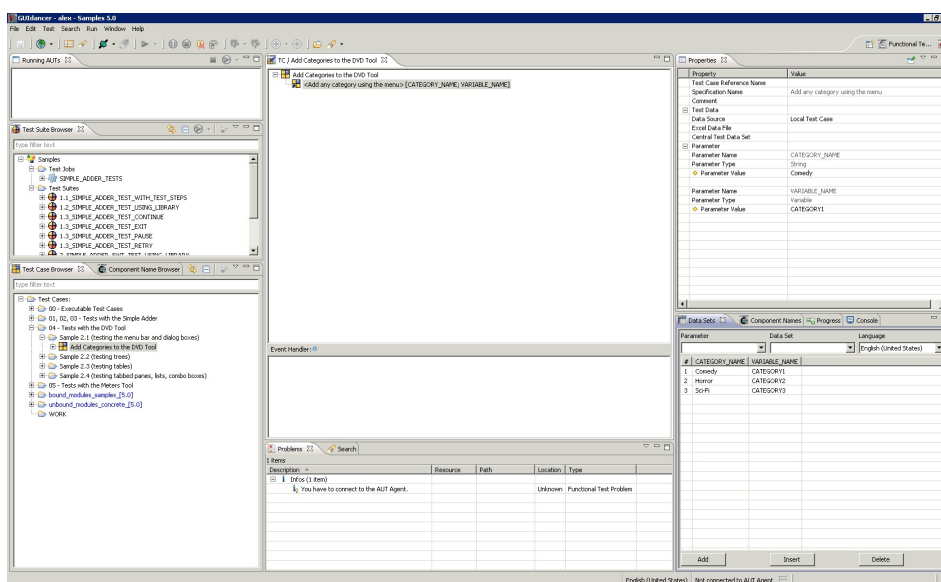


Figure 6.1: Jubula Specification Perspective

6.1.2 The Functional Test Execution Perspective

In the Functional Test Execution Perspective, you can see the following (Figure 6.2 → page 243):

- The Test Suite Browser
- The Test Result View
- The Properties View
- The Image View

You cannot edit in the Functional Test Execution Perspective, but you can see the results of tests that have been started interactively..

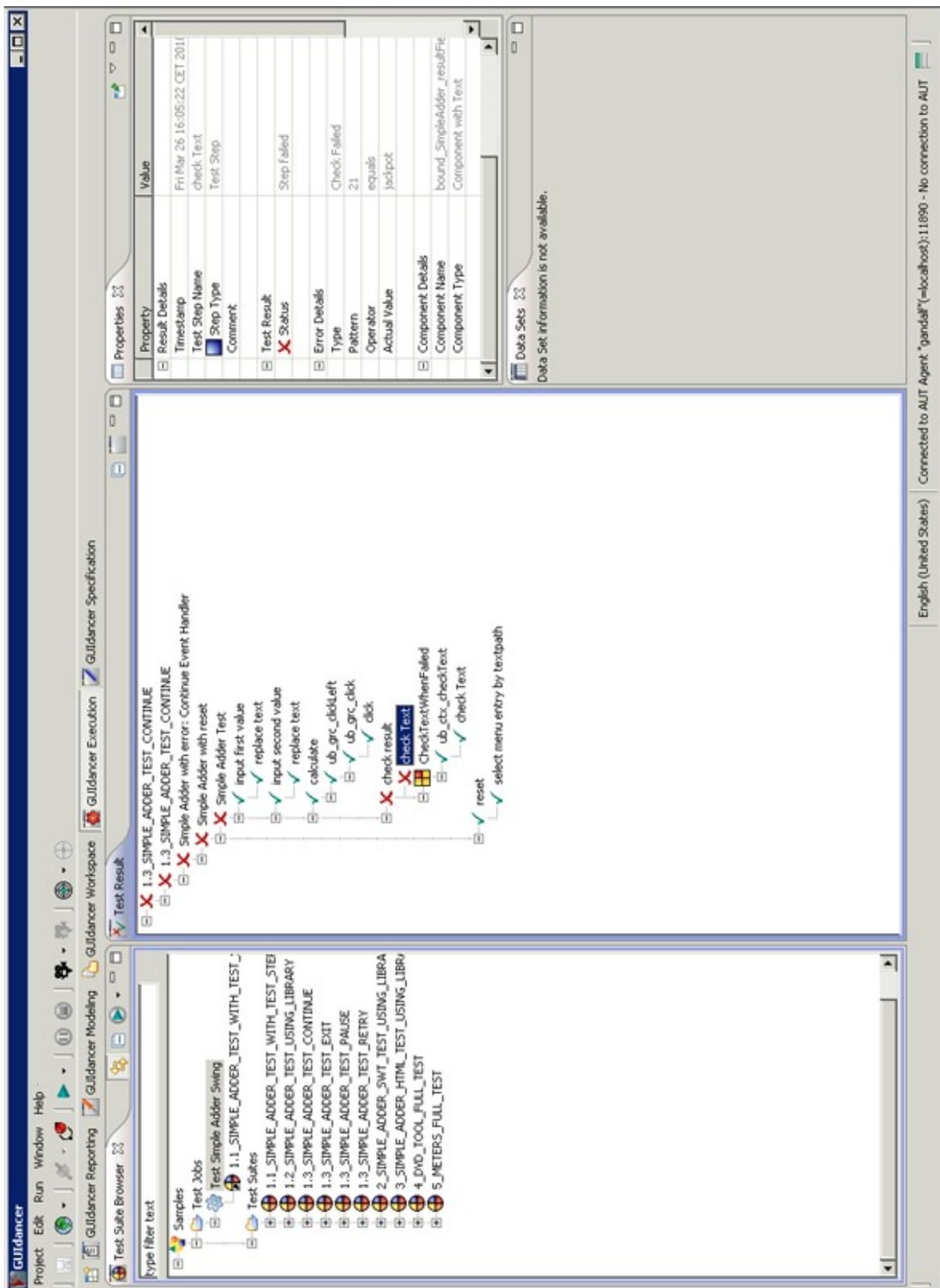


Figure 6.2: Functional Test Execution Perspective

6.1.3 The Functional Test Reporting Perspective

In the Functional Test Reporting Perspective, you can see an overview of all the test runs in the current database, for all Projects it contains. You can reopen test runs if the details for the report are still in the database (→ page 143) and analyze the test using the Properties View and the Image View.

The Functional Test Reporting Perspective (Figure 6.3 → page 245) contains the following views:

- The Test Result View
- The Properties View
- The Image View
- The Test Result Summary View

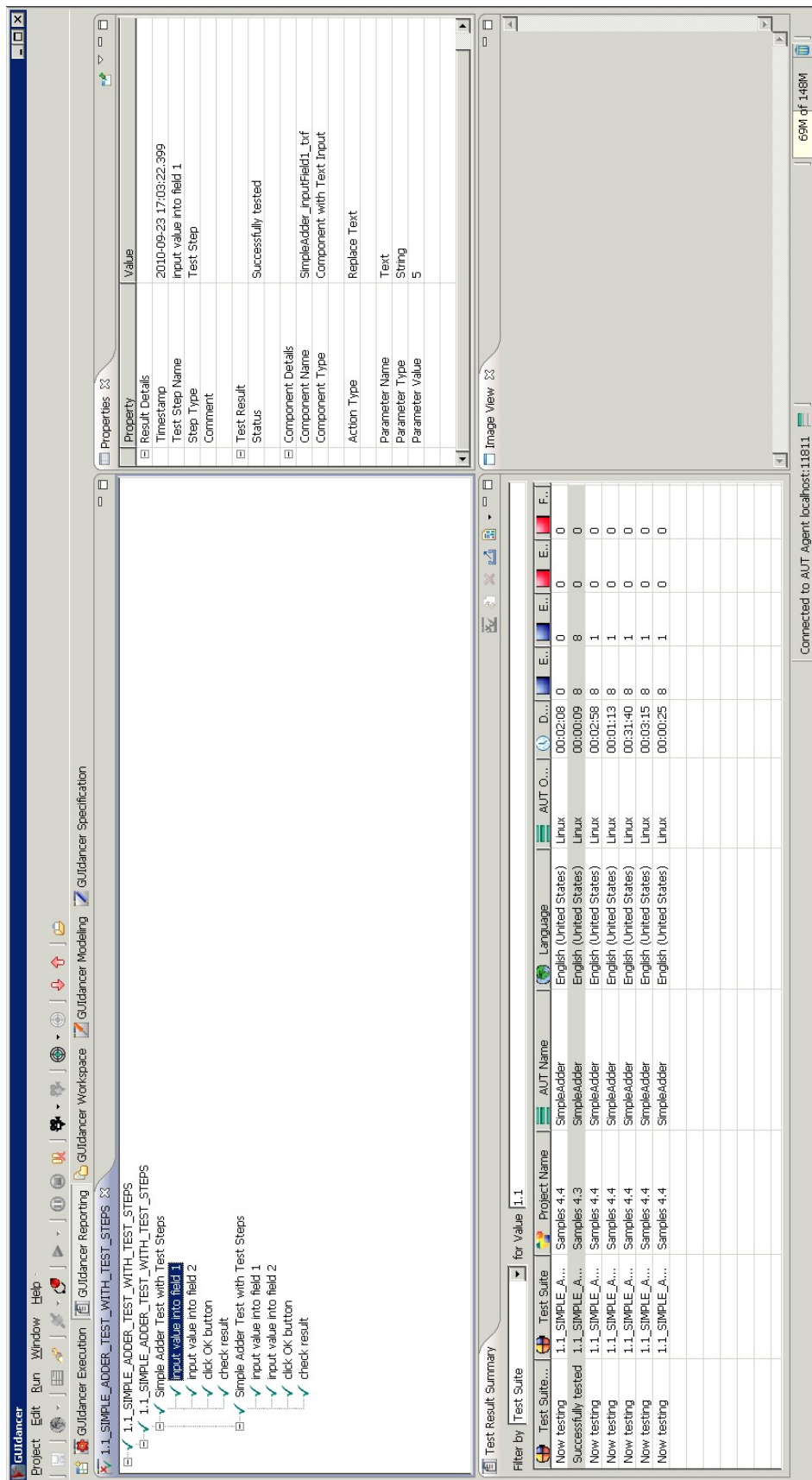


Figure 6.3: Jubula Reporting Perspective

6.1.4 The Jubula workspace perspective

In the workspace perspective (Figure 6.4 → page 247), you can view the Projects and files in your workspace. The workspace perspective contains the following:

- Navigator View
- An editor area to view e.g. Excel and HTML files.

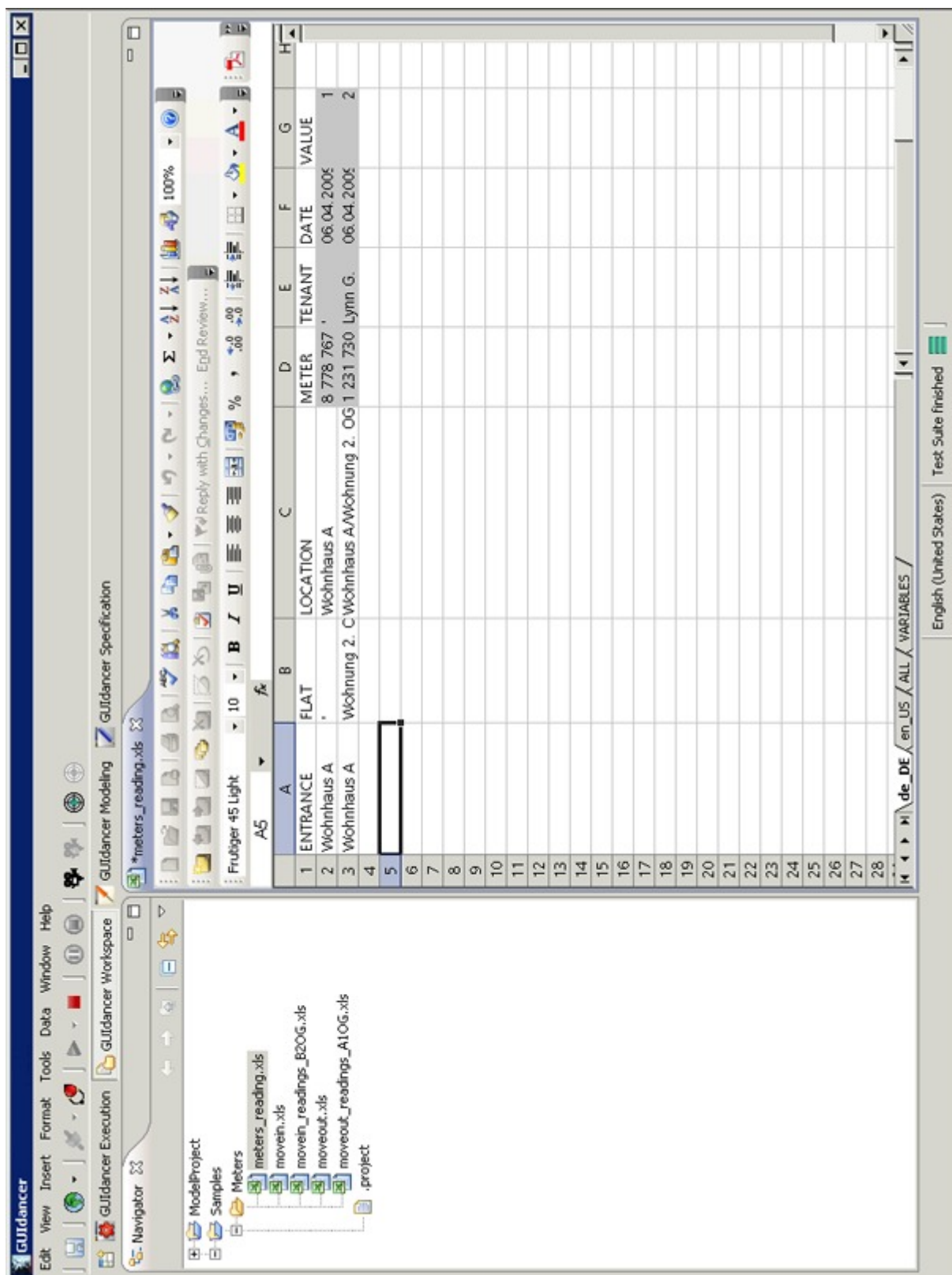


Figure 6.4: Jubula Workspace Perspective

6.2 Browsers

There are three browsers in Jubula:

The Test Case Browser lets you create and manage your Test Cases.

The Test Suite Browser lets you create and manage your Test Suites.

The Component Name Browser lets you see, merge add and delete component names.

6.2.1 The Test Suite Browser

The Test Suite Browser is by default in the upper left-hand corner of the Functional Test Specification Perspective. In this browser, you can create Test Suites and Test Jobs for executing tests.

More information on working with browsers is available in the Tasks chapter (→ page 58) .



You can filter in the Test Suite Browser using the field at the top. Use star * as a wild card.

6.2.2 The Test Case Browser

The Test Case Browser is by default in the lower left-hand corner of the Functional Test Specification Perspective. In this browser, you can create Test Cases, which are the "building blocks" in Jubula. They are used to group Test Steps and other Test Cases.

More information on working with browsers is available in the Tasks chapter (→ page 58) .



You can filter in the Test Case Browser using the field at the top. Use star * as a wild card.

6.2.3 The Component Name Browser

The Component Name Browser (Figure 6.5 → page 249) is by default in the lower left-hand corner of the Functional

Test Specification Perspective, behind the Test Case Browser. In this browser, you can see component names used in this Project and in other Projects. You can rename, add, delete and merge component names and can also find where you have used component names in your test.

You can filter in the Component Name Browser using the field at the top. Use star * as a wild card.

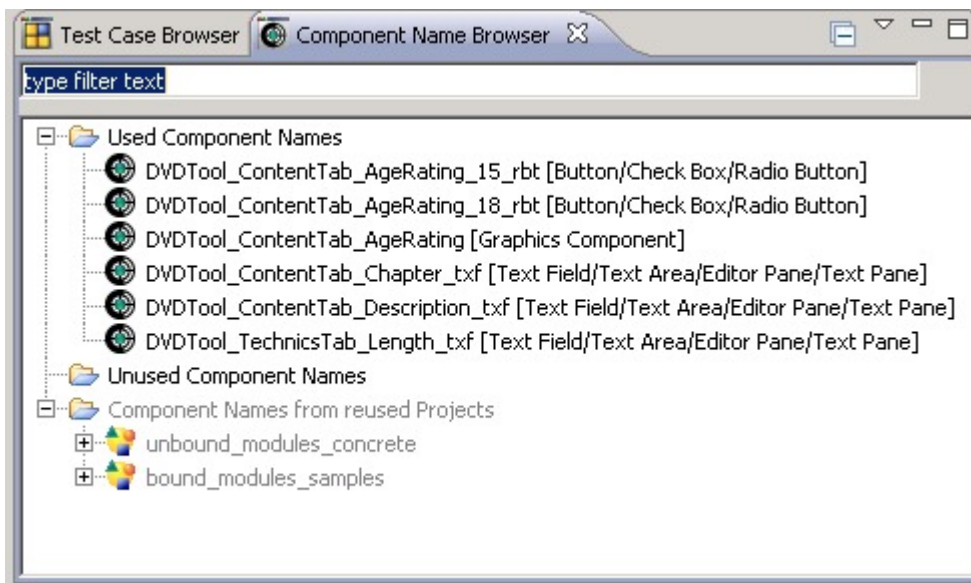


Figure 6.5: Component Name Browser

6.3 Editors

Editors appear in the editor area in the middle of the perspective when you double-click an item in one of the browsers or in the Navigator View.

You can tell what sort of editor it is in two ways:

1. The tab for each editor shows an ID-code for what type of editor it is:
 - "TC" for the Test Case Editor
 - "TS" for the Test Suite Editor
 - "OM" for the Object Mapping Editor
 - "TD" for the Central Test Data Editor

The tab also shows the name of the item the editor is for.

6.3.1 Test Case Editor

You can open the Test Case Editor by double-clicking a Test Case in the Test Case Browser.

In the Test Case Editor:

- You can create, modify and delete Test Steps.
- You can add and delete other Test Cases.
- You can add Event Handlers. They appear in the Event Handler area below the Test Case Editor.
- You can alter test data and component names for the Test Case.

When you single-click items (Test Cases, Test Steps) in the Test Case Editor, the support views (Properties View, Component Names View and Data Sets View) show details about this item. Using these views, you can edit the properties, test data and component names for this item.

6.3.2 Test Suite Editor

You can open the Test Suite Editor by double-clicking a Test Suite in the Test Suite Browser.

In the Test Suite Editor:

- You can add Test Cases to Test Suites.
- You can alter the Test Suite properties (e.g. the AUT it uses, the default Event Handlers).
- You can alter test data for the Test Cases shown.



In the Test Suite Editor, you can only alter details for the top-level Test Cases for each subtree.

When you single-click Test Cases in the Test Suite Editor, the support views (Properties View, Component Names View and Data Sets View) show details about this Test Case. Using these views, you can edit the properties, test data and component names for this item.

6.3.3 Object Mapping Editor

Each AUT in a Project has an Object Mapping Editor. You can open the Object Mapping Editor by right-clicking on a Test Suite in the Test Suite Browser and selecting *"open with Object Mapping Editor"*.

You can also open the Object Mapping Editor by single-clicking the chosen Test Suite and clicking on the *"start Object Mapping Mode"* button on the toolbar. This also starts the Object Mapping Mode.

The Object Mapping Editor has four tabs, a split view, a tree view, a table view and configuration tab. In the Object Mapping Editor:

- You can see all the component names you have used in the Test Suite you selected, and any other Test Suites which use the same AUT.
- You can see any technical names you have collected from the AUT.
- You can see the component and technical names you have assigned to each other to complete mapping.

6.3.4 Central Test Data Editor

You can open the Central Test Data Editor via the button on the toolbar.

In the Central Test Data Editor:

- You can create, modify and delete central test data sets.
- You can search for places where central test data sets have been used.

6.4 Views

Jubula has the following views:

- Properties View
- Data Sets View
- Component Names View
- Test Result View

- Problem View
- Search result view
- Navigator View
- Console
- Inspector View
- Test Result Summary View
- Running AUT's View
- Image View
- Progress View

The first three views are *support views*. Their content changes depending on the current selection – they show details corresponding to the currently selected item.

Use `Window` → `Show View` → `Other` to see a full list of views that can be opened in the ITE.

6.4.1 The Properties View

The Properties View is in the upper right-hand corner of each perspective. The Properties View shows details about the selected item in a browser or editor.

If you select something from a browser, you can see its details in the Properties View, but you can't edit them. You can only edit details from within an editor. Read-only fields in the Properties View have a lock icon at their left-hand side.



Test Suite details

If you select a Test Suite, you will see the following:

- The Test Suite name.
- Any comments for the Test Suite.
- The step delay for this Test Suite.
- The name of the AUT this Test Suite uses.
- The reentry properties for the default Event Handlers.

Test Case details

If you select a Test Case, the Properties View shows the following:

User interface

- The Test Case name.
- Any comments for the Test Case.
- The data source details for this Test Case
- Any parameters/parameter values if references have been used in the Test Case.

Test Step details

If you select a Test Step, you will see the following:

- The Test Step name.
- Any comments for this Test Step.
- Details about the component in the Test Step:
 - The component type.
 - The component name (given by you).
- Details about the action in the Test Step.
- Details about the parameters in the Test Step. For each parameter you will see:
 - The parameter name.
 - The parameter type (e.g. if it is an integer, a string etc).
 - The parameter value (either a concrete value or a reference).



Test result details

If you select an item from the Test Result View, you will see the following:

- The result details:
 - The name of the item.
 - What type of item it is.
 - Any comments for the item.
- The test result – if it passed or failed.
- For Test Steps, you can also see the component, action and parameter details. If the Test Step failed, you can also see the error details.
- For Event Handlers, you can see the type of Event Handler it was, and what the reentry property was.

Icons in the Properties View

When the Properties View contains data (i.e. a filled-in *parameter value* field), there are certain icons to help you understand the data.

-  original data
-  overwritten data

- Values which are the same as in the original specification display a small gray diamond to the left of the field.
- Values which have been overwritten at a place of reuse show a small yellow diamond.

6.4.2 The Data Sets View

The Data Sets View is in the lower right-hand corner of the Functional Test Specification Perspective by default. It lets you add, modify, delete and see any data for Test Cases. For each Test Case containing references or for each central test data set, you can add data sets in this view. This is also the view where you can translate your data and overwrite data when you reuse a Test Case. When data in the Data Sets View are uneditable, they are shown in gray. Values in black may be edited.

To make entering data easier, pressing »ENTER« will spring to the next cell or row, or to the "Add" button if no more rows have been added. The »TAB« key springs to the "Add" button.

6.4.3 The Component Names View

The Component Names View is in the lower right-hand corner of the Functional Test Specification Perspective by default, just behind the Data Sets View. It lets you see the component names used in a reused Test Case, and add more component names. This means that you can use a Test Case to execute the same actions on different components.

6.4.4 The Test Result View

The Test Result View is in the middle of the Functional Test Execution Perspective by default. It can also be displayed in the bottom right-hand corner of the Functional Test Specification Perspective via the preferences. It follows and displays the results of the test execution. Each Test Step and Test Case

is marked with a green tick or a red cross depending on if it passed or failed.

If you select an item in the Test Result View, you can see its details in the Properties View. If the Project is open and the item is still available, you can double-click on a Test Step or Test Case in the Test Result View, to highlight it in the Test Suite Browser. Double-clicking on this item in the Test Suite Browser will open the editor for this item in the Functional Test Specification Perspective.

6.4.5 The Problem View

The Problem View is by default at the bottom of the Functional Test Specification Perspective, under the editor area. The Properties View helps you to work with Jubula by giving you information, warnings and errors. Double-clicking on a message will either carry out the necessary action or open a dialog/editor to sort out the problem, if this is possible.

6.4.6 The search result view

The search result view (Figure 6.6 → page 256) is at the bottom of the Functional Test Specification Perspective, under the editor area, and just behind the Problem View by default. If you perform searches with Jubula, the results are shown here. For most searches, if you double-click on an item in the search result view, the corresponding item is highlighted in the Test Case Browser or Test Suite Browser. You can see a history of previous searches, and re-run them directly from within the view. You can also refresh the current search.

6.4.7 The Navigator View

The Navigator View is at the top right-hand corner of the workspace perspective by default. The Navigator View gives you a view of your workspace and the files in it.

6.4.8 The console

The console is shown in the bottom right-hand corner of the Functional Test Specification Perspective. It lets you follow certain processes in Jubula, for example, the import of Projects.

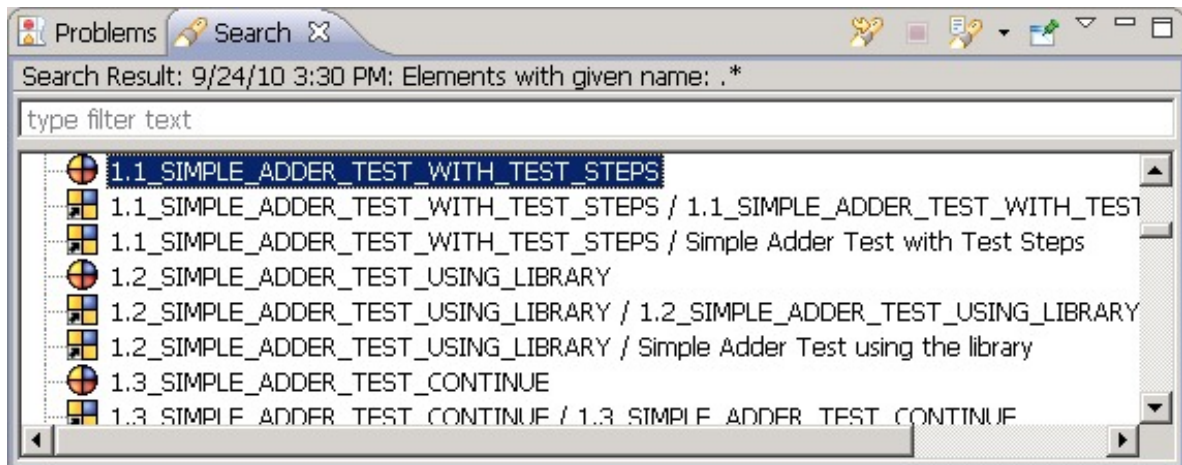


Figure 6.6: Search Result View

6.4.9 The Inspector View

The Inspector View can be used to help you test RCP AUT's which use the GEF framework. When the inspector is activated, figures can be clicked in the GEF canvas, and their textpath (used to identify them in the test) is shown. The Inspector View is shown by default in the bottom right-hand corner of the Functional Test Specification Perspective.

6.4.10 The Test Result Summary View

The Test Result Summary View offers an overview of all tests that have run in this database. It can be filtered and sorted. The Test Result Summary View is displayed in the Functional Test Reporting Perspective.

6.4.11 The Running AUT's View

The Running AUT's View is above the Test Suite Browser in the Functional Test Specification Perspective. It shows an overview of all running AUT's and can be used to stop AUT's and to start Test Suites.

6.4.12 The Image View

The Image View is used to see automatically generated screenshots of errors that have occurred in tests.

6.4.13 The Progress View

The progress view shows which longer-running actions are currently being executed. These can include starting AUT's, connecting to the AUT Agent, and executing tests.

6.5 The status bar

The status bar in the lower right-hand corner of the perspective gives important information about the AUT Agent status, working language, AUT and the observation mode and Object Mapping Mode.

Chapter 7

Concepts

The following sections introduce the concepts and capabilities of Jubula.

7.1 Overview

Jubula is based on the *Eclipse* development platform, a platform-independent, open-source framework for developing applications.

With the ITE, you can specify, modify, execute and analyse tests. The user interface offers various ways of working with Jubula, so that you can choose the way that suits you best. You can alter the way the client looks, and the way it behaves. You can work using the menu bar, the tool bar, context-sensitive menus and shortcuts.

The important thing to remember is that Jubula lets you work the way *you* want to.

7.2 Testing with Jubula

The information in this section help you to understand how Jubula tests your application, and how you can write Jubula tests for the actions you want to test.

7.2.1 Understanding how Jubula works

7.2.1.1 Actions

Jubula supports "high-level" actions. By this we mean that some of the actions which can be chosen for a component

actually carry out a number of actions. This makes the creation of Test Cases easier and quicker, and makes them more understandable.

For example, Jubula offers two actions to enter text into a text field. The action *Replace Text* selects the whole text in the component and then enters the text input. This effectively overwrites any text which was already in the component. The *Input Text* action clicks once in the component and then enters the text, which means that any text previously in the component remains.

Another example is the selection and navigation through trees and menus. Like a human tester, Jubula works using the path to the item to select, and doesn't first have to select the individual sub-items in the path.

High-level actions can be used on all standard components without problems. However, if the behaviour or look-and-feel of a component has been changed (e.g. double-click in a text field brings up a dialog instead of selecting the word) then some high-level actions may not work. In this case you will have to combine the low-level actions offered by Jubula. Often, there are many ways of achieving the same effect in your test, and it may just be a case of trying out a few different ones to see which works for you and your AUT.

When writing tests, it helps to be aware of things that a human tester does implicitly. Often, these are things that have to be explicitly stated in an automated test, like waiting for the application to be ready, or selecting an item before opening the context menu, or even pressing enter to close a cell editor in a table.

7.2.1.2 Test execution

Jubula generally executes tests on your AUT by sending real clicks and key presses, in the same way that a manual tester would. This means that a Jubula test will give you the same results as a manual test – if a menu is disabled, an item can't be selected and so on.

Once clicks and key presses have been sent, Jubula waits for confirmation from the AUT that they have arrived (manual testers use their eyes, Jubula uses confirmers).

If the AUT or the components in it need to be scrolled to access the area being tested, scrolling happens automatically. In the same way, if a tree is collapsed, you do not need to specify

an expand action before selecting a node. The expansion is done as part of the selection.

7.2.2 Standards conformance

Jubula is designed to test components in their original and intended implementations. If these components have been derived and altered such that they no longer respond to actions in the same way (i.e. rewriting the single-click event for a textbox, non-standard responses to certain key combinations, etc.), Jubula may no longer be able to test these components. It is therefore recommended for the purposes of testing with Jubula – and also for the sake of conformance to usability standards – that the toolkit conventions are held to.

There may, of course, be situations where exceptions are necessary. For this reason, Jubula has an API via which testing classes can be implemented so that even changed components can be tested. See the manual on extending Jubula for more information.

7.3 Architecture

Jubula is a client-server application. We refer to the client part as the Integrated Test Environment or ITE, and the server component as the AUT Agent.:

The ITE is where tests are specified and evaluated.

The AUT Agent controls the application under test (AUT) and executes the specified tests.

You can install the ITE and AUT Agent on different machines; even on different platforms. This lets you specify and run your tests on different machines, and means that you can run the same test on different platforms.

7.3.1 ITE

The ITE is the interface between you and the AUT. It provides you with browsers to see Test Cases and Test Suites, and editors to modify them.

In the client, you specify your tests without any programming. Jubula lets you write tests modularly, structure them to fit your

needs, and easily change them to follow developments in the AUT.

7.3.2 AUT Agent

The AUT Agent is the link between your specified tests and the AUT. Once the AUT Agent has started the AUT, object mapping can take place, and tests can be observed and executed.

You can install the AUT Agent on the same machine as the client or on any other machine in the network, as long as the AUT can also run on the machine. The AUT Agent can even run on a different platform than the client.

This separation of the client and AUT Agent means that you don't have to sacrifice control of your machine during test execution. It also means that you can test your AUT on different systems using the same test, and from the same client.

7.3.3 Working with the ITE and AUT Agent on different machines

You can install the ITE on any machine in the network, and then install the AUT Agent on various other machines in the network.

Within the ITE, you can define configurations for your AUT. The configurations specify where Jubula should look for the AUT jar or exe file, the JRE binary etc., and on which machine it should look.

Jubula lets you specify different AUT Agent hosts and port numbers to connect to.

7.4 Database structure

You can store your Projects in a single-user database or in a multi-user database (recommended).

7.4.1 Supported systems

Jubula uses *Eclipse Link* as an object relational mapping layer.

Jubula is tested with *Oracle*. Other database solutions supported by *Eclipse Link* may also be used as a database, though these have not been tested by our team.

7.4.2 Single-user

If you work on a single-user database, only one tester can work on a specific Project at a time. This also applies to the test executor – the ITE and the test executor cannot access a single-user database at the same time.

This method reduces the complexity of the system and its configuration, and should really only be used for demo purposes. Project sharing remains possible, however, through Jubula's import/export capability.

7.4.3 Multi-user

When using a multi-user database, multiple testers can work on a single Project concurrently.

Within such a scenario, all testers currently working on a Project may view and execute all parts of it. As soon as one tester alters a part of the Project, a lock is placed on that unit to disallow other users from making changes of their own. They are, however, still able to view and execute a locked test unit. Once the user with the lock has saved all changes, the lock is removed, and the test unit is once again available for editing.

This database scenario is well-suited for larger testing operations, allowing for tight collaboration among testers and reducing the overall time needed to specify complex testing projects.

7.5 Approaches to testing

Jubula supports modular testing. You can either create modules in advance and combine them to make larger Test Cases, or you can create modules from existing Test Cases when you realize you need them.

7.5.1 Writing modules in advance

If you can identify which reusable modules a test will require in advance, then you can specify them in a general or abstract way, making them easy to adapt and reuse.

These small units are used to construct other modules or building blocks, which are more adapted to the task at hand. Concrete details can be added as needed.

One of the advantages of this approach is that tests are flexible and easy to maintain, since a change made at a central point can update many places where that Test Case has been reused. Tests are also generally more efficient, because time is not wasted specifying what is essentially the same Test Case over and over again.

The difficulty with this approach testing is that it requires a good knowledge of the structure of the test – you have to know which Test Cases are going to be required multiple times, and this is often not easy to recognize at the beginning of testing.

7.5.2 Creating modules from existing Test Cases

It is often initially more intuitive to start writing a test in a linear fashion, adding steps as necessary. Each step is specified for the current purpose, and contains all the necessary details (e.g. data).

The possible problem with this method is that similar or identical parts are often separately and multiply specified, which means that making changes at a later point can be difficult and time-consuming.

To avoid this, Jubula lets you extract Test Cases from other Test Cases to combine them into reusable modules (→ page 64) .

7.5.3 Choosing a method

Obviously, these two approaches lie on a continuum. Jubula lets you “mix-and-match” so that you can choose the approach that works best for you – a combination of the two or one method for one Test Case and the other for another Test Case.

For example, if you know that a certain test procedure (e.g. logging in) has to be executed frequently, you can specify a Test Case to do this, and specify it to be easily reusable (using references as data-placeholders, and using general component names to be reassigned). For parts of the test which are not likely to be carried out more than once, you can specify them with data and definite component names.

7.6 Test hierarchy

The following sections describe the test hierarchy in Jubula. Basically, you can think of the test hierarchy as a tree structure which can be of any size, complexity and depth (Figure 7.1 → page 265).

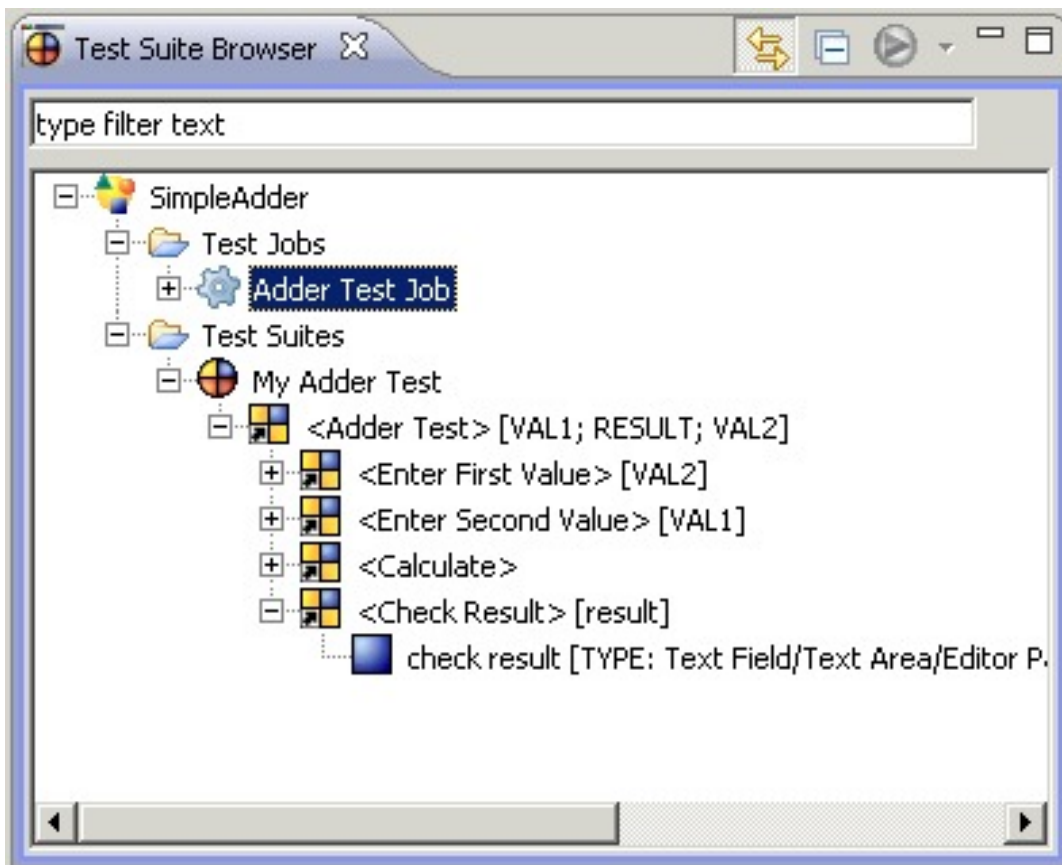


Figure 7.1: Test Hierarchy

The hierarchical structure of Jubula gives you flexibility in your tests to make changes and keep the test in touch with current software development.

7.6.1 Test Steps

A Test Step is the smallest unit in the test hierarchy. Each Test Step represents one action on one component (or user-interface element) in the AUT.

The interaction is composed of three details, which we refer to as "CAP" (component, action, parameter):

Component: a component is a user-interface object (e.g. a button, a combo box).

Action: the action is the operation to be executed on the selected component. Each component has a number of actions which can be executed on it, for example, buttons can be clicked, an input can be made into a text field.

Parameter: the parameter(s) are the data or variables associated with an action. When a button is clicked, the parameter is the amount of clicks. When you are entering text into a text field, the parameter is the text you want to enter. The amount and type of parameters depends on the action.

The only detail which needs to be fixed at this point in the specification is the action. The actual component to be tested and the parameters can be added or changed later on.

The specification is also separate from the AUT. You give the component you specify a *component name*, which you use to identify the component in your test. This component name is assigned to the actual component in the AUT at a later point. In this way, specification can begin before the AUT is available. An example if a Test Step could be entering text (e.g. `hello`) into a text field:

Component	Text field/text area/text pane/...
Action	Enter text
Parameter	hello



We recommend using the Test Cases in the library Projects installed with Jubula instead of writing Test Steps. Using the Test Cases from the library Projects saves time and improves the flexibility of your tests.

7.6.2 Test Cases

Test Cases are a level up from Test Steps in the test hierarchy. They are the building blocks or keywords from which other Test Cases and Test Suites are made. They are the reusable elements of Jubula.

A Test Case can contain any number of Test Steps and other Test Cases nested to any depth. Test Cases are given user-defined names.

Jubula's design allows Test Cases to be reused – in other Test Cases and in Test Suites. There are no limits on how often a Test Case can be reused, as long as no circular dependencies (infinite loops) are created.

7.6.3 Test Suites

Test Suites are the containers for Test Cases to be executed and can contain as many Test Cases as necessary. Each Test Suite must be bound to one AUT which it will test. An AUT can be bound to more than one Test Suite.

An important feature of Jubula is the ability to test a program modularly. Program parts can be tested as and when they are delivered using the specific Test Suites for each part. Only the chosen Test Suite in a project will be executed, which means that testing can begin whether all program components are available/functional or not.

7.6.4 Test Jobs

Test Jobs are the containers for Test Suites which are to be executed in a sequence. A Test Job can contain Test Suites which test different AUT's or different instances of the same AUT (with some restrictions, see the previous section (→ page 113) for details). Each Test Suite in a Test Job is bound to an AUT and specifies which AUT it will test based on the AUT ID.

7.6.5 Projects

Jubula organizes tests into Projects. A Project is the top-level container in Jubula. Projects contain runnable tests and the details about the AUT's to be tested (how to start them, in which language etc.).

You can work jointly on Projects using the multi-user database, and you can also share Projects using the import and export functions.

7.7 Reusability

Test Cases are the reusable units in Jubula. Test Cases can be referenced in other Test Cases to create modularly built tests. Being able to reuse Test Cases means that you should put some thought into how to design them so that they are as specific as they need to be while also being generic enough to reuse in different situations.

You can improve the flexibility of a Test Case by:

- Using references (→ page 81) instead of concrete data if the data can change. References allow you to feed a Test Case with different data each time you reuse it.
- Using the abstract components (e.g. component with text, component with text input, graphics component) (→ page 268) to specify your tests wherever possible.
- Allowing the component it tests to be defined when you reuse it (→ page 103) .

7.7.0.1 Abstract, concrete and toolkit specific components

Jubula has three levels of components:

Abstract components are general, high-level components from which other components are derived. They are described in terms of what features a component has, e.g. graphics component, component with text. They group actions together which can all be executed on components of this type.

Concrete components are components which are available to all graphical toolkits, but which are restricted to a certain component type, e.g. combo box, list.

Toolkit specific components are the most specific components in Jubula. They are only available for certain toolkits. For example, a HTML link is a component which is only available in Web applications.

We recommend that you specify your tests as abstractly as possible, and as concretely as necessary. If you want to create a Test Case to click a button, it is better to use the abstract component *graphics component*. The *graphics component* also contains the *click* action, and has the advantage that the Test Case can then be used on other components than buttons. If you want to select a cell from a table, however, you will have to use the concrete component *table*, because this is the highest level which offers this action.

The more abstract your specification, the more flexible your Test Cases are.

7.8 Multi-lingual testing

7.8.1 Project and AUT languages

With Jubula, you can test different language versions of your AUT with the same test. All you have to do is translate the data for the test.

To make multilingual testing possible, you have to tell Jubula which languages you are using:

Project language(s): These languages are specified during Project creation and are valid for all AUT's in this Project. You will be able to have AUT's which support one or more of these languages.

Default language: One of the Project languages is the default language. This is the language which will automatically be set as the working language when you open the Project.

AUT language(s): When you are defining an AUT, you choose one or more AUT languages from the selection of Project languages. These languages are the languages the AUT supports. You can start the AUT in these languages and write test data for these languages.

The working language: In the Jubula toolbar, you can change the working language to any of the Project languages. The working language determines which Test Suites you can edit (only those whose AUT supports this language) and the language the AUT is started in.

7.9 Object mapping

Object mapping is the process which joins your *component names* from your Test Steps to the actual *technical names* for the components in the AUT. Because object mapping can be the last step before execution, you can start specifying tests before the software is even available.

Object mapping consists of two steps:

1. Collecting the technical names for the components from the AUT.
2. Assigning these technical names to the component names you used in your Test Steps.



Object mapping is bound to the AUT, not to a Test Suite. All Test Suites which use the same AUT share an object map.

7.9.1 Component names

When you specify Test Steps, you specify component names for the components you want to test. When you add Test Cases to a Test Suite, all component names used in the Test Cases are collected by the Object Mapping Editor. You can then collect the technical names from the AUT.

7.9.2 Technical names

You do not need to consult the developers to collect technical names; you use the running AUT to collect the names. If the developers have named the components, this name will be collected. If the developers have not named the component, Jubula generates a name.



Jubula addresses high-level components in the mapping mode. You will not be able to map individual menu entries or tree nodes, for example. Instead, you map the tree component, and specify which entry/node to select/check as a parameter for the action.

7.9.3 Assigning technical names to component names

Once you have collected the technical names from the AUT, you can “assign” them to the component names using drag-and-drop. In this way, you are telling Jubula which real component you are referring to in each Test Step.

The fact that the object map is separate from the specification means that any changes that need to be made to update a test only need to be made once.

7.9.4 Locating components during test execution

The component recognition system used by Jubula uses various details about mapped components to find the right component during the test. The location of the component in the AUT hierarchy, the components near to it and the name of the component all play a role in component recognition. For each possible component, Jubula calculates the similarity to the originally mapped component. The component with the highest result is selected.

This method makes object mapping generally resilient to changes in the AUT.

7.10 Test execution

The prerequisites for test execution are covered in the previous chapter (→ page 136) .

7.10.1 Test Step execution

Jubula executes tests on the following basis:

- A user-defined component is located in the user interface of the AUT.
- The specified action is executed on this component, with the parameters you gave for the action.

Jubula bases the success of an action on whether it was carried out or not. There are no implicit checks which verify the state of the application to check if the execution of an action

actually had an effect. For example, if an action is carried out to click a button, this Test Step will be marked as successful even if the button is disabled. From Jubula's perspective, a click was executed on the button component.

It is worth bearing this in mind if a test has failed, especially because of a "component not found" error. What may have happened is that a further component was not found due to a dialog still being in focus, because the "close" button was disabled, for example. Although a click was executed on this button, it did not have the desired effect. The test continued, but ran into problems.

7.11 Observing user actions

As explained in the introduction (→ page 8) and in the section on how to record (→ page 159) , we believe that recording tests has various disadvantages in any tool.

However, you can use Jubula to observe Test Cases in a running Java AUT. You can record actions and checks in your AUT, and the output for these actions is Test Steps.

You can create the same tests with observing as you can with specification. The real differences are:

- object mapping is carried out automatically in observing mode. Because of this, you do not provide a component name, but one is created by Jubula.
- you must use concrete values for parameters in the observation mode. You can (and should), however, change these to references in the specification perspective later so that your tests are more reusable.

You can reuse observed Test Cases in the same ways as you can reuse specified Test Cases.

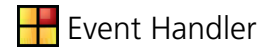


The observation mode supports high level actions. To find out exactly how an action works, look up the component and its action in (→*Reference Manual* p. 21)

7.12 Event Handlers

Jubula offers Event Handlers to react to errors during test execution and to continue the test in a way you define.

An Event Handler is a Test Case used with a special function. It is only executed when a certain type of error occurs. There are four different types of error in Jubula, and each Test Case can have an Event Handler for each error type. You can specify different reactions to each different error type, so that serious errors can stop the test, but unimportant errors will not affect the execution.



Because Event Handlers are normal Test Cases, they can also contain Event Handlers, can do any of the things that a normal Test Case can do. The only restriction is that you cannot add a Test Case to itself as an Event Handler.

7.12.1 How Event Handlers work

When an error occurs in a Test Step, its parent Test Case is searched for an Event Handler for this type of error. If no Event Handler is found, the parent Test Case of this Test Case is searched and so on. Once a corresponding Event Handler is found, it is activated.

If no Event Handler is found in the highest Test Case in this part of the tree, a default Event Handler (set in the Test Suite properties) is activated.

Each Event Handler requires an "*Event Type*" and a "*Reentry Type*". For information on these terms, see the section in the Tasks chapter (→ page 146) .

7.12.2 Default Event Handlers

Default Event Handlers are a part of the Test Suite and can be considered as being present at the Test Suite level. They are activated when no Event Handler has been found for the event type in the part of the tree where the error occurred. The default reentry type for each event type can be specified in the Properties View for the Test Suite Editor.

The reentry types for the default Event Handlers are the same throughout the tree. Everywhere a certain type of error occurs, the same action will be taken, unless the part of the tree where the error occurred has its own Event Handler for that event type.

7.12.3 Customized Event Handlers

Customized Event Handlers are essentially Test Cases, and can be empty or contain other Test Steps/Test Cases. They can be used to specify particular behavior in case of an error in one part of the test tree, or they can be placed so as to affect how the test continues. If an Event Handler contains Test Steps/Test Cases, the specified reentry will happen when the Event Handler has been executed.

Customized Event Handlers are valid for all Test Cases under the Test Case where the Event Handler is nested, unless one of these child Test Cases has an Event Handler of its own for this error type.

7.13 Extensibility

Jubula is extensible in two ways:

- New actions can be defined for existing components
- New project-specific components which are derived from standard components can be added and actions for them created.

Both of these processes require somebody to program the new action or component, and to link the new action or component to the AUT Agent process.

For information on how to do this, see the separate guide to enhancing Jubula.

7.14 Summary

After reading this chapter, the structure and concepts of Jubula should be clear. You can specify reusable Test Cases to make Test Suites in a Project. To enhance the reusability and structure of test elements and the ease of use of Jubula, there are various features supported by the program. To look at these features in more detail and find out how to use them, see the "*Tasks*" chapter (→ page 21) .

Chapter 8

Glossary

Action A method which is carried out on a *Component*. An *Action* is part of a Test Step, and serves to either execute a command (e.g. click, enter text), or verify property or state information about a *Component* (e.g. check *Component* name, check visibility).

AUT The Application Under Test. The AUT is the entire program that you wish to test.

AUT Configuration The information required in order to locate and start an AUT in the configuration desired by the user. An *AUT Configuration* includes program information such as *JRE* location and class path, and AUT configuration information such as command-line parameters and environment variables. Details about the AUT are given when a project is created, and are entered into the relevant fields in the Test Suite Editor.

CAP The three parts of a Test Step : Component, Action, Parameter.

Classname for AUT The fully qualified classname which contains the main method of the AUT.

Classpath for AUT An environment variable that informs *Java* where to look for the class files which are needed to run the AUT.

Component A user interface object. Each component has a set of actions that can be carried out upon it.

Component Name The symbolic name you choose for a component in the AUT and use in a Test Step or Test Case.

Component Names View In this view, component names in reused Test Cases can be reassigned to create multiple component names for one Test Step. This allows many components of the same type or within the same group or family to be mapped to the same Test Step. The default position of the Component Names View is in the lower right-hand corner of the perspective.

Component Type The type of graphical user interface object which will be acted upon in a Test Step, e.g. "Menu Bar" or "Combo Box".

Data Sets View The view in which multiple parameter values for a Test Case can be defined and edited. The default position of the Data Sets View is the right-hand side of the perspective, in the middle.

Empty string To enter an empty string as a parameter (i.e. nothing at all), use backslash \.

To actually write a backslash, you must enter two backslashes.

Escape character The character used to cancel any special function that the following symbol may have. The default escape character is \.

Event Handling The process by which errors are dealt with during the execution of a test. If no user-defined Event Handler has been specified, a default Event Handler is activated.

Graphical User Interface A means of interacting with a computer by the use of a graphically oriented (as opposed to text-oriented) display. Modern graphical user interfaces (GUI's) are controlled by a mouse as well as a keyboard, and consist of windows, buttons, text areas, and other manipulable objects which are designed to make computer operation simpler and more intuitive.

ITE Short for *Integrated Test Environment*. By this we mean the application where tests are written.

JAR The *Java™* Archive file format. It allows multiple classes and other program files and data to be combined into one compressed file, which often represents a complete program. In this context, a JAR describes the *Java™* Archive of the AUT.

Object mapping The process by which component names you created in your test are attached to the technical names used in the AUT.

Parameters The information that is given to an action to perform a Test Step, such as name, value, etc. A parameter consists of:

- a name
- a type
- a value

Most actions require one or more parameters.

Parameter Name A name which describes a parameter which is sent to an action.

Parameter Type The type of value which is sent as a parameter to an action. The following parameter types are supported:

- String
- Integer
- Boolean

Parameter Value The contents of the parameter which is sent to an action. A parameter value can either be a static value (e.g. `Hello`, `123`), a reference (e.g. `=REFERENCE`) or a variable.

Port Number A software address used to identify and communicate between individual programs or services on a single computer or between multiple computers on a network. A hostname and port number must be given in order to connect to an AUT Agent.

Project A Project contains all the information for the test, including Test Suites and Test Cases, AUT's and data. Projects are stored in the database.

Project Properties Information pertaining to the active project, including the name (all projects in the database must have unique names), the Project language(s), and the data language for the Project.

Propagation When references are entered, the parameter they replace is carried up into the next Test Case in the hierarchy. This is propagation.

Properties View The view for displaying and editing the properties of selected Test Steps or Test Cases, located by default in the top right-hand corner of the perspective. Test Steps and Test Cases can be edited in the Properties View by clicking on their icon in their respective editor. The Properties View is also used to display detailed results of a Test Suite once it has been executed.

Reference Symbol The character used to denote a reference within a parameter. The default reference symbol is =.

Specification The method by which test details and instructions are entered by the user in the ITE. Tests are not *programmed*, but rather *specified* in plain language, then object-mapped to the program's technical names.

Step Delay The time to wait between Test Steps, given in milliseconds. A step delay causes a slower test execution.

Technical Name A name given to a graphical user interface object of an AUT by a software developer. Technical names are used to uniquely identify an object within an application and are mapped to a component name as defined by the user in a Test Step.

Test Case A reusable test object which contains Test Steps or other Test Cases.

Test Case Browser A view which shows the hierarchy of Test Cases as entered and defined by the user. The Test Case Browser is by default in the bottom left-hand corner of the perspective.

Test Case Editor An editor area used to create and alter Test Cases and Test Steps. The Test Case Editor is located by default in the center of the perspective.

Test Data Values which are entered into parameters for an action in Test Steps. Test data may be entered directly into a parameter or referenced.

Test Job A Test Job is a collection of Test Suites which are designed to be executed after each other. The Test Suites can test the samme AUT, different instances of the same AUT, or different AUT's entirely.

Test Result View The view which displays the results of an executed Test Suite.

Test Step The smallest test object. A Test Step belongs to a Test Case. A Test Step consists of

- a **Component**
- an **Action**
- a **Parameter**

Test Suite A Test Suite uses an AUT configuration and contains all Test Cases and their respective Test Steps, in the order in which they are to be tested.

Test Suite Browser The view containing the Test Suite tree, where the Test Cases in a Test Suite can be seen. The Test Suite Browser is situated by default in the top left-hand corner of the perspective.

Test Suite Editor An editor pane used to view the properties of the Test Suite and the Test Cases used in it, and to edit their parameter values and names. The Test Suite Editor is located by default in the center of the perspective.

Workspace Personal preferences relating to the client layout and actions are saved in a Workspace. All changes made in the "*Preferences*" dialog are stored here.

Chapter 9

Index

- Abstract components, 108
- Action, 117, 266
 - Specify, 117
- Action Error, 111, 146, 147
- Activation, 26, 48, 49, 54
- Add
 - AUT Agent, 150
 - AUT Configuration, 48, 49, 54
 - Comments, 63
 - Component Name, 102
 - Event Handler, 146
 - Test Cases, 60
 - Test Suite, 60
- Application activation, 26, 48, 49, 54
- Assigned names, 124
- AUT
 - Configuration, 48, 49, 54
 - Errors, 184
 - Languages, 269
 - RCP, 211
 - Start, 136
 - Stop, 190
- AUT Agent, 21, 262
 - Add, 150
 - Connect, 23
 - Disconnect, 190
 - Embedded, 21, 23
 - Preferences, 151
 - Errors, 183
 - Parameters, 22
 - Preferences, 150
 - Quiet Mode, 22

- Starting, 22
- Stop, 190
- AUT Configuration
 - Add, 48, 49, 54
 - Edit, 48, 49, 54
- AUT ID, 49, 54, 111, 206
- autrun command
 - Define AUT, 52
- Best Practices
 - RCP, 212
- Break, 111, 148
- Browser, 239
 - Component Name, 240
 - Test Case, 240, 248
 - Test Suite, 240, 248
- CAP, 117, 266
- Categories
 - Object Mapping, 124
 - Test Cases, 68
 - Test Jobs, 68
 - Test Suites, 68
- Central test data
 - Editor, 251
- Change
 - Database, 27
- Changes in the AUT, 134
- Check failed, 111, 147
- Classname, 49
- Classpath, 49
- Cleanup
 - Object Mapping Editor, 129
- Client, 261
- CmdLineParameter, 49
- Comment
 - Add, 63
- Component, 103, 117, 266
 - Names, 270
 - Delete, 107
 - Global, 103
 - Hierarchy, 108
 - Local, 103
 - Merge, 106
 - New, 102
 - Reassigning, 103

- Rename, 106
- View, 103, 240, 251
- Type, 117
- Component Name
 - Delete, 58
 - Renaming, 58
- Component name
 - Show where used, 178
- Component Name Browser, 240
- Component not found, 111, 146, 147
- Concatenation, 88
- Concrete Value, 80
- Configuration
 - AUT, 48, 49, 54
 - Test Suite, 111
- Configuration error, 111, 146, 147
- Connect
 - AUT Agent, 23
- Console
 - View, 251
- Continue, 111, 148
- Continue without Event Handler, 139
- Create
 - Project, 30
- Data Sets View, 97, 240, 251
- Database, 262
 - Change, 27
 - Log-in, 26
 - Preferences, 151
 - Select, 27
- dbtool, 198
- Default
 - Language, 30
- Default Event Handler, 111
- Default Language, 269
- Define AUT
 - autrun command, 52
- Delete
 - Component Name, 58
 - Component Names, 107
 - Test Case, 58
 - Test Job, 58
 - Test Steps, 61
 - Test Suite, 58
- Delete Project, 39

- Design For Testability
 - Web, 220
- Design for Testability
 - RCP, 212
 - Swing, 210
- Disconnect AUT Agent, 190
- Edit
 - AUT Configuration, 48, 49, 54
 - Project Properties, 33
 - Test Case, 61, 77
 - Test Job, 61
 - Test Steps, 119
 - Test Suite, 61
- Editor, 239
 - Central test data, 251
 - Object Mapping, 251
 - Preferences, 152
 - Test Case, 250
 - Test Suite, 110, 115, 250
 - View, 240, 251
- Embedded AUT Agent, 23
 - Preferences, 151
- Empty String, 99
- Environment, 49
- Environment variables, 84
- Error Messages, 164
- Error Types, 111, 146, 147
- Errors
 - AUT, 184
 - AUT Agent, 183
 - Test Suite not executable, 186
- Escape Character, 99
- Event Handler, 141, 273
 - Add, 146
 - Default, 111
 - Error Types, 111, 146, 147
 - Reentry Properties, 111, 148
- Excel data, 94
- Excel File Configuration, 95
- Excel TODAY function, 97
- Execution
 - Perspective, 175, 242
 - Test
 - Pause, 138
 - Prepare, 136

- Start, 137
- Stop, 138
- Exit, 111, 148
- Export
 - All, 42
 - Project, 42
- Extensibility, 274
- Extracting
 - Test Case, 64
- Find, 181
- Functions, 85
 - Syntax, 86
- GEF, 213
 - Accessing Figures, 214
 - Inspector, 213
- Global Component Names, 103
- Heuristic, 126
- Hierarchy of Component Names, 108
- Highlight in AUT, 128
- ID attribute name, 54, 220
- Image
 - View, 251
- Import
 - Project, 41
- Information Messages, 164
- Inspector
 - GEF, 213
- Inspector View, 251
- Integrated Test Environment, 261
- ITE, 261
 - Starting, 25
- JRE parameters, 84
- Keyboard Layout, 211
- Language
 - Default, 30
- Languages
 - AUT, 269
 - Project, 30, 269
 - Working, 269
- Launch Configurations, 205
- Library of Test Cases, 73

- Local Component Names, 103
- Locked
 - Test Case, 188, 263
- Log-in
 - Database, 26
- Maintainability, 270
- Merge Component Names, 106
- Messages
 - Error, 164
 - Information, 164
 - Warning, 164
- Names
 - Component, 270
 - Reassigning, 103
 - Technical, 270
- Navigator View, 246, 251
- New
 - Component Name, 102
 - Project, 30
 - Test Case, 71
 - Test Job, 115
 - Test Step, 117
 - Test Suite, 110
- Object Mapping, 122, 270
 - Assigned names, 124
 - Categories, 124
 - Collect components, 131
 - Editor, 251
 - Cleanup, 129
 - Refresh, 128
 - Highlight in AUT, 128
 - in categories, 125
 - Map components, 133
 - Preferences, 126, 152
 - Unassigned component names, 124
 - Unassigned technical names, 124
- objectmapping, 185
- Observation, 272
- Observation Mode, 159
 - Preferences, 153
 - Start, 159
 - Tips, 159
- Observing

- Test Steps, 160
- Open Project, 38
- Overwrite data, 100
- Parameter, 80, 117, 266
 - Concatenation, 88
 - Concrete Value, 80
 - Data Set, 97
 - Empty String, 99
 - Escape Character, 99
 - Excel file, 94
 - Function, 85
 - JRE, 84
 - Overwrite, 100
 - Reference, 81
 - Variable, 83
- Parameters
 - Delete, 82
 - Edit, 82
- Pause, 111, 148
 - Test Job, 138
 - Test Suite, 138
- Pause on Error, 139
- Pause Test Execution, 138
- Perspective, 239
 - Execution, 175, 242
 - Reporting, 244
 - Specification, 175, 240
 - Workspace, 246
- Predefined Variables, 85
- Preferences
 - AUT Agent, 150
 - Database connection, 151
 - Editor, 152
 - Embedded AUT Agent, 151
 - Object Mapping, 126, 152
 - Observation Mode, 153
 - Reset, 175
 - Test, 149
 - Test Result, 153
- Problem View, 164, 240, 251
- Progress
 - View, 251
- Project, 267
 - Create, 30
 - Delete, 39

- Export, 42
- Import, 41
- Languages, 30, 269
- New, 30
- Open, 38
- Properties, 33
- Refresh, 39
- Save As, 40
- Version, 44
- Properties View, 240, 251
- RCP, 211
 - Best Practices, 212
 - Design For Testability, 212
 - Generate Technical Names, 212
 - Remote Control, 211
 - Set Data, 212
- RCP AUT's, 211
- rdesktop, 188
- Reassigning
 - Component Names, 103
- Reentry Properties, 111, 148
- References, 81
- Refresh
 - Object Mapping, 128
- Refresh Project, 39
- Rename
 - Component Name, 106
 - Referenced Test Case, 62
 - Referenced Test Suite, 62
- Renaming
 - Component Names, 58
 - Test Cases, 58
 - Test Jobs, 58
 - Test Suites, 58
- Replace
 - Test Case, 65
- Reporting
 - Perspective, 244
- Results
 - View, 141
 - XML and HTML Reports, 142, 153
- Retry, 148
- Return, 111, 148
- Revert changes, 68
- Running AUT's View, 251

- Save As
 - New Test Case, 67
- Save Project As, 40
- Search, 181
 - Component in AUT, 128
 - Components in Test Cases, 128
- Search Result View, 251
- Select
 - Database, 27
- Server, 261
- Set Data, 212
- Set Name, 210
- Show specification
 - Test Cases, 177
 - Test Suites, 177
- Show where used
 - Component name, 178
 - Test Cases, 177
- Specification
 - Perspective, 175, 240
- Speed of test execution, 140
- Start
 - AUT, 136
 - AUT Agent, 22
 - ITE, 25
 - Observation Mode, 159
 - Test Execution, 137
 - Test Job, 137
 - Test Suite, 137
- Status Bar, 257
- Stop
 - AUT, 190
 - AUT Agent, 190
 - Test Execution, 138
 - Test Job, 138
 - Test Suite, 138
- Storing Variables, 83
- Support Package, 188
- Swing
 - Design For Testability, 210
 - Set Name, 210
- System variables, 84
- Technical Names, 270
- Test
 - Execution

- Prepare, 136
- Pause, 138
- Preferences, 149
- Result View, 251
- Results, 141
- Start, 137
- Stop, 138
- Test Case, 267
 - Add, 60
 - Browser, 240
 - Categories, 68
 - Delete, 58
 - Edit, 61, 77
 - Editor, 250
 - Extracting, 64
 - Library, 73
 - Locked, 188, 263
 - New, 71
 - Renaming, 58
 - Replace, 65
 - Save As, 67
 - Show specification, 177
 - Show where used, 177
- Test data, 80
 - Editor, 251
- Test execution
 - Speed, 140
- Test Executor, 192, 196
- Test Job, 113–115, 267
 - Categories, 68
 - Delete, 58
 - Edit, 61
 - Pause, 138
 - Renaming, 58
 - Start, 137
 - Stop, 138
- Test Result
 - Preferences, 153
 - View, 242
- Test Result Summary
 - View, 244
- Test Result Summary View, 251
- Test Step, 266
 - Delete, 61
 - Edit, 119

- New, 117
- Observe, 160
- Test Suite, 110, 267
 - Add, 60
 - Browser, 240
 - Categories, 68
 - Configuration, 111
 - Delete, 58
 - Edit, 61
 - Editor, 110, 115, 250
 - Failed, 187
 - New, 110
 - not executable, 186
 - Pause, 138
 - Renaming, 58
 - Show specification, 177
 - Start, 137
 - Stop, 138
- Translating data, 97
- Troubleshooting, 183
- Typesetting Conventions, 12
- Unassigned names
 - Component, 124
 - Technical, 124
- User Interface
 - Adapting, 175
- Variables, 83
 - Environment, 84
 - Pre-defined, 85
 - Storing, 83
 - System, 84
- Verify failed, 146
- Versioning Projects, 44
- View, 239, 251
 - Component Names, 103, 240, 251
 - Console, 251
 - Data Sets, 240, 251
 - Editor, 240, 251
 - Image, 251
 - Inspector, 251
 - Navigator, 251
 - Problem, 164, 240, 251
 - Progress, 251
 - Properties, 240, 251

- Results, 141
- Running AUT's, 251
- Search Result, 251
- Show, 175
- Test Result, 251
- Test Result Summary, 251
- Views
 - Navigator, 246
 - Test Result, 242
 - Test Result Summary, 244
- Warning Messages, 164
- Web
 - Design For Testability, 220
- Working Directory, 49
- Working language, 269
- Workspace
 - Perspective, 246
- XML and HTML Result Reports, 142, 153