Testing Tech
TTworkbench User's Guide

User's Guide

Testing Tech TTworkbench User's Guide: User's Guide

This document is subject to change without notice.

Testing Technologies IST GmbH
Michaelkirchstrasse 17/18
10179 Berlin
Germany

phone	+49 30 726 19 19 0
fax	+49 30 726 19 19 20
internet	www.testingtech.com

Note

Individual copies of the present document can be downloaded from www.testingtech.com .

The present document may be made available in more than one electronic version or in print. In any case of existing or perceived difference in contents between such versions, the reference version is the HTML Format.

Users of the present document should be aware that the document might be subject to revision without notice. If you find errors in the present document, please use the ticket system at support.testingtech.com and report it.

Published Published on 2009-06-03

Testing Technologies TTworkbench is powered by Eclipse technology and includes Eclipse plug-ins that can be installed and used with other Eclipse 3.3-based offerings.

It includes software developed by the Apache Software Foundation (http://www.apache.org/), ANTLR (http://www.antlr.org/), Tigris.org (http://gef.tigris.org/), and L2FProd.com (http://www.L2FProd.com/).

Table of Contents

1. Introduction 1
    How this Manual is Organized 1
    Related Documentation 1
2. Installation 3
    Requirements 3
        Eclipse and Java 3
        License Management 3
    Installer Distribution 3
        Perform the Installation 3
        Installed Files 4
        Uninstall TTworkbench 4
    Update Site Distribution 4
        Requirements 4
        Perform the Installation 5
        Uninstall TTworkbench features 5
    License to Enable TTworkbench 6
    Workspace, Temporal Data and Preferences 6
3. Global TTCN-3 Preferences 7
    The TTCN-3 Preferences Page 7
        TTCN-3 Edition 7
        TT3 plugins 8
        Java Compiler 8
        Reporting 9
4. TTCN-3 Project 10
    Creating a TTCN-3 Project 10
    Project Properties 13
        General 13
        TTCN-3 Sources 15
        Output 18
        Environment 18
        Compiler settings 19
        TT3 Plugins 20
5. Using TTworkbench CL Editor 22
    Properties 22
    Development Perspective 23
        Overview 23
        TTCN-3 Projects navigator view 23
    Editing a Test Suite 27
        Introduction 27
        Using Core Language Viewer (CLViewer) 29
        Outline of Test Suite Structure 30
        Quick Outline 32
        TTCN-3 Search 33
        Find references in workspace 34
        Outline view components 35
        Folding support for template fields 35
        Highlighting of Keywords 36
        Comparing TTCN-3 files 36
        Mark Occurrences 36
        Validation of Syntax 37
        Formatting of Text 38

Open Declaration 38
Completion Assistance 39
Progress Information 40
Code Folding 40
Refactoring Support 41
Quick Fix 43
Showing single element or whole TTCN-3 file 43
Task tags inside TTCN-3 comments 44
Template Wizard 44
Preferences 49
General 49
Syntax Preference 50
Typing Preference 51
Folding Preference 52
Mark Occurrences 53
TT3 Plugins Preference 54
Template Preference 55
6. Using TTworkbench GFT Editor 56
GFT Basics 56
Language Concepts 56
Mapping between GFT and TTCN-3 Core Language 56
Module Structure 57
GFT Editor UI and Workflow 58
About the GFT Editor Working Environment 58
Editing Symbol Attributes 60
Using Context Menus 62
Showing/Hiding Optional Symbol Attributes 62
Go to/Import Referenced Diagram 63
Undoing and Repeating Multiple Actions 63
To Do Support 63
Importing Definitions 64
Importing TTCN-3 Core Language 67
Exporting TTCN-3 Core Language 68
Exporting GIF 68
Opening and Saving a GFT File 68
GFT Diagrams 68
Overview 68
Control Diagram 71
Test Case Diagram 71
Function Diagram 72
Altstep Diagram 74
GFT Symbols 75
Instances 75
Actions 78
Comments 79
Execute Test Cases 80
References 81
Labels and Goto 81
Inline Expressions 82
Defaults 93
Creates 93
Start Components 94
Conditions 94
Messages 99

Timers 105
Text Symbols 108
GFT Example 109
Control Diagram 110
Invoking Functions 112
Main Test Case 115
Functions and Altsteps 120
Save TTCN-3 and GIF 123
7. Using TTworkbench TTthree 124
Properties 124
Preferences 127
General Settings 127
Code Generation 128
Perform the Compilation 130
Command-line Mode 133
Batch Compiler 133
Advanced Batch-Compiler for Linux 138
TTthree Server 139
Starting 139
Command line options 140
TTCN-3 Documentation Generation (T3Doc) 140
Generate HTML Documentation 140
The Text of a Documentation Comment 141
General Description 141
Tagged Paragraphs 141
8. Using TTworkbench TTman 149
Overview 149
Using TTman 151
Test Campaign 151
Management View 169
Meta Campaign View 180
Parameters View 181
Properties View 183
Textual Logging View 183
Graphical Logging View 185
Data View 189
Dump View 190
Preferences 192
TTman 192
Logging 195
Report 198
E-mailing 199
Command-line Mode 201
9. Using TTworkbench TTdebug 204
Overview 204
How to start debugging a test suite 205
Breakpoints 206
Setting breakpoints 206
Temporarily disabling breakpoints 207
Setting a breakpoint hit count 207
Debugging TTCN-3 208
Debug Perspective 208
Debug View 209
Breakpoints view 211

Variables View 211
Timers View 214
Port Queue View 214
Debugging multiple components 215
Automatic perspective switch 215
Debugging Java 216
Handling the SUT 216
Preferences 216
10. Additional Runtime Plugins 217
11. Frequently Asked Questions 218
12. What's New 225
13. Contacting Technical Support 226
Testing Technologies Technical Support Contact Information 226
Index 227

List of Figures

3.1. TTCN-3 main preferences 7
3.2. TT3Plugins preferences: extensions of the TTCN-3 compiler 8
3.3. Java compiler preferences 9
3.4. Reporting preferences 9
4.1. Creating a new TTCN-3 project 10
4.2. Java properties of new TTCN-3 project 11
4.3. TTCN-3 properties of new TTCN-3 project 12
4.4. Adding TTCN-3 compiler nature 13
4.5. General settings page 14
4.6. Builders properties of a TTCN-3 project featuring the TTCN-3 compiler nature 15
4.7. TTCN-3 Sources settings page 16
4.8. TTCN-3 Sources menu actions in the TTCN-3 Projects navigator view 17
4.9. Output folder setting page 18
4.10. Environment variables setting page 19
4.11. Compiler settings page 20
4.12. Project specific TT3 Plugins page 21
5.1. TTworkbench CL Editor 22
5.2. Switching layout mode in the TTCN-3 Projects navigator view 24
5.3. Flat layout mode of the TTCN-3 Projects navigator view 25
5.4. Combined layout mode of the TTCN-3 Projects navigator view 26
5.5. Hierarchical layout mode of the TTCN-3 Projects navigator view 27
5.6. New module wizard 28
5.7. New module in editor 29
5.8. Editable and non-editable generated TTCN-3 files in the navigator 30
5.9. Outline showing definitions as found in module 31
5.10. Outline with definitions grouped by category 32
5.11. Quick outline 33
5.12. TTCN-3 search 34
5.13. Find references in workspace 35
5.14. Comparing a local TTCN-3 file with a revision from source code management 36
5.15. Validate command 37
5.16. Error reporting 38
5.17. Open declaration 39
5.18. Completion assistance 40
5.19. Code folding 41
5.20. Parameters Page for the Rename Refactoring Command 42
5.21. Quick Fix 43
5.22. Show Source of Selected Element Only 44
5.23. Template in Core Language 45
5.24. Edit page of the Template wizard 46
5.25. Preview page of the Template wizard 47
5.26. Content Assist for From Type in the Template wizard 48
5.27. CL Editor General preference page 50
5.28. CL Editor Syntax preference page 51
5.29. CL Editor Typing preference page 52
5.30. Folding preference page 53
5.31. Mark Occurrences preference page 54
5.32. CL Editor TT3 plugins preference page 54
5.33. CL Editor Template preference page 55
6.1. Relation between TTCN-3 Core Language and GFT 57
6.2. TTCN-3 Module Tree Structure in GFT Editor 58

6.3. Core Language Tab 59
6.4. Tools Palette 60
6.5. Tool Tips on Symbol Attributes 60
6.6. List of Attributes 61
6.7. Property Panel 61
6.8. Symbol's Context Menu 62
6.9. Show / Hide Optional Symbol Attributes 62
6.10. To Do Item Panel 63
6.11. Choosing Multiple TTCN-3 Module Files for Import. 64
6.12. Property Panel of an Import Statement 65
6.13. Choosing the Definitions to be Imported 66
6.14. Importing TTCN-3 Core Language in GFT Editor 67
6.15. Sequential Behavior 69
6.16. Diagram Type Selection 70
6.17. Target Group Selection 70
6.18. Control Diagram 71
6.19. Test Case Diagram 72
6.20. Function Diagram 73
6.21. Add a Return Symbol to a Function Component Instance 74
6.22. Altstep Diagram 75
6.23. Control Instance Symbol 76
6.24. Test Component Instance in Test Cases 77
6.25. Test Component Instance Symbol in Functions and Altsteps 77
6.26. Changing the Type of Test Components 78
6.27. Port Instance Symbol 78
6.28. Action Symbol 79
6.29. Event Comment Symbol 80
6.30. Text Symbol Comment 80
6.31. Comment in Property Panel 80
6.32. Execute Symbol 81
6.33. Function Reference 81
6.34. Label and Goto 82
6.35. Setting the Goto Type 82
6.36. Type Change of an Inline Expression 83
6.37. Change the Number of Alternatives 83
6.38. If 84
6.39. If-Else 84
6.40. For 84
6.41. While 85
6.42. Do-While 85
6.43. Alt 86
6.44. Alt With Altstep Invocation 86
6.45. Selecting/Deselecting an Alt 87
6.46. Alt With Else Branch 88
6.47. Change Operation in Alternative 89
6.48. Associate a Return Statement with a Return Value 90
6.49. Stop Execution Operation 91
6.50. Interleave 92
6.51. Call 92
6.52. Activate 93
6.53. Deactivate 93
6.54. Create 93
6.55. Start 94
6.56. Modify the Condition Type 94

6.57. Disable/Enable an Alternative 95
6.58. Else Branch of an Alternative 95
6.59. Popup Menu for Verdict Setting 96
6.60. Popup Menu for Port Operation Setting 97
6.61. Done 99
6.62. Send Operation 101
6.63. Receive Operation 102
6.64. Receive Any Message Operation 102
6.65. Receive on Any Port 102
6.66. Connected Start Timer Stop/Timeout Timer Operation 106
6.67. Start Timer Operation 108
6.68. Stop Timer Operation 108
6.69. Timeout Timer Operation 108
6.70. Control 110
6.71. Control Instance 111
6.72. Declaration 111
6.73. Complete Control Diagram 112
6.74. Property Panel of a Function 113
6.75. Return Statement of an Instance 114
6.76. Function basicCapabilityTests 115
6.77. Test Component Instance and Port Instances 116
6.78. Property Panel of an Activation 117
6.79. Nested Inline Expressions 118
6.80. Message Value Assignment 118
6.81. Actions 119
6.82. If Inline Expressions 119
6.83. Add a Stop Symbol to a Component Instance 120
6.84. Procedure-based Communication 122
6.85. Altstep 123
7.1. TTworkbench TTthree 124
7.2. General preference page 128
7.3. Code generation preference page 129
7.4. Compilation progress 130
7.5. Compilation successful 131
7.6. Generated jar file 131
7.7. Compilation failed 132
7.8. Problems found 132
8.1. Overview of TTman 149
8.2. Open the TTCN-3 execution management perspective 150
8.3. Open the TTCN-3 execution management perspective with right click 151
8.4. Generating the Default Campain 153
8.5. Starting the test campaign wizard 154
8.6. The test campaign wizard (first page) 155
8.7. Selecting test cases in the test campaign wizard 156
8.8. Setting test case properties in the test campaign wizard 157
8.9. Setting the test adapter in the test campaign wizard 158
8.10. Starting the meta campaign wizard 159
8.11. The meta campaign wizard (first page) 160
8.12. Selecting campaigns in the meta campaign wizard 161
8.13. Setting test campaign properties in the meta campaign wizard 162
8.14. Setting the test adapter in the Management View 163
8.15. Selecting the test adapter 164
8.16. Configuring the codec plugins 165
8.17. Configuring the external functions plugins 166

8.18. Configuring the port plugins 167
8.19. Configuring the test adapter parameters 168
8.20. Management view 169
8.21. Verdict Filter 170
8.22. Importing a test campaign 171
8.23. Start a test case from the menu bar 172
8.24. Start a test case from the context menu 172
8.25. Importing a Test Session 173
8.26. Selecting the data source for the Test Report 175
8.27. Setting test report type and destination 176
8.28. Setting filtering and sorting options 177
8.29. Tester Properties 178
8.30. Test Report Example 179
8.31. Meta Campaign view 180
8.32. Importing a meta campaign 181
8.33. Parameters view 182
8.34. Module Parameter Editor 182
8.35. Properties view 183
8.36. Logging View 184
8.37. View TTCN-3 Source 185
8.38. Graphical Logging View 186
8.39. Data View 189
8.40. The received message does not match the given template 190
8.41. Dump view interpreting the input as text 190
8.42. Dump view interpreting the input as hex 191
8.43. The Dump view interpreting the input as formatted XML with highlighting. 192
8.44. Complete test case execution is set by default 193
8.45. Error debug level is set by default 194
8.46. An error displayed in the output Console 194
8.47. Online logging is supported by default 195
8.48. TTman preferences logging generation 196
8.49. TTman preferences for textual logging 197
8.50. TTman appearance settings 198
8.51. User and test dependent information for test reports 199
8.52. Mail Settings 200
8.53. Server and Account Settings 201
9.1. TTdebug 204
9.2. Debug button 205
9.3. Toggling a breakpoint by using the context menu of the vertical ruler 206
9.4. Disabling a breakpoint by using the context menu of the vertical ruler 207
9.5. Breakpoint Properties Page 208
9.6. Eclipse Debug Perspective 209
9.7. Perspective bar 209
9.8. Debug View 210
9.9. Breakpoints View 211
9.10. Variables View 212
9.11. Open the column configuration 213
9.12. Editing a basic value in the Variables View 213
9.13. Timers View 214
9.14. Port Queue View 215
11.1. Update TTtools path 224

List of Tables

7.1. T3Doc Tags 142

Chapter 1. Introduction

The TTworkbench is a TTCN-3 IDE (Integrated Development Environment) which provides various capabilities based on the Eclipse platform. It supports a broad spectrum of test development, ranging from the specification to the compilation and the execution of tests. The "Testing Tech TTworkbench User's Guide, User's Guide" supplies usage and reference information for TTworkbench. This manual is designed for all users of TTworkbench.

To use TTworkbench efficiently, you should be familiar with TTCN-3 and with the Eclipse platform.

How this Manual is Organized

Installation	Gives you detailed instructions how to install TTworkbench on your system.
Using TTworkbench CL Editor	Introduces the usage of CL Editor, the first part of the TTworkbench.
Using TTworkbench GFT Editor	Introduces the usage of GFT Editor, the second part of the TTworkbench.
Using TTworkbench TTthree	Introduces the usage of TTthree, the third part of the TTworkbench.
Using TTworkbench TTman	Introduces the usage of TTman, the fourth part of the TTworkbench.
Frequently Asked Questions	Provides answers to some questions frequently asked by the customers.
Contacting Technical Support	Collects contact information for the technical support.

Related Documentation

After installation and before you begin using TTworkbench, please review any `README` files and Release Notes to ensure that you have the latest information about the product.

For additional resources on TTCN-3 and Eclipse, refer to the following publications:

- ES 201 873-1 TTCN-3 Core Language Version: 3.1.1
- ES 201 873-5 TTCN-3 Runtime Interface Version: 3.1.1
- ES 201 873-6 TTCN-3 Control Interface, Version: 3.1.1
- ETSI Protocol & Testing Competence Centre: www.etsi.org/ptcc
- Abstract Syntax Notation One (ASN.1): Specification of Basic Notation ITU-T Rec. X.680 (2002) | ISO/IEC 8824-1:2002
- ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER) ITU-T Rec. X.690 (2002) | ISO/IEC 8825-1:2002, Specification of Packed Encoding Rules (PER) ITU-T Rec. X.691 (2002) | ISO/IEC 8825-2:2002
- eclipse.org: www.eclipse.org

- Eclipse Modeling Framework: www.eclipse.org/emf
- Eclipse Graphical Editing Framework: www.eclipse.org/gef

Chapter 2. Installation

Requirements

Eclipse and Java

TTworkbench is based on the Eclipse 3.3.1.1 platform and runs on a machine with Windows Vista, Windows XP, Windows 2000, Red Hat Linux at least version 7.1 (x86/GTK) or SuSE Linux at least version 9.1 (x86/GTK) or higher with at least *1 GB* RAM. The Eclipse Foundation claims: “Although untested, Eclipse should work fine on other OSes that support the same window system.” This is also true for TTworkbench, which may perform well on other platforms. However, Testing Technologies tests and supports running TTworkbench only on platforms listed above.

A comprehensive list of the supported OS and Java environments can be found in the Eclipse 3.3.1.1 ReadMe file at http://www.eclipse.org/eclipse/development/readme_eclipse_3.3.1.1.html#TargetOperatingEnvironments

TTworkbench needs a Java Platform, Standard Edition (Java SE, formerly known as J2SE) version 5.0. This version supersedes J2SE 1.4.2 and is also known as version 1.5.

Java comes in two flavors, as J2SE Runtime Environment (JRE 5.0) for executing Java applications and as J2SE Development Kit (JDK 5.0) with tools for Java application development added. Java can be downloaded from http://java.sun.com or http://www.ibm.com/developerworks/java/jdk/.

TTworkbench is distributed as an installer downloadable from our website. Only the aforementioned JDK or JRE is required to install TTworkbench on your machine. See the section called “Installer Distribution”.

Another way of obtaining TTworkbench is to download and install using Eclipse's built-in Update Manager. See the section called “Update Site Distribution”.

License Management

TTworkbench uses the FLEXnet Publisher license management of Macrovision. It is possible to download the lmgrd license daemon version 10.1 or higher from http://www.macrovision.com/services/support/fnp_utilities.shtml . An end user's guide can be found at http://www.macrovision.com/services/support/enduser.pdf.

Please contact Testing Technologies Sales Department to get a valid license file sales@testingtech.com.

Installer Distribution

Perform the Installation

After a JDK or JRE has been installed, run the TTworkbench installer. For Linux platforms it is distributed as a java archive with a `jar` filename extension. Start it by executing **java -jar TTworkbench(Express/Basic/Professional)-v(version)-installer.jar** on the command line. For MS Windows computers the installer is distributed as an executable program file (name extension is `exe`).

Note

For updates of an existing TTworkbench installation

it is recommended to uninstall the old version before installing the current.

You will be guided through the installation. Read the accompanying information texts and choose the directory where TTworkbench shall be installed. TTworkbench should be installed into an empty folder to avoid overwriting important files accidentally. The installation wizard offers to copy a license file you select to TTworkbench's installation folder. You can skip this step at this point if you want to copy it there after installation, see the section called “License to Enable TTworkbench”.

Note

Existing files in the destination folder will be overwritten!

Most subsequent updates of TTworkbench can be performed via the built-in Update Manager, described in the section called “Perform the Installation” unless Testing Technologies recommends otherwise. When after connecting Testing Technologies's update site the Update Manager requires newer Eclipse components, the safest bet is to use the installer version of TTworkbench.

Installed Files

The installer places the TTworkbench application, an uninstaller program and licensing information into the chosen directory. Nothing is changed, added or removed outside this location with the exception of shortcuts.

A shortcut to TTworkbench is placed on the desktop and in the start menu (or K menu on Linux with the KDE desktop environment). The start menu entry will also contain a shortcut to the uninstaller. On Linux systems the installation directory contains a shell script to start up TTworkbench.

Uninstall TTworkbench

A shortcut to the uninstaller can be found in the start menu. This uninstaller removes all installed files, but will not touch files and directories created after installation. It is an executable jar-file

Because Eclipse workbench stores configuration files inside its application directory, those files will remain after uninstallation. Usually you will not need them, so you can remove them too by enabling the option Force the deletion of.... Please double check the directory before using this switch. If you would like to uninstall TTworkbench without the uninstaller you can delete the installation directory and remove created shortcuts.

If an installed TTworkbench has been moved after installation, the uninstaller will not work. To uninstall, remove the installation directory manually. Additionally remove shortcuts on desktop and in start menu.

Do not just disable TTworkbench features from eclipse as described in the section called “Uninstall TTworkbench features”, because your TTworkbench installation would not start afterwards.

Update Site Distribution

Requirements

As a second option TTworkbench can be installed onto an existing Eclipse workbench via Testing Technologies' update site. So, first a JDK 5.0 and the Eclipse 3.3.1.1 platform have to be installed on your system. Download Eclipse from eclipse.org at www.eclipse.org/downloads. TTworkbench requires Release Build *3.3.1.1*. You should choose the complete Eclipse SDK. It contains the Eclipse Java Development Environment, which is needed to get the most out of TTworkbench.

Additionally some Eclipse components have to be installed:

- EMF can be downloaded from its home site eclipse.org at download.eclipse.org/tools/emf/scripts/downloads.php . Here, the package *EMF and SDO RT* is sufficient.
- GEF can be downloaded also from eclipse.org at download.eclipse.org/tools/gef/downloads/
- BIRT (downloadable from eclipse.org at http://www.eclipse.org/birt/phoenix/)
- WTP (downloadable from eclipse.org at http://www.eclipse.org/webtools/)
- DTP (downloadable from eclipse.org at http://www.eclipse.org/datatools/)

Perform the Installation

Open Eclipse Workbench and create an update-site entry with the URL which you obtained from Testing Technologies. In order to do this, navigate menu Software Updates, Find and Install... in the Help menu. In the dialog box appearing choose Search for new features to install , then Next.... Here you can add an entry for the Testing Technologies update-site with New Remote Site... . Choose a name of your liking.

When accessing Testing Technologies' update-site a user name and a password is required. Enter those values you've received from our customer support. Then you can search for the TTworkbench feature on the newly created update-site. When a list of features to install is shown, select Testing Tech TTworkbench and install it. Restart the workbench as requested. After installation, please enable TTworkbench with a license: see the section called “License to Enable TTworkbench”.

Note

When after connecting Testing Technologies's update site the Update Manager requires newer Eclipse components, the safest bet is to use the installer version of TTworkbench.

Because the Update Manager does only update features and plugins an additional step is necessary.

Note

For the next step, please close TTworkbench or Eclipse respectively.

The directory `(TTworkbench or Eclipse install loc)/plug-ins/com.testingtech.ttworkbench.execpack_x.x.x/` contains zip files with updated versions of the TTworkbench executable and scripts for command line mode. Use the latest version of this directory, which should have been installed in the previous step. Please extract the zip file according to your operating system (`starter_linux_gtk.zip` for Linux or `starter_win32.zip` for Windows) into the installation directory of TTworkbench. Existing executables and scripts can safely be overwritten.

Uninstall TTworkbench features

A feature in Eclipse can be disabled and uninstalled via the integrated configuration management system. In the Help menu, navigate submenu Software Updates and choose Manage Configuration. A dialog labeled Product Configuration appears. Here, please select the TTworkbench feature and choose Disable or Uninstall in the pane on the right side and restart the workbench.

This procedure must not be performed for the *main feature* of the running Eclipse workbench installation. Eclipse would not start if the main (or branding) feature is disabled.

Note

In TTworkbench installed using the installer, the branding feature is one of TTworkbench-Basic, -Professional or -Enterprise. These must not be disabled -- the whole installation would cease to work otherwise.

To un-install TTworkbench simply delete the installed features, plugins, the license file and the temporal data as denoted above.

License to Enable TTworkbench

After installation and before effectively using TTworkbench, be sure to have the license file `license.dat` exactly in the directory, where the TTworkbench executable has been installed, e.g. `C:\Program Files\TTworkbench\` or `/usr/local/TTworkbench/` or similar according to your environment.

Without a valid license, Eclipse features from Testing Technologies refuse to start. The licensing mechanism will shut down the workbench after a number of unsuccessful attempts to find a valid license.

Workspace, Temporal Data and Preferences

When starting up, TTworkbench asks for a place to store project data and preferences. This place is called the *workspace*. Only one workspace can be active at a time, but you can switch workspaces to organize your work. It is highly recommended to store your workspace(s) *outside the TTworkbench installation directory*.

Besides files and projects you work with, TTworkbench creates temporal data in your workspace under `.metadata/.plugins/` in directories of the form `com.testingtech.ttworkbench.*/` .

Preferences are stored compliant to the eclipse platform in your workspace under `.metadata/.plugins/org.eclipse.core.runtime/.settings/` in files of the form `com.testingtech.ttworkbench.*.prefs` .

The TTCN-3 compiler TTthree uses disk space temporarily during compile runs. For this purpose it creates a directory named `TTthree-$USER/` in the current system's default temporal data directory.

Chapter 3. Global TTCN-3 Preferences

This chapter explains the global TTCN-3 preferences.

The TTCN-3 Preferences Page

When you choose Window -> Preferences from the menu, you will find a TTCN-3 tab selectable from the left side menu.

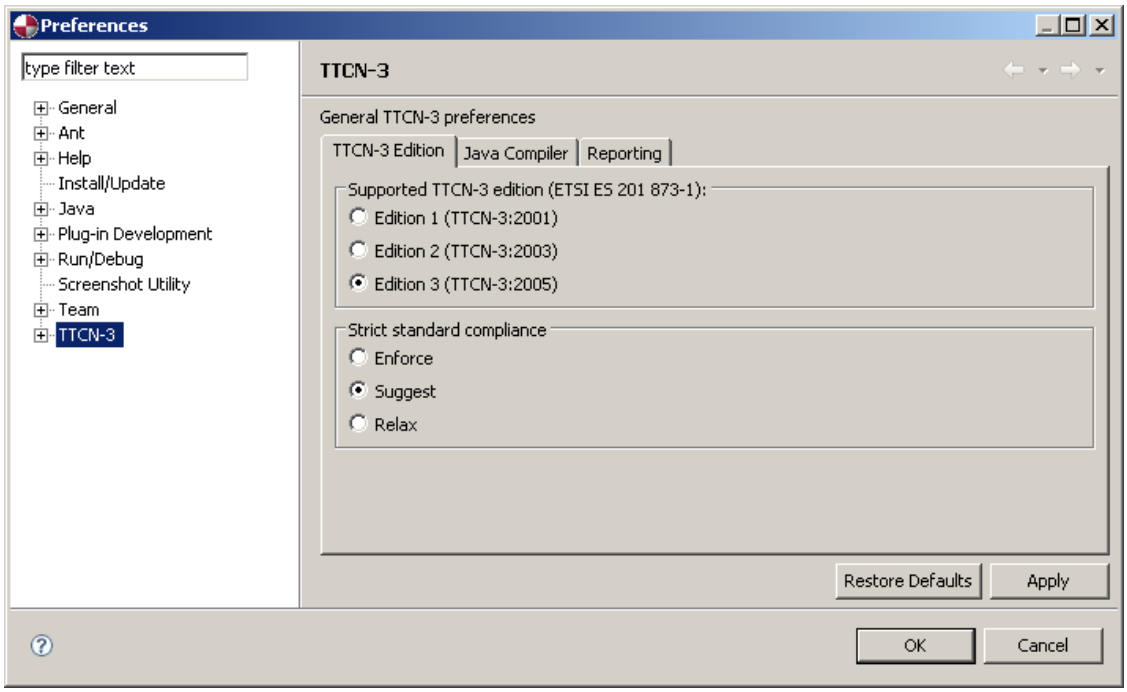
Preferences directly included in the container preference category TTCN-3 are global preferences used by all TTworkbench features.

TTCN-3 Edition

The preference page titled TTCN-3 Edition as shown by Figure 3.1, “TTCN-3 main preferences” provides the selection of supported TTCN-3 editions as defined in ETSI ES 201 873-1 documents. The editions 1, 2 and 3 are supported currently. They are also indicated by the language strings TTCN-3:2001, TTCN-3:2003, TTCN-3:2005 and TTCN-3:2008, respectively. The default TTCN-3 edition in this TTworkbench is edition 3.

Please note, the language declaration in the current module, if present, determines always the final setting for the TTCN-3 edition. For example, by global setting for TTCN-3 edition 3, a module with the declaration **language "TTCN-3:2003"** will be treated as a TTCN-3 edition 2 module. If the module language declaration is absent, the global setting will be applied. Please refer to the description of TTworkbench features for specific options related to TTCN-3 editions.

Figure 3.1. TTCN-3 main preferences



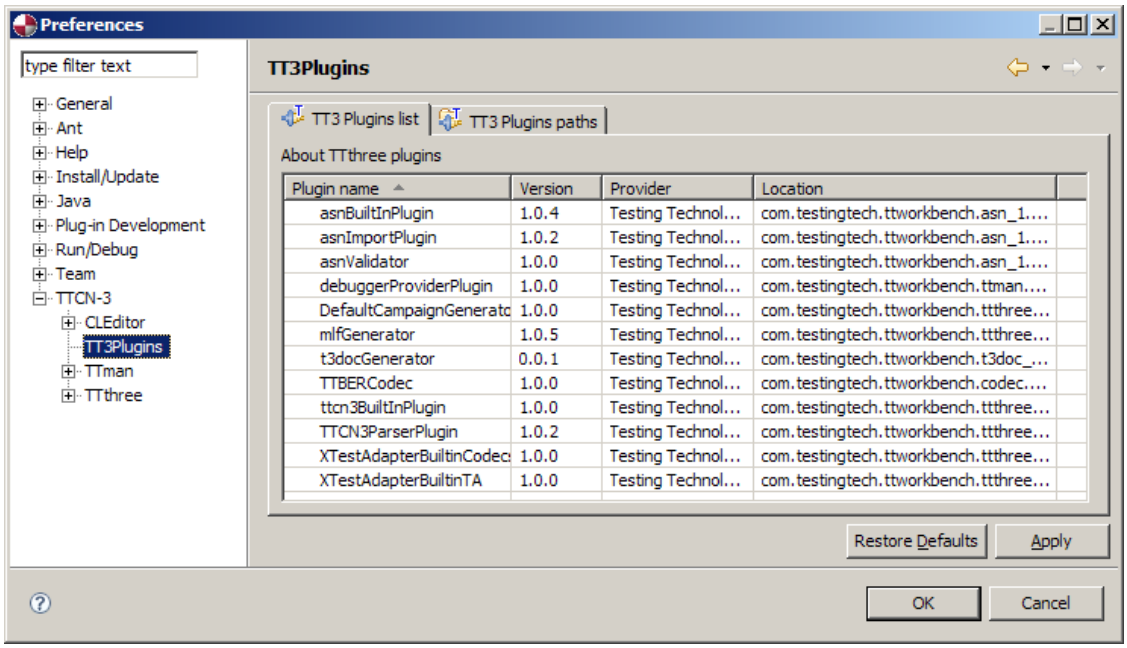
Non-standard language extensions provided by TTworkbench can be checked during compilation and they can be treated as errors or warnings or can be ignored. This can be set on the preference page shown in

Figure 3.1, “TTCN-3 main preferences”. The default setting is to issue a warning for each non-standard extension that is used.

TT3 plugins

The active TTthree plugins, which are not Eclipse plugins but extensions of the TTCN-3 compiler, shows the read-only TT3 Plugins list page as shown in Figure 3.2, “ TT3Plugins preferences: extensions of the TTCN-3 compiler ” . The second tab TT3 Plugins paths allows for dynamic addition and removal of plugins by managing their locations.

Figure 3.2. TT3Plugins preferences: extensions of the TTCN-3 compiler



Java Compiler

This page defines the Java compiler command and options for the generation of Java class files. Either the built-in Eclipse Java compiler or an external compiler can be configured.

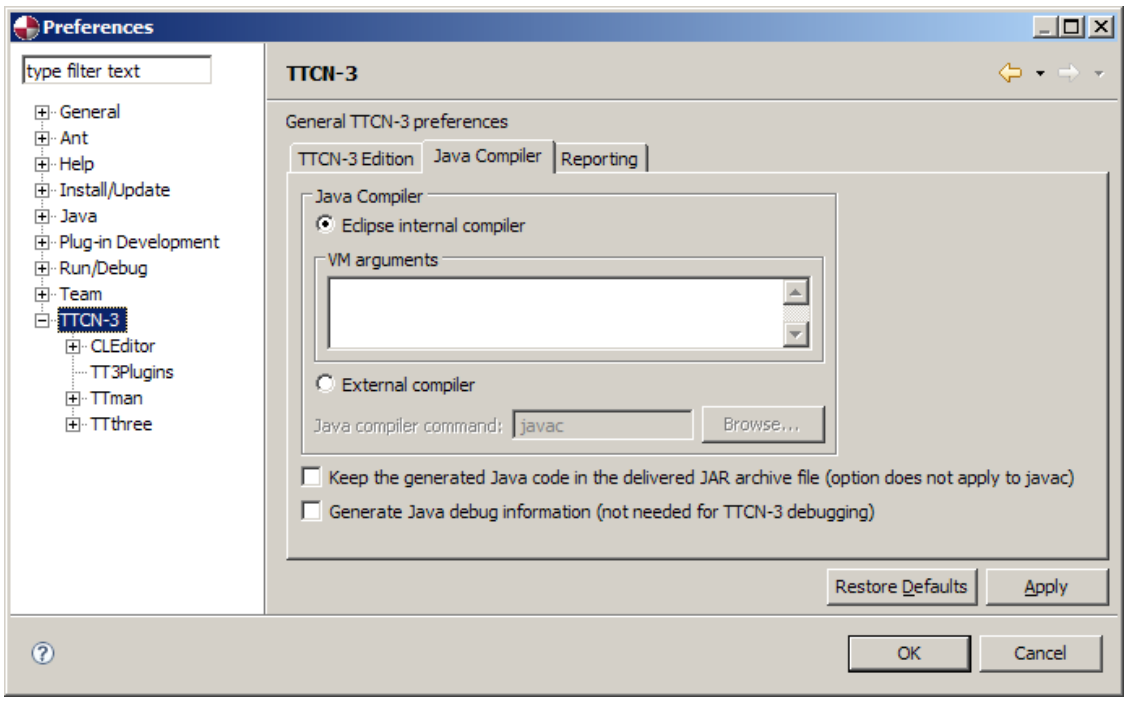
In case the internal compiler is used, this will be started in a separate JVM with a predefined configuration. In case this default configuration does not fit your requirements, additional parameters for the virtual machine can be configures, e.g. for more heap space **-Xmx768m**.

In case an external compiler has to be used, either the absolute path to the compiler can be given or only the name of the executable file. In the second case, TTthree assumes that the configured Java compiler, e.g. javac or jikes, is installed on your system and is in your PATH.

Keep the generated Java source code in the delivered JAR archive file: if deselected, the JAR archive file contains only the compiled Java class files. This option is not applicable for javac.

Debugging mode: if selected, the debugging function of the selected Java compiler is used. On selection, the verbosity level will be set to **info** (see also the section called “Reporting”) automatically.

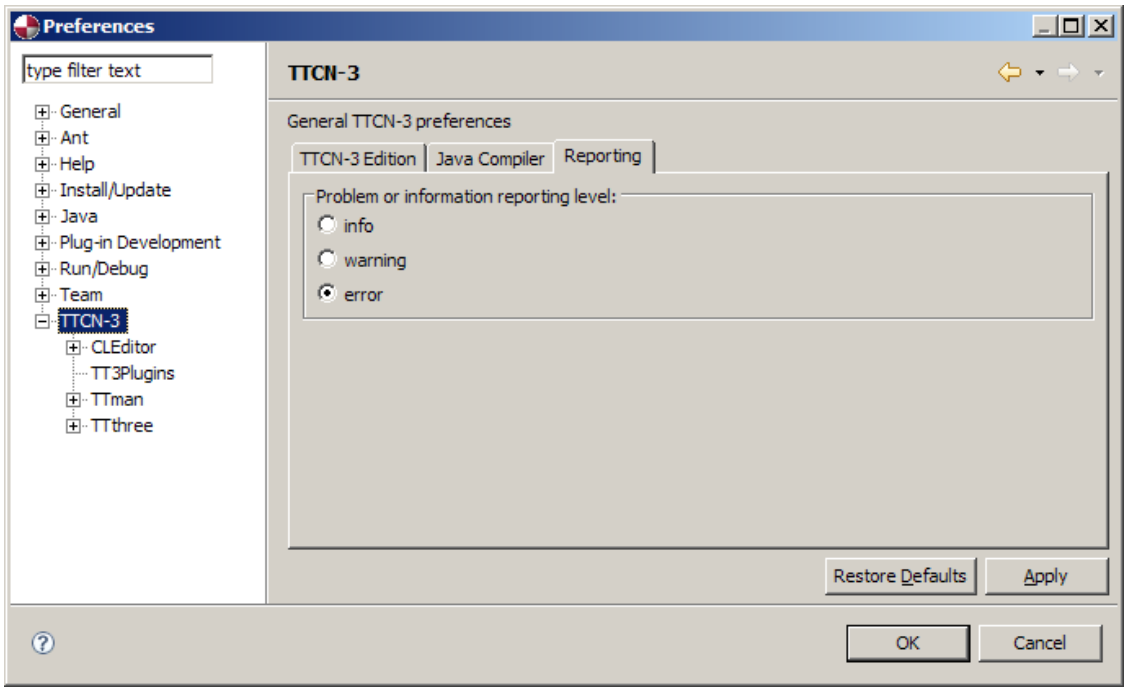
Figure 3.3. Java compiler preferences



Reporting

This preference page (Figure 3.4, “Reporting preferences”) defines the verbosity level for TTthree. Depending on the level, TTthree will produce output during the compilation process. The defined verbosity levels are: **info**, **warning**, and **error**.

Figure 3.4. Reporting preferences

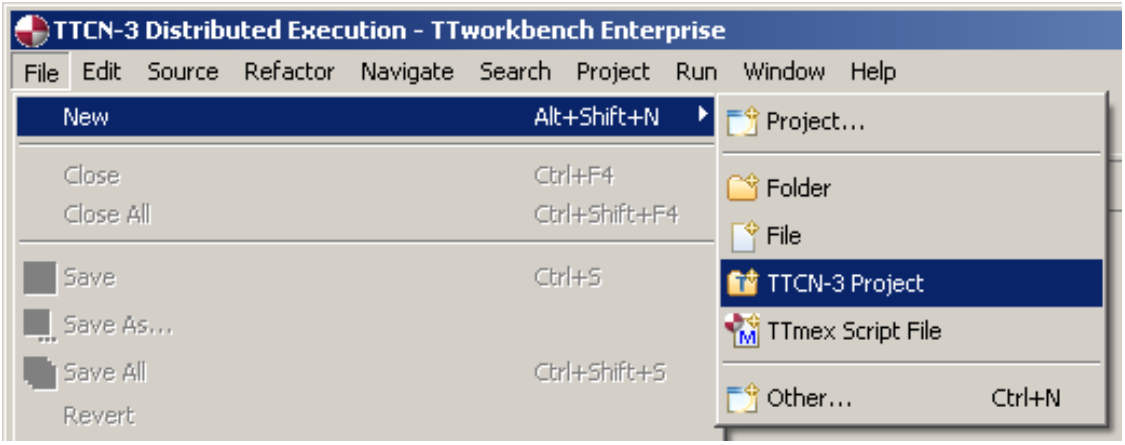


Chapter 4. TTCN-3 Project

Creating a TTCN-3 Project

A new TTCN-3 project can be created by using the menu **File > New > TTCN-3 Project** .

Figure 4.1. Creating a new TTCN-3 project



Now the New TTCN-3 Project wizard appears. As usual, first choose a name and location for the new project. A TTCN-3 project has the Java project nature too, because this is needed for developing test adaptors for instance. That's why on the second page of the wizard Java specific properties can be set. (Figure 4.2, “Java properties of new TTCN-3 project”). The third a last page of the wizard is needed to set TTCN-3 properties for this project (Figure 4.3, “TTCN-3 properties of new TTCN-3 project”). On all pages it is recommend to use the default values. Click "Finish" to get back to the workspace with the created project.

Figure 4.2. Java properties of new TTCN-3 project

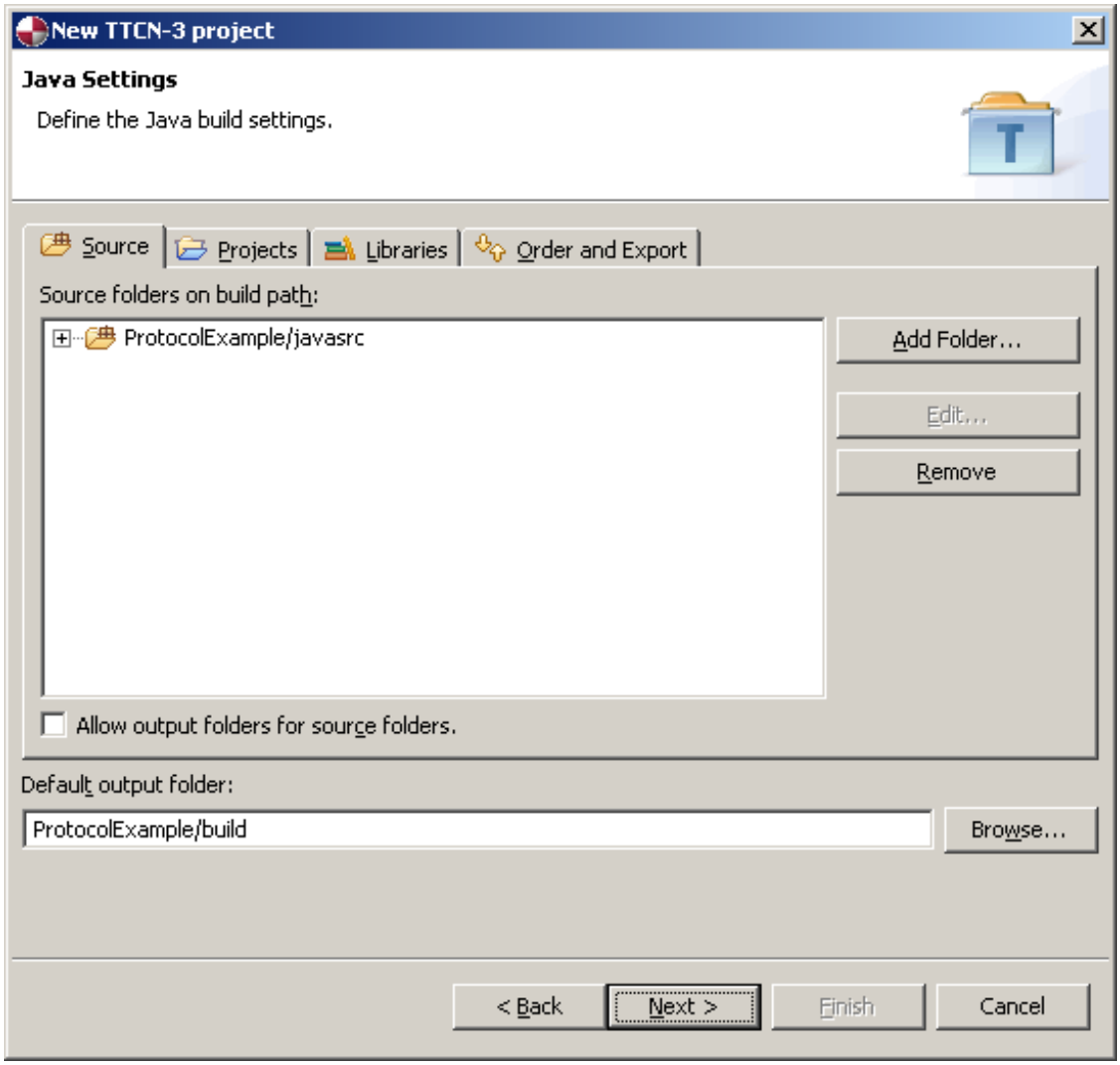
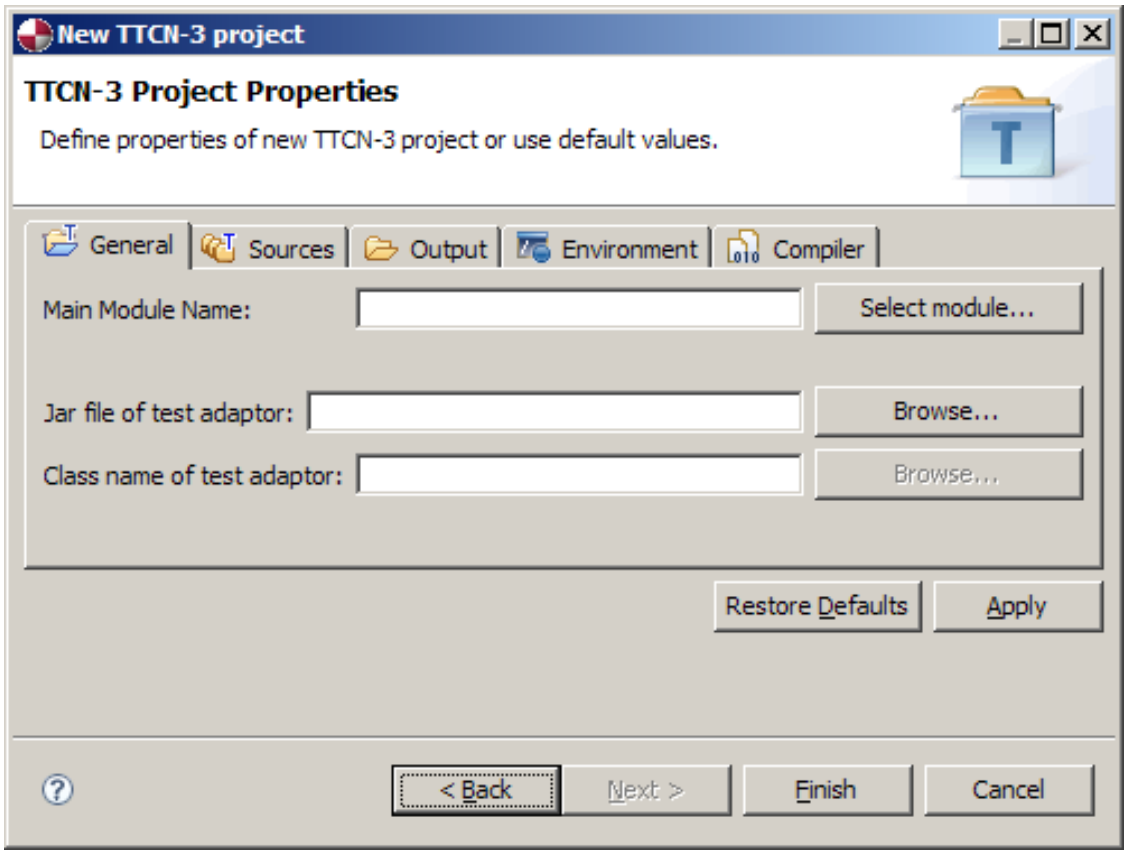
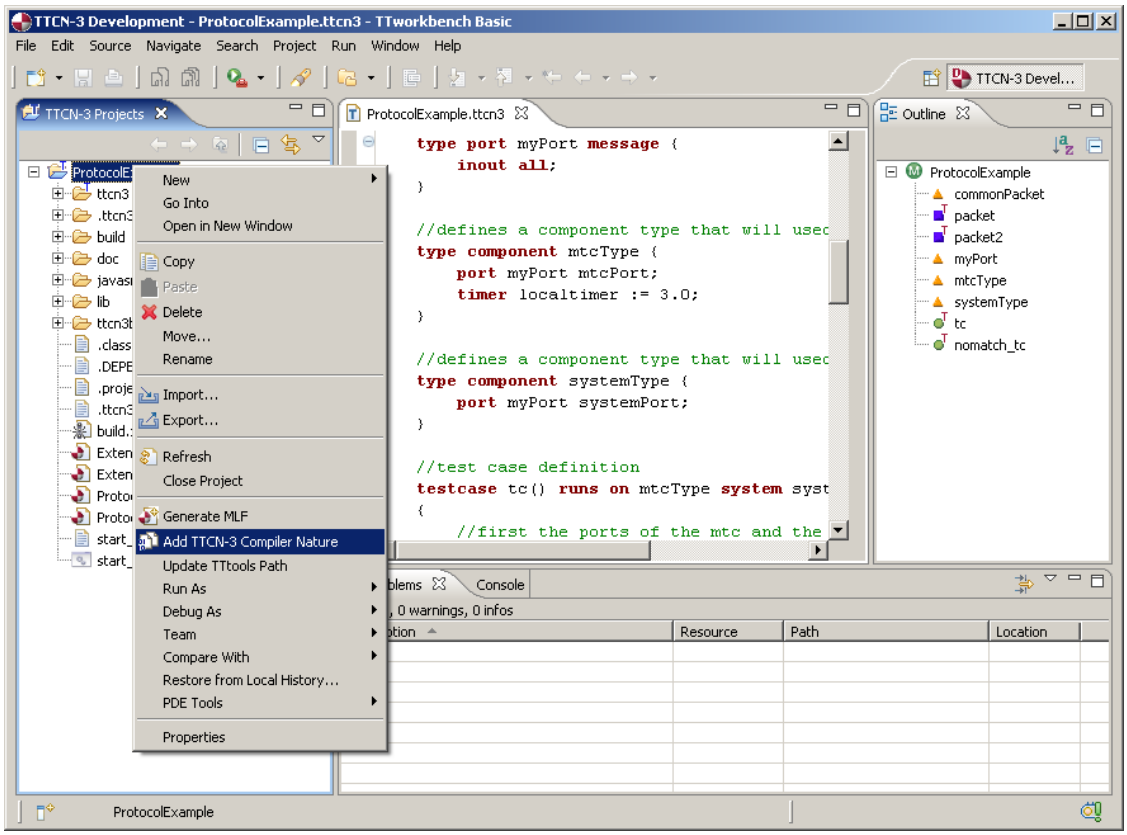


Figure 4.3. TTCN-3 properties of new TTCN-3 project



An already existing non-TTCN-3 project can be changed into a TTCN-3 project by using the menu commands **Turn into TTCN-3 Project** or **Add TTCN-3 Compiler Nature** . These commands can be found in the context menu for projects in the Workspace Navigator or Java Package Explorer views. These commands enable test execution and definition of a project's TTCN-3 properties.

Figure 4.4. Adding TTCN-3 compiler nature



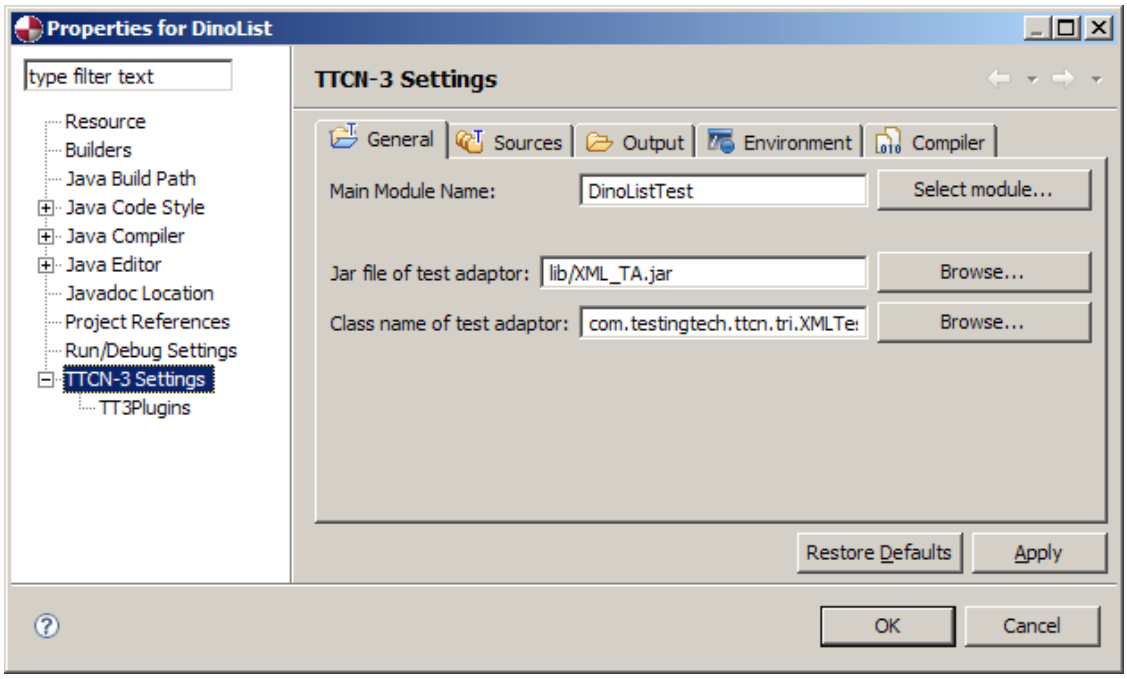
Project Properties

A TTCN-3 project can be configured by setting its Project Properties. Please go to **File** > **Properties** > **TTCN-3 Settings**. The properties are structured into several setting pages.

General

This page (Figure 4.5, “General settings page”) allows to set the name of the main module as specified in the TTCN-3 test suite, i.e. without file extension, and the standard test adaptor of the project.

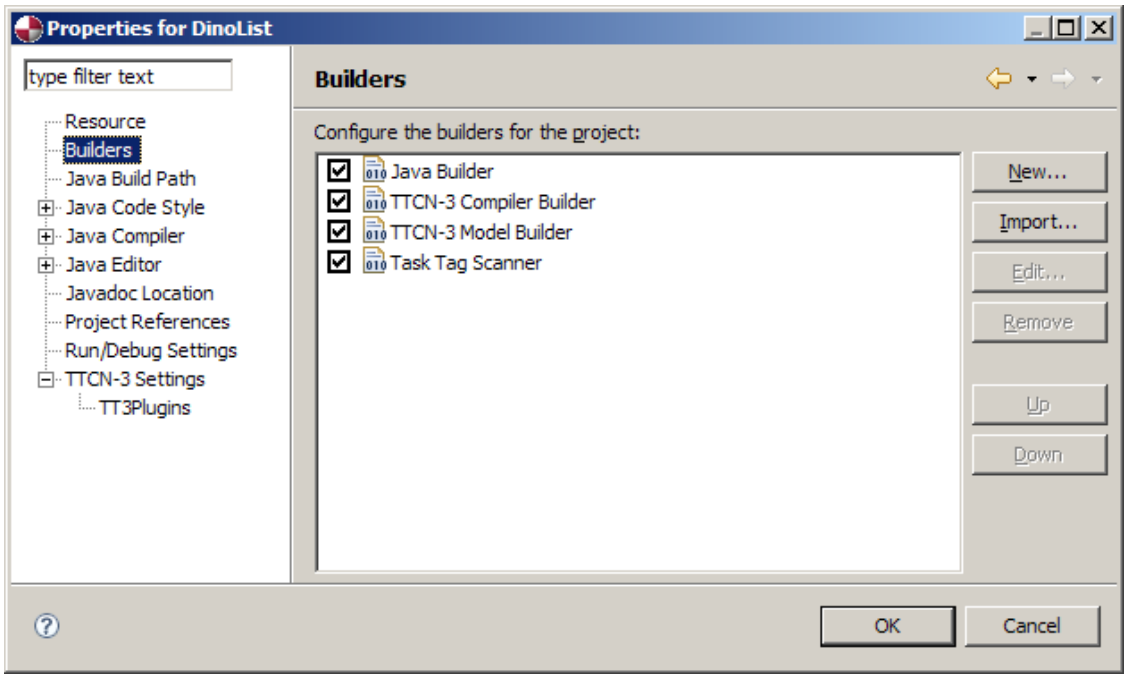
Figure 4.5. General settings page



Main module

The Main module of a project is depicted as  in the navigator view. The TTCN-3 compiler builder uses this module name to start automatic re-compilation upon file changes. This builder is active only when a TTCN-3 project has the TTCN-3 compiler builder nature in addition to its TTCN-3 nature. Automatic re-compilation can be temporarily turned off by clearing the checkbox for this builder in the Builders property page of this project. Figure 4.6, “Builders properties of a TTCN-3 project featuring the TTCN-3 compiler nature” shows this page. The nature can be removed entirely by choosing "Remove TTCN-3 Compiler Nature" in the context menu of a project.

Figure 4.6. Builders properties of a TTCN-3 project featuring the TTCN-3 compiler nature



The main module can also be set from within the TTCN-3 Projects navigator view. In the context menu appearing when you right-click on a TTCN-3 source file inside a TTCN-3 project, you'll find the action TTCN-3 Sources > Set as Main Module. This sets the current project's Main Module to the selected TTCN-3 module.

Test adaptor

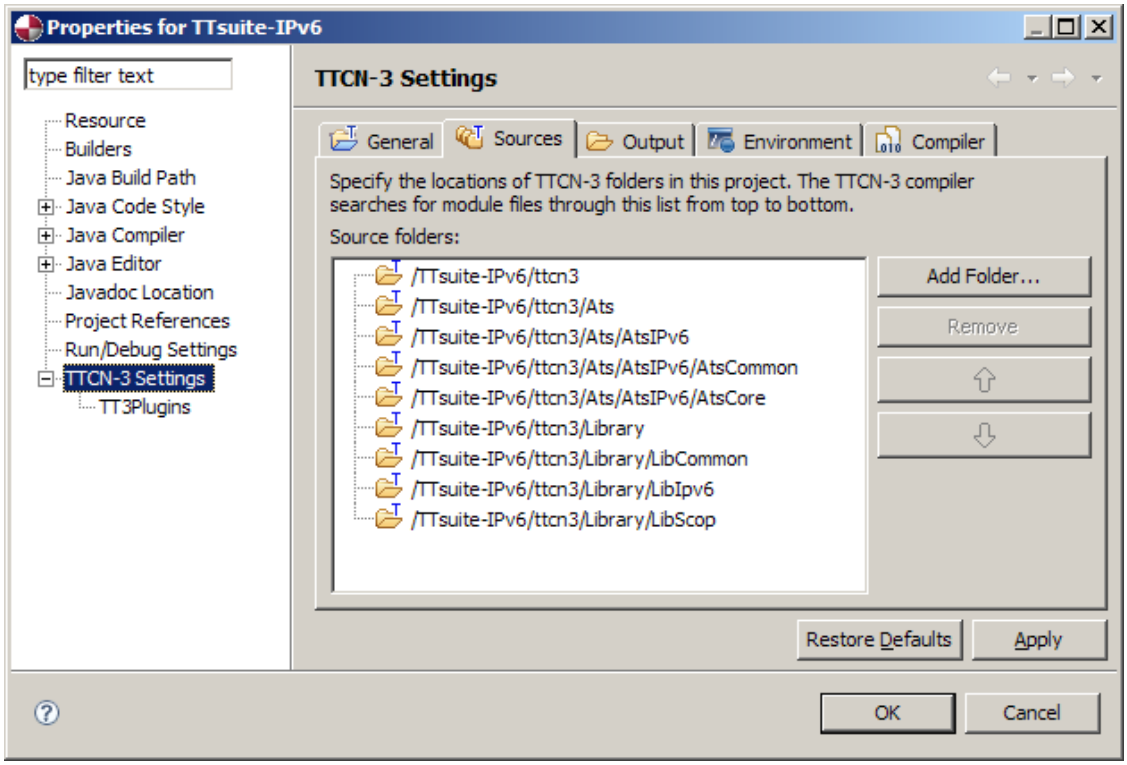
If the main module is specified, setting the Jar file that contains the test adaptor and selecting the adaptor class enables the generation of a Default Campaign for this project.

TTCN-3 Sources

This page (Figure 4.7, “TTCN-3 Sources settings page”) specifies the search path for imported modules. It is a list of so-called *TTCN-3 source folders* where module files should be located. The path is used by CL Editor and by TTthree. The default path is the directory `ttcn3` in the project root. Only folders from the current project can be selected here. When mapping TTCN-3 module names to files, the compiler searches through this list from top to bottom. It doesn't descend into subfolders, so any folder that is to be searched has to be added to this list.

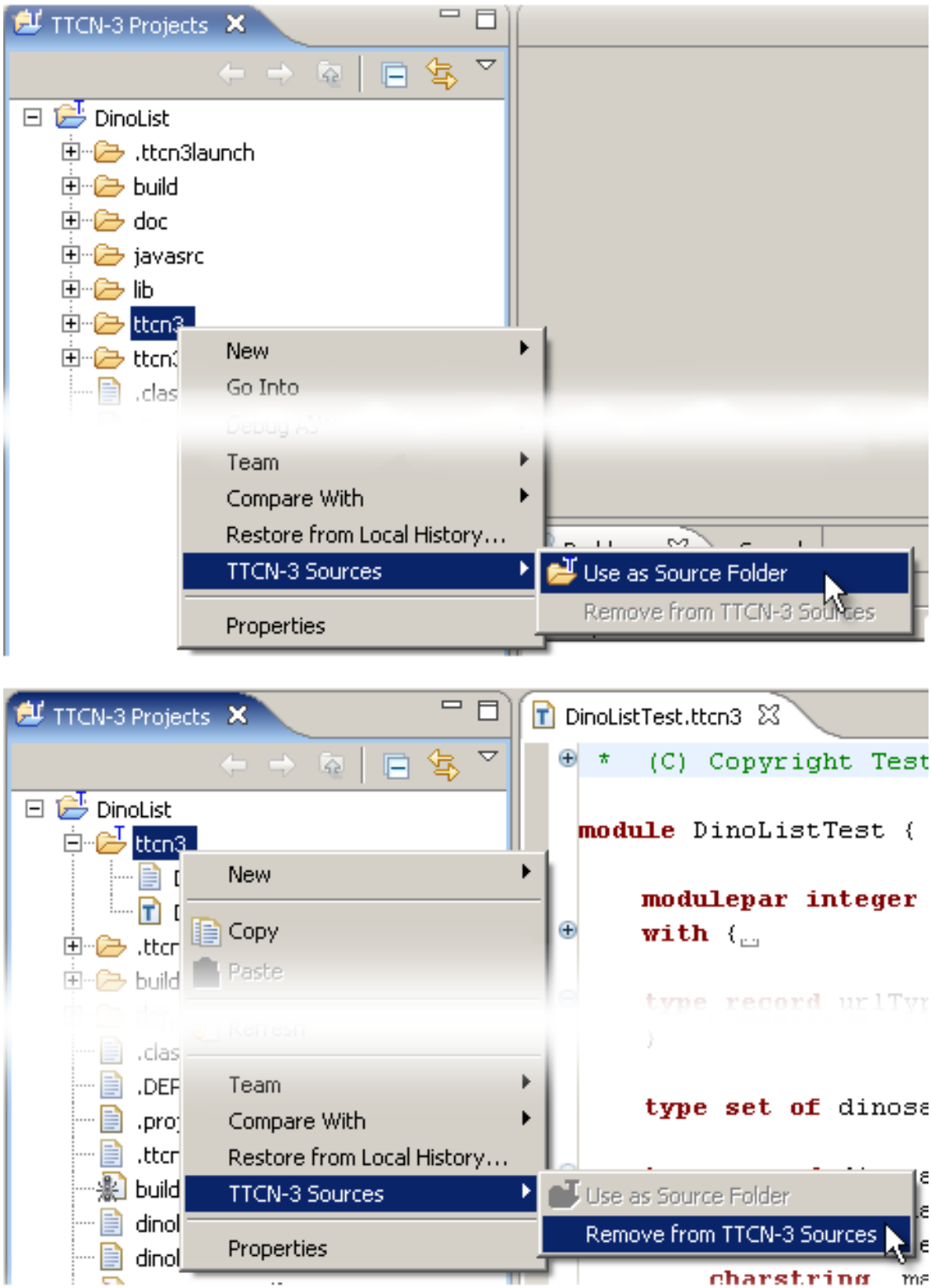
If modules from a different project need to be imported, those projects have to be selected in the property page Project References. In the referenced projects the TTCN-3 Sources path has to be set accordingly.

Figure 4.7. TTCN-3 Sources settings page



For adding or removing folders from this list, the TTCN-3 Projects navigator view provides two shortcuts. In the context menu appearing when you right-click on a folder inside a TTCN-3 project, you'll find the actions TTCN-3 Sources > Use as Source Folder and TTCN-3 Sources > Remove from TTCN-3 Sources. They change the current project's TTCN-3 Sources setting. This also works with a selected group of folders.

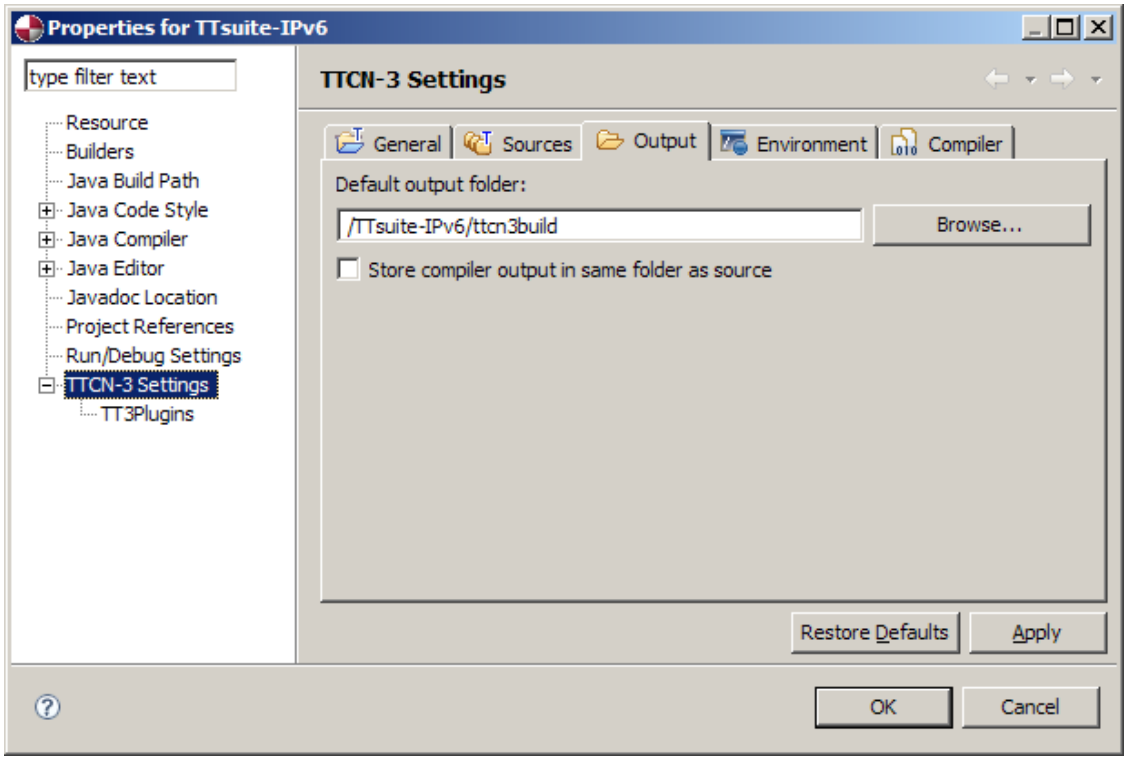
Figure 4.8. TTCN-3 Sources menu actions in the TTCN-3 Projects navigator view



Output

This page (Figure 4.9, “Output folder setting page”) defines the folder TTthree shall use as output folder for the compiled test suite.

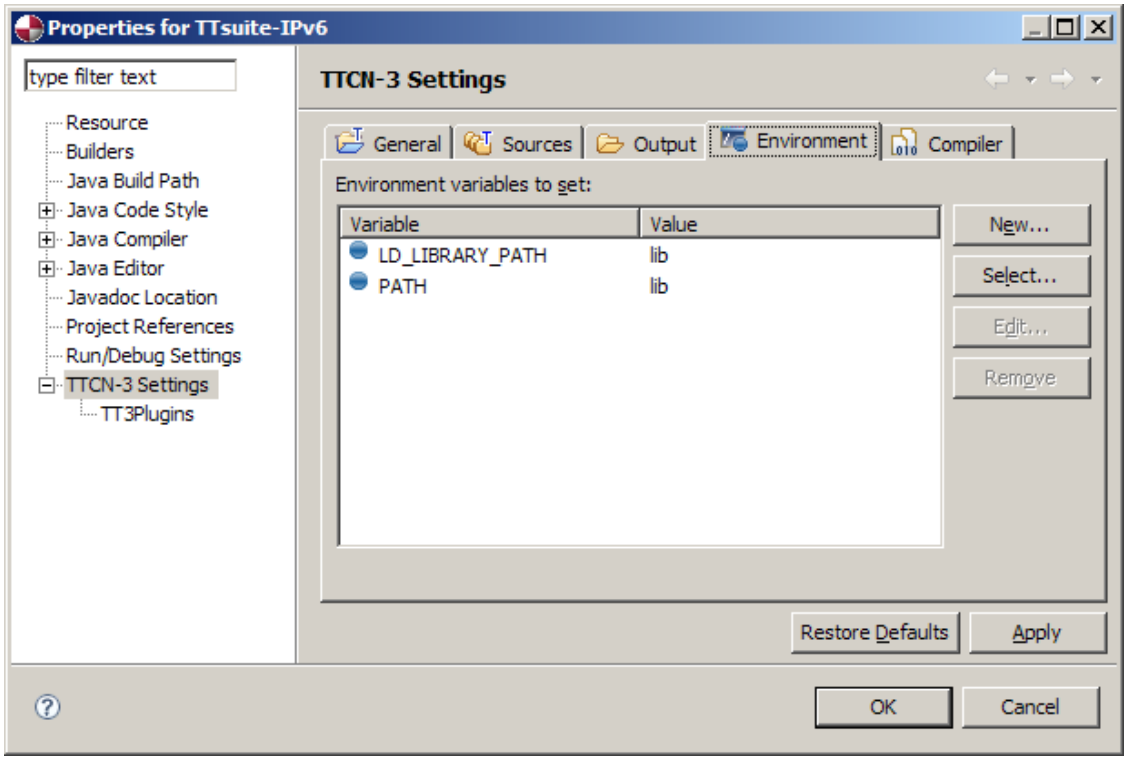
Figure 4.9. Output folder setting page



Environment

For a test execution it may be required that particular environment variables have specific values. These can be defined here. It is possible to re-define existing variables as well as to define new variables.

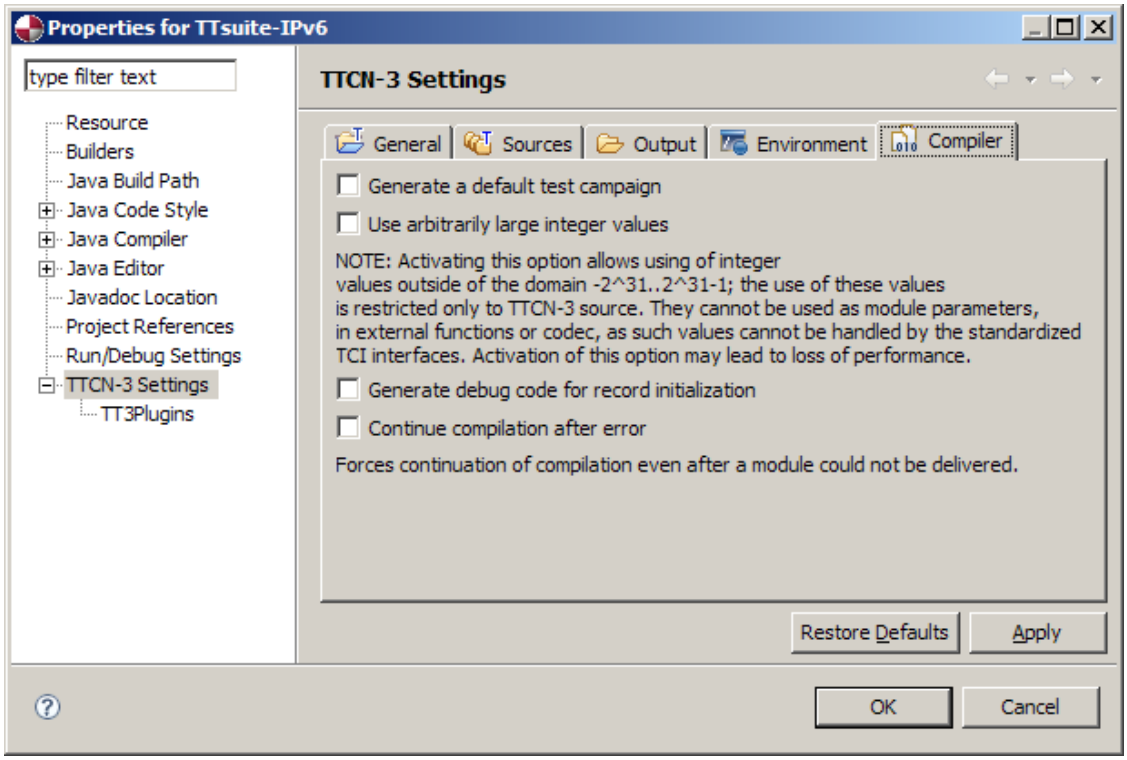
Figure 4.10. Environment variables setting page



Compiler settings

This page (Figure 4.11, “Compiler settings page”) provides the possibility to configure project specific compiler options.

Figure 4.11. Compiler settings page



Generate a default test campaign

If this option is set, the compiler automatically generates a Default Campaign after a successful compilation. The default test campaign is generated only after a compilation (build or rebuild), not after a validation. No output is generated in case the module that is compiled is not set as root module. To allow this to work, a main module and a test adapter have to be set (see the section called “General”).

Use arbitrarily large integer values

Activating the option "Use arbitrarily large integer values" allows using of integer values outside of the domain $-2^{31}..2^{31}-1$; the use of these values is restricted only to TTCN-3 source. They cannot be used as module parameters, in external functions or codec, as such values cannot be handled by the standardized TCI interfaces. Activation of this option may lead to loss of performance.

Generate code for debugging

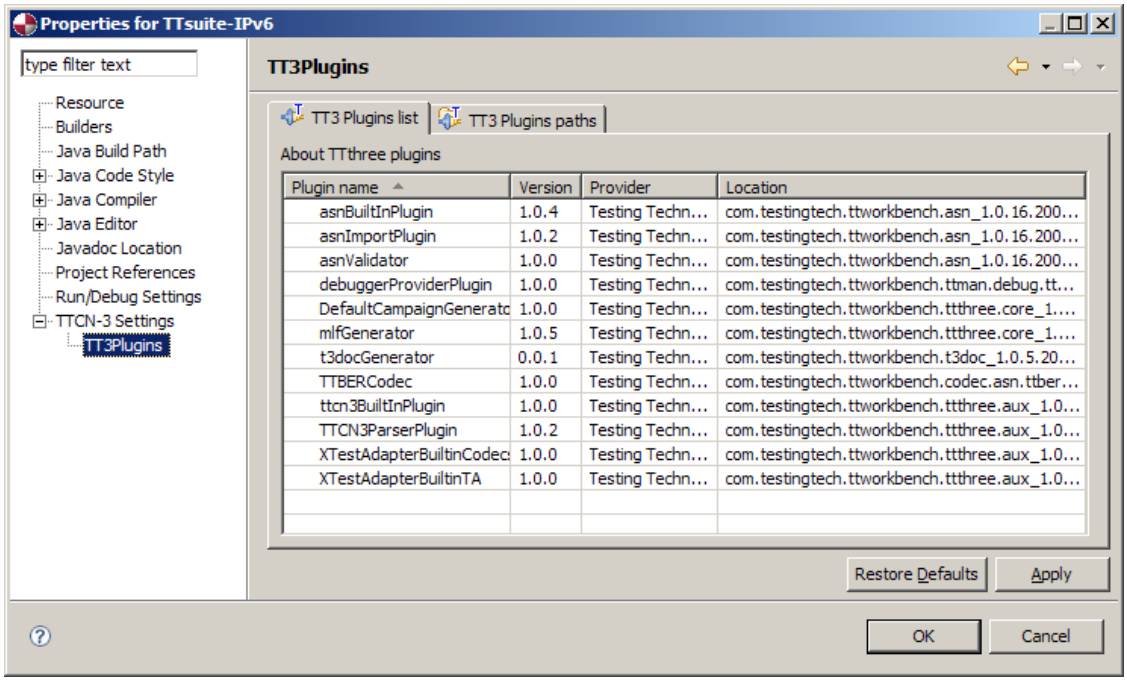
Generate debug code for record initialization. If set, generated java classes will contain code which can be used to monitor initialization of TTCN-3 record structures. By default no such code will be generated.

TT3 Plugins

Extensions to the TTCN-3 compiler, called TT3 Plugins, are usually defined on workspace level as described in TT3 Plugins. For specific puposes it may be useful to define a TT3 Plugin on project level. In this case it will only be used while compiling and executing test cases in this project.

To specify a TT3 Plugin, it must be located in the workspace. A folder in the current project is recommended. Click the Add Folder... button and select the folder, where the TT3 Plugin's files can be found.

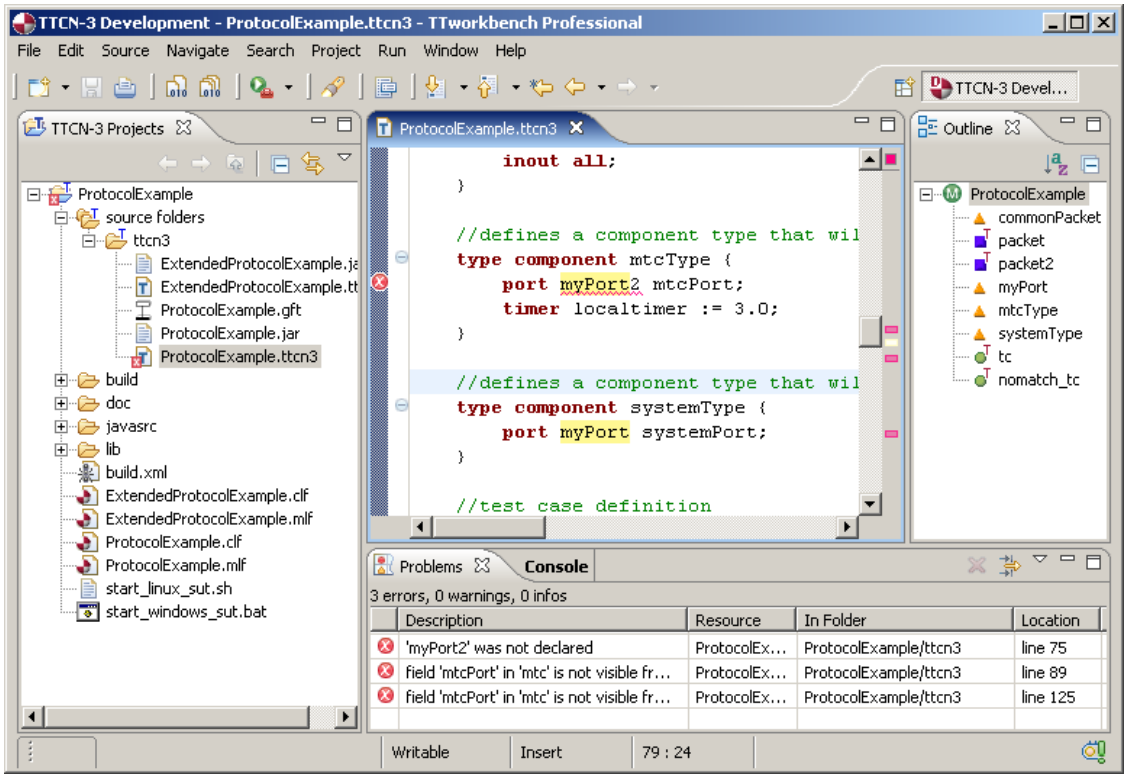
Figure 4.12. Project specific TT3 Plugins page



Chapter 5. Using TTworkbench CL Editor

The CL Editor is a text editor that provides rich capabilities for editing TTCN-3 Core Language based test suites. It relies on the Eclipse basic text editor framework. The layout and the functionality of the CL Editor are in-line with other Eclipse based editors. The CL Editor, as shown in Figure 5.1, “TTworkbench CL Editor”, uses different views, for example to outline the test suite structure, to collect errors and warnings found during syntax checking, or to print progress information. The functionality of the CL Editor can be parameterized by dedicated preference pages. The remainder of this section describes the properties and usage of the CL Editor in detail.

Figure 5.1. TTworkbench CL Editor



Properties

The CL Editor provides currently the following functions:

- Full support for TTCN-3 Edition 1 (TTCN-3:2001), 2 (TTCN-3:2003), 3 (TTCN-3:2005 and TTCN-3:2008) - ETSI ES 201 873-1
- Standard text editor functions e.g. open/save, copy/paste, find/replace
- Syntax highlighting
- Syntax checking with navigation of errors/warnings

- Text formatting
- Bookmarks and tasks
- Preference page
- Extended context menu
- Outline of test suite structure
- Outline view components
- Find references in workspace
- Quick outline
- File association
- Search for declarations in the global context
- Wizard for creation of new TTCN-3 module
- Completion assistance
- A message building system, called template wizard
- Quick fix for certain problems and errors
- Comparisons of local files with each other or with a SCM repository
- TTCN-3 search
- Folding support for template fields

Known limitations:

- **address** type not supported
- **variant** attribute not supported
- Limited support for **universal charstrings**

Development Perspective

Overview

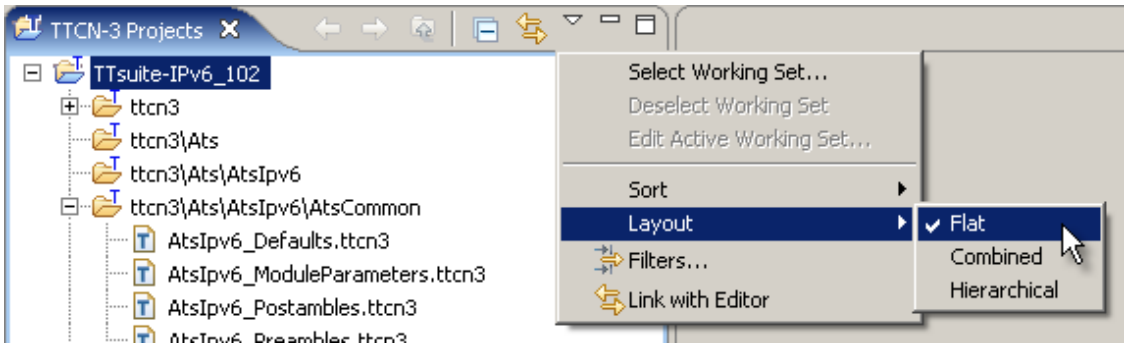
The TTCN-3 Development Perspective  constructs the editing environment of CL Editor. Besides the editor, the Development Perspective consists of an outline view, a problems view, a console view and the TTCN-3 Projects navigator view  . The Development Perspective is launched automatically upon first startup. It can be also explicitly started by **Window > Open Perspective > TTCN-3 Development**.

TTCN-3 Projects navigator view

The TTCN-3 Projects navigator view is based on Eclipses Resource Navigator view, part of the Resource Perspective. It supports any kind of project in the workspace. Projects can be grouped into Working Sets.

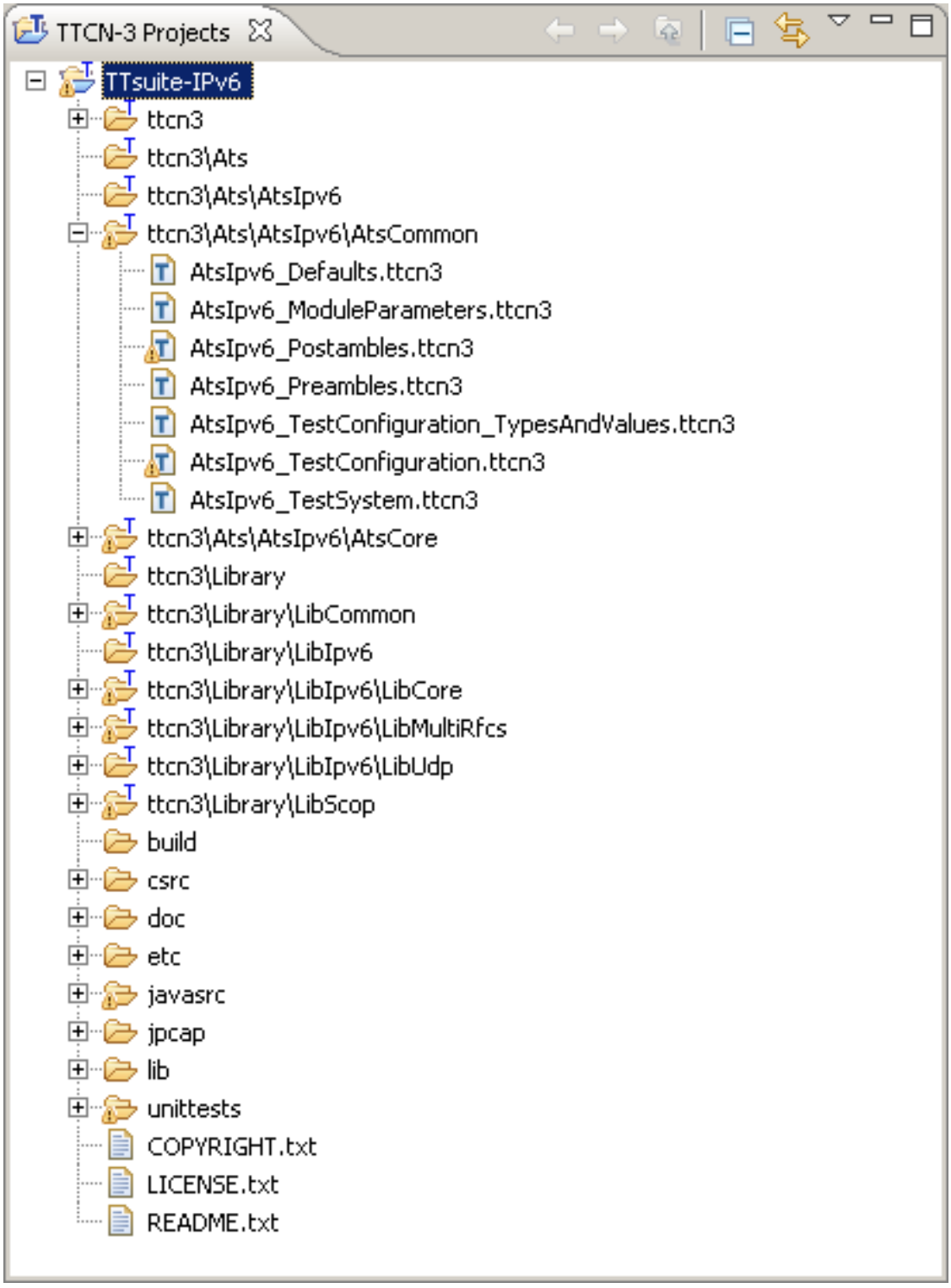
In contrast to similar workspace views in the Eclipse environment, this view focuses on TTCN-3 related aspects of projects. So it lists TTCN-3 modules, TTCN-3 source folders (as set in the project's preferences, see the section called “TTCN-3 Sources”) and their parent folders first. The TTCN-3 Projects navigator view supports three layout modes for source folders, Flat, Hierarchical and Combined. The modes differ in the way they present TTCN-3 source folders and their contents. Layout modes can be switched via the local view menu.

Figure 5.2. Switching layout mode in the TTCN-3 Projects navigator view



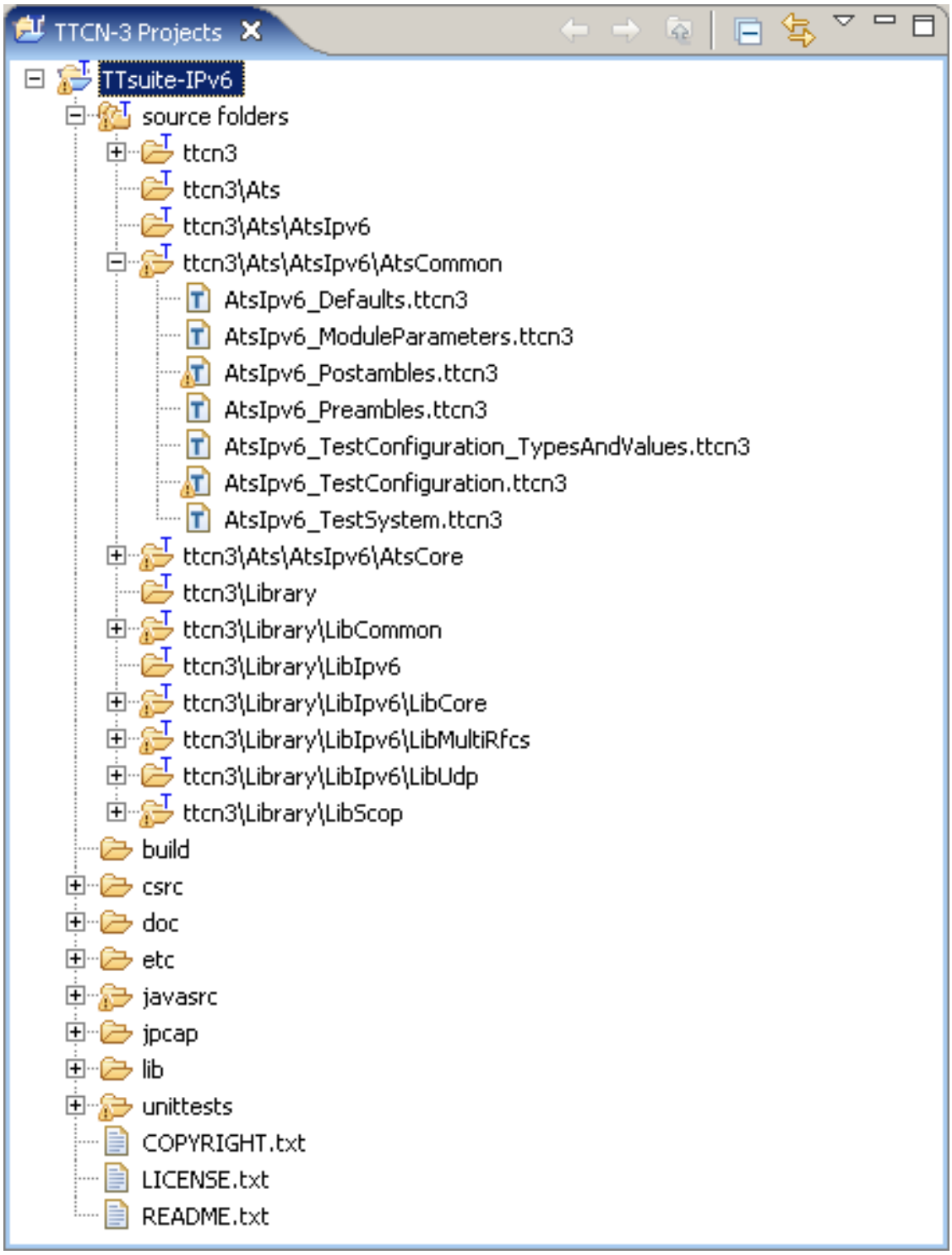
In Flat layout mode all TTCN-3 source folders are visible in the root of the project. Their hierarchical structure is condensed (flattened) in order to ease navigation in projects with many or deeply structured source folders.

Figure 5.3. Flat layout mode of the TTCN-3 Projects navigator view



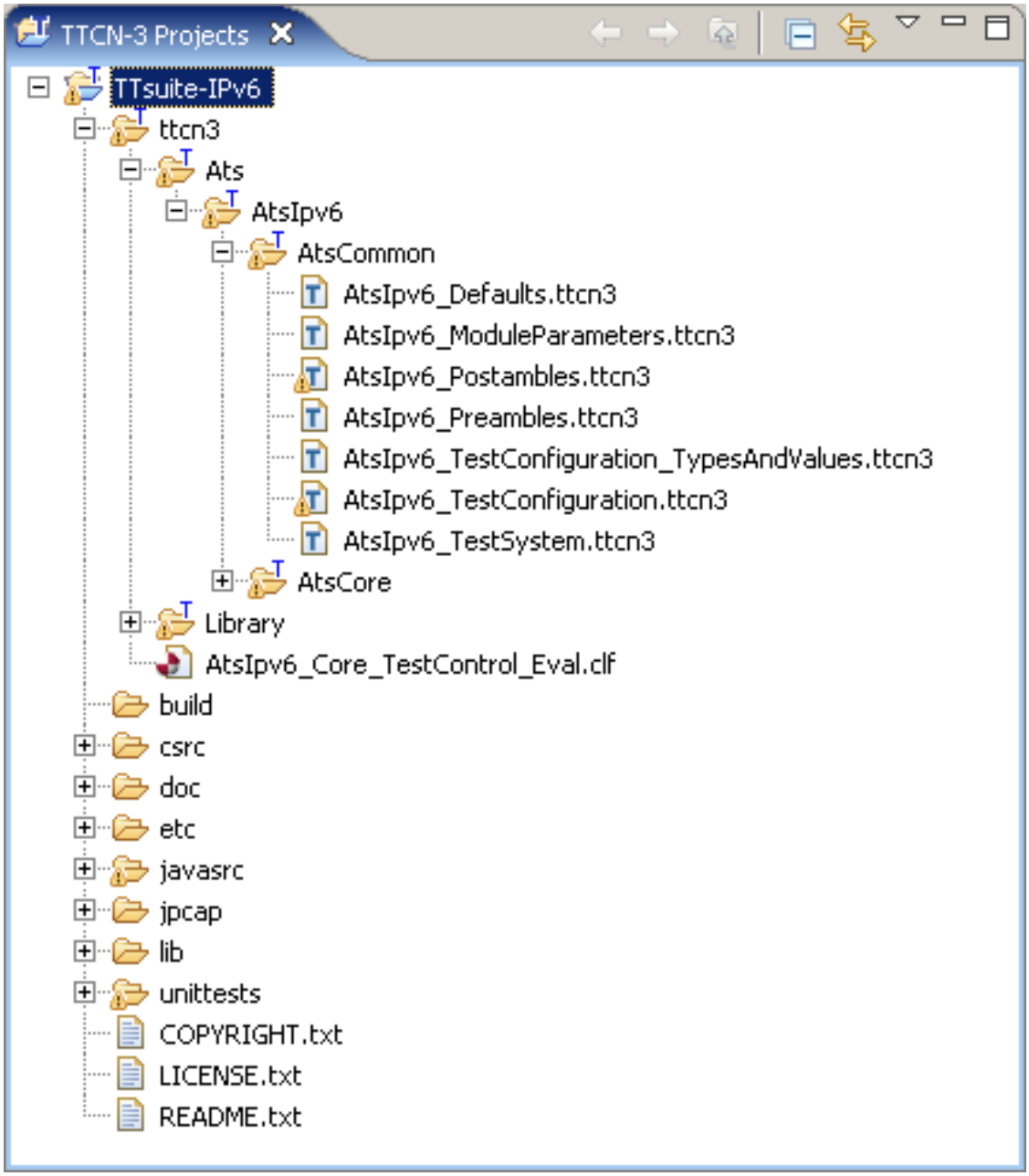
In Combined layout mode all TTCN-3 source folders get combined under a virtual folder called *source folders* in the root of the project. Their hierarchical structure is like in Flat layout mode condensed (flattened) in order to ease navigation in projects with many or deeply structured source folders.

Figure 5.4. Combined layout mode of the TTCN-3 Projects navigator view



In Hierarchical layout mode source folders keep their file system structure. As said above they are listed first in a project's tree structure regardless of their name.

Figure 5.5. Hierarchical layout mode of the TTCN-3 Projects navigator view



Editing a Test Suite

Introduction

A test suite may consist of one or more modules. For each module, a file with the extension ".ttcn3" must be created. The file name shall be the same as the identifier of the module. For example, the file `ProtocolExample.ttcn3` describes the module `ProtocolExample`.

By default, the extension ".ttcn3" is associated with the CL Editor. This can be verified using **Window > Preferences > Workbench > File Associations**.

To create `ProtocolExample.ttcn3` in the project `Demo`, the New TTCN-3 Module wizard can be used, which is accessible over **File** > **New** > **TTCN-3** > **Module**. The source folder is in this case the location of the project `Demo`, and the module name is `ProtocolExample` (see Figure 5.6, “New module wizard”). The wizard will create the file `ProtocolExample.ttcn3` and open it automatically.

Figure 5.6. New module wizard

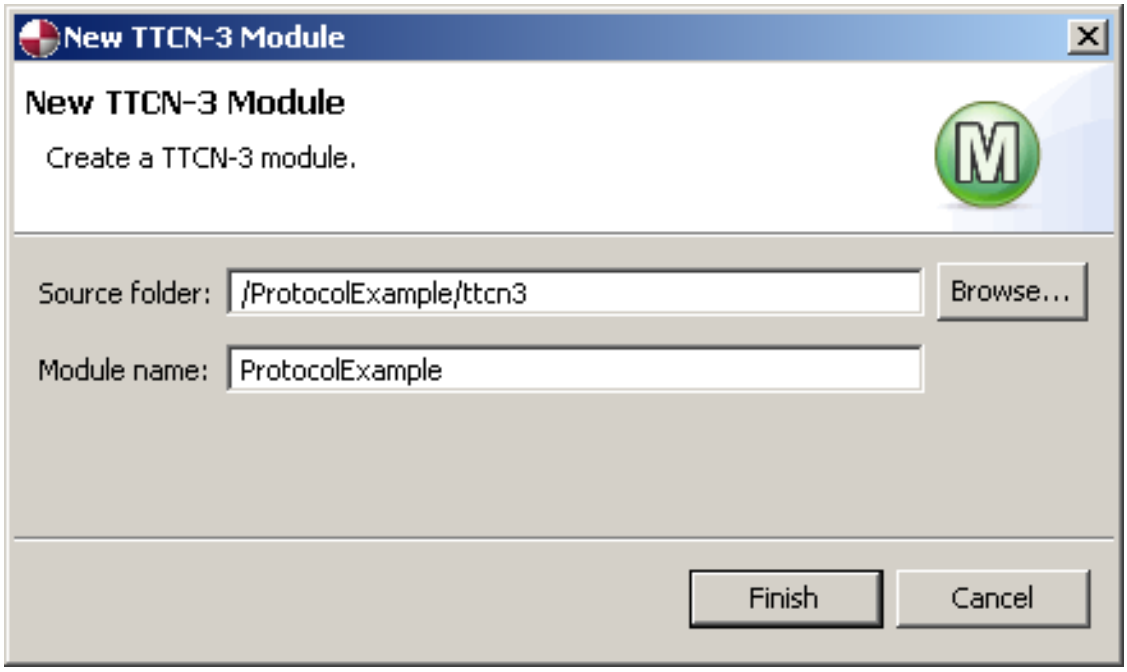
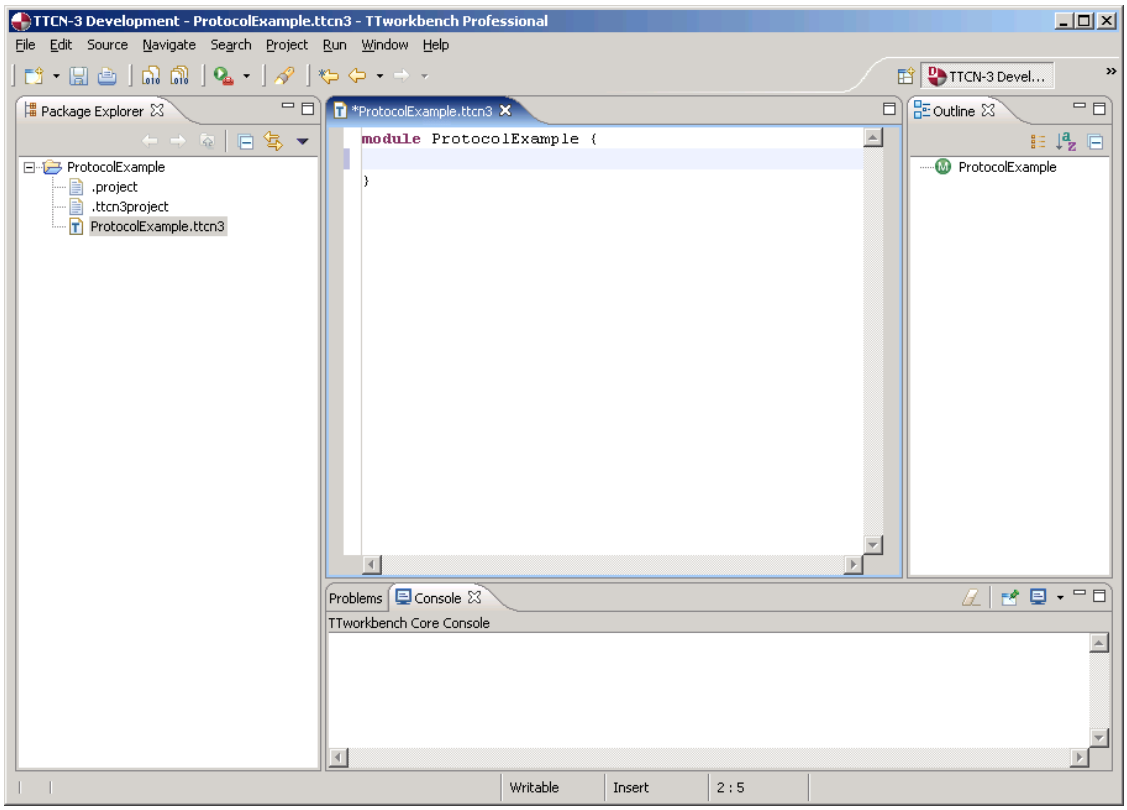


Figure 5.7. New module in editor

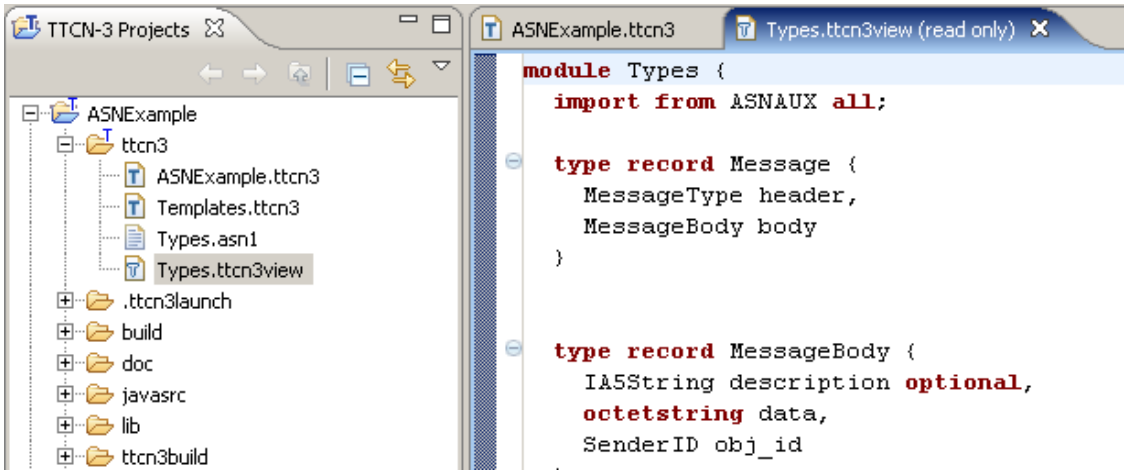


To open an existing TTCN-3 source file with the CL Editor, double-click on the file if CL Editor is set for the file association (default). Otherwise, an editor of the choice can be selected by right-click and **Open With**. As CL Editor uses the Eclipse text editor framework, standard commands such as copy, paste, save, revert are already available. In addition, CL Editor provides highlighting of keywords, validation of syntax, formatting of text, navigation of declarations, completion assistance, etc., as introduced in the following. The CL Editor actions such as Format, Validation, Open Declaration are integrated into the standard editor menus. They are also available over the context menu.

Using Core Language Viewer (CLViewer)

Editing generated TTCN-3 source code may not be reasonable. So by default for `*.ttcn3view` files which are generated for instance from ASN.1 inputs the CLViewer is used. It opens a file in read only mode and shows this state in the title bar. Those files have the viewer icon ( ) instead of the icon used for editable files ( ).

Figure 5.8. Editable and non-editable generated TTCN-3 files in the navigator

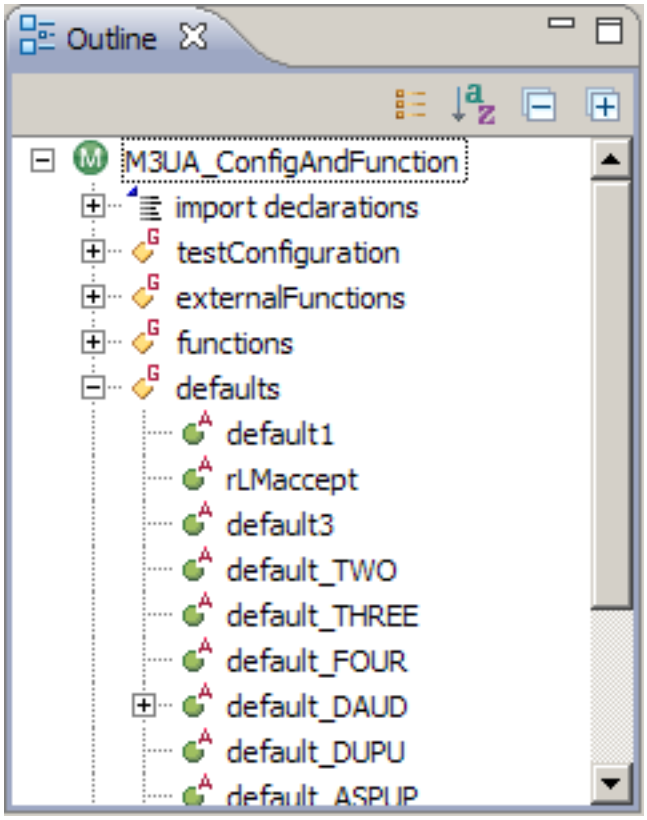


Outline of Test Suite Structure

The structure of the module being edited is shown as a tree in the **Outline** view, which can be opened by **Window > Show View > Outline**.

By default the definitions included in a TTCN-3 module are shown in the order of their appearance in the file. Alphabetical sorting is available by clicking action icon . Regardless of sorting mode import declarations are listed first for clarity.

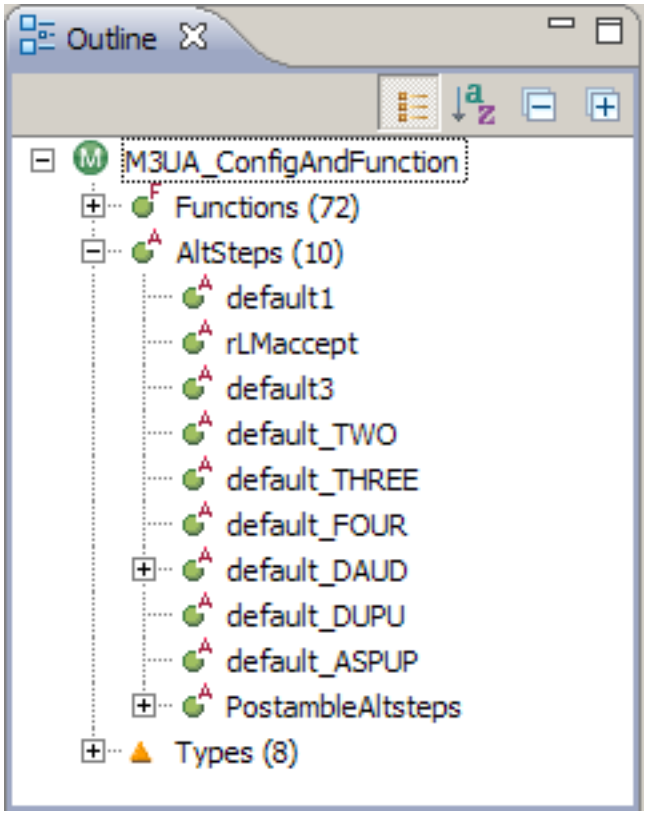
Figure 5.9. Outline showing definitions as found in module



The outline view always synchronizes with the selection in the current TTCN-3 editor (CL Editor or CL Viewer) and vice versa.

An alternative view mode can be enabled with the button **Group by Category** switch in the outline's tool bar. Definitions of the same kind will be put in a (virtual) folder which shows the number of definitions it contains.

Figure 5.10. Outline with definitions grouped by category



Quick Outline

To use the Quick Outline feature in the TTCN-3 Core Language Editor, perform one of the following procedures:

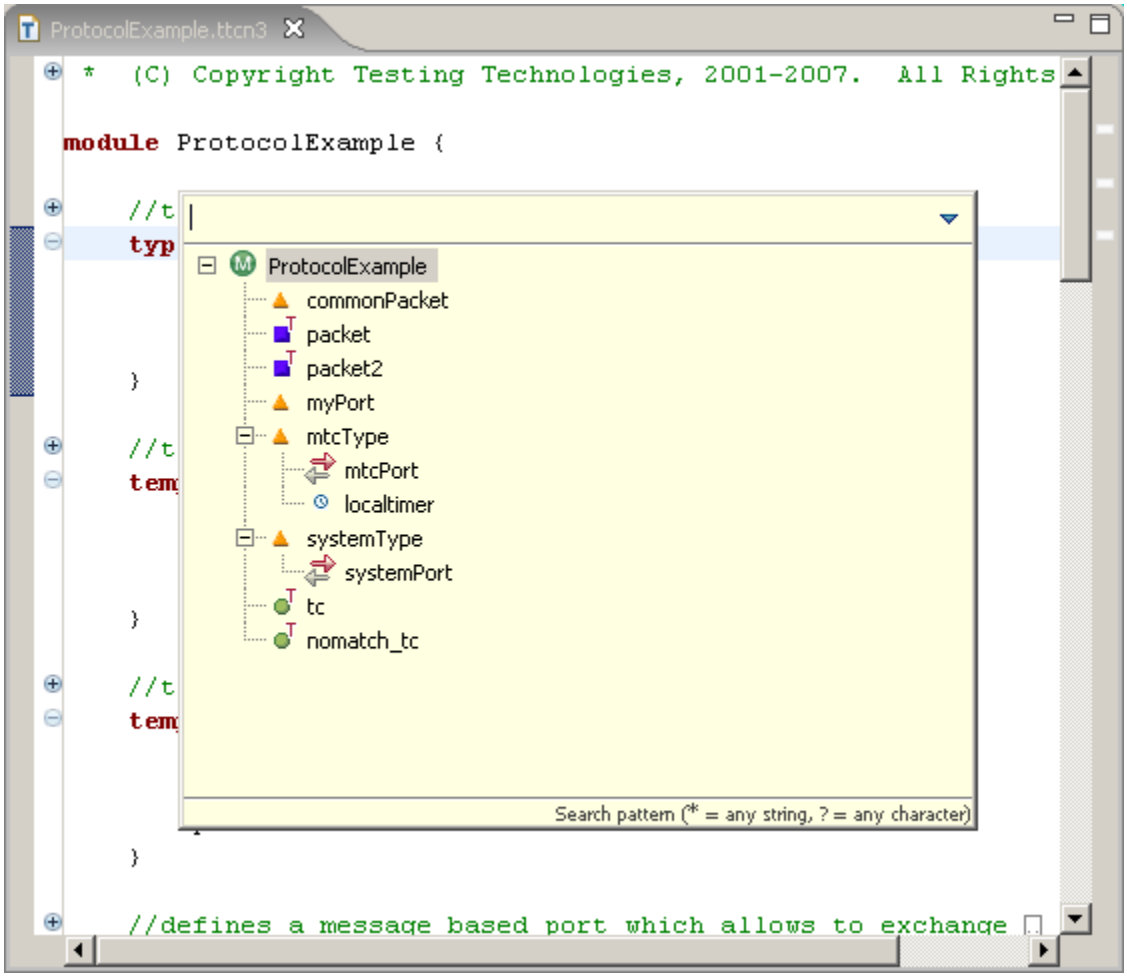
- Press Ctrl+O.
- In the main window's menu bar, select **Navigate > Quick Outline**.
- In the TTCN-3 Editor, right-click and select Quick Outline.

After selecting an element either by clicking on it with the left mouse button or pressing enter after choosing the right element with the arrow keys, the editor will jump to the element's declaration.

At the top of the Quick Outline window, there is a line editor. The Outline below gets filtered according to what you type here.

To customize the size and position of the Quick Outline window, click the triangle in the upper right corner and choose the appropriate option.

Figure 5.11. Quick outline



TTCN-3 Search

TTworkbench offers the possibility of searching for TTCN-3 elements. To open the TTCN-3 search dialog, press **Ctrl+H** or in the main window's menu bar, choose Search > Search. In the now opening dialog, open the TTCN-3 Search tab.

To specify your search, enter your search string in the text field at the top. For setting more detailed contraints on the searched element or limiting the searched locations, there are three sections available with the following options:

- **Search for:**

  Type

  Group

  Module

  Template

  Module parameter

Constant

Field

- **Limit to:**

  Declarations

  References

  All occurrences

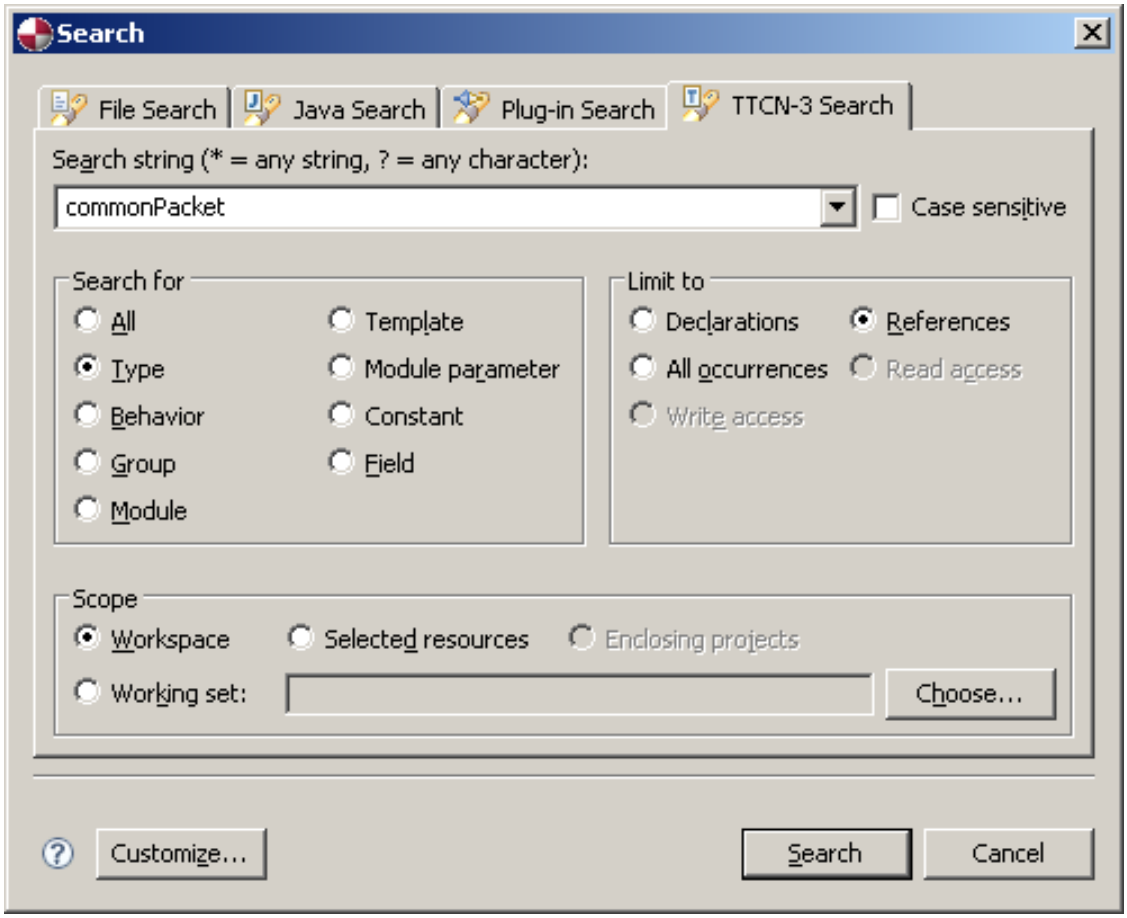
- **Scope:**

  Workingspace

  Working set

  Enclosing projetcts

Figure 5.12. TTCN-3 search



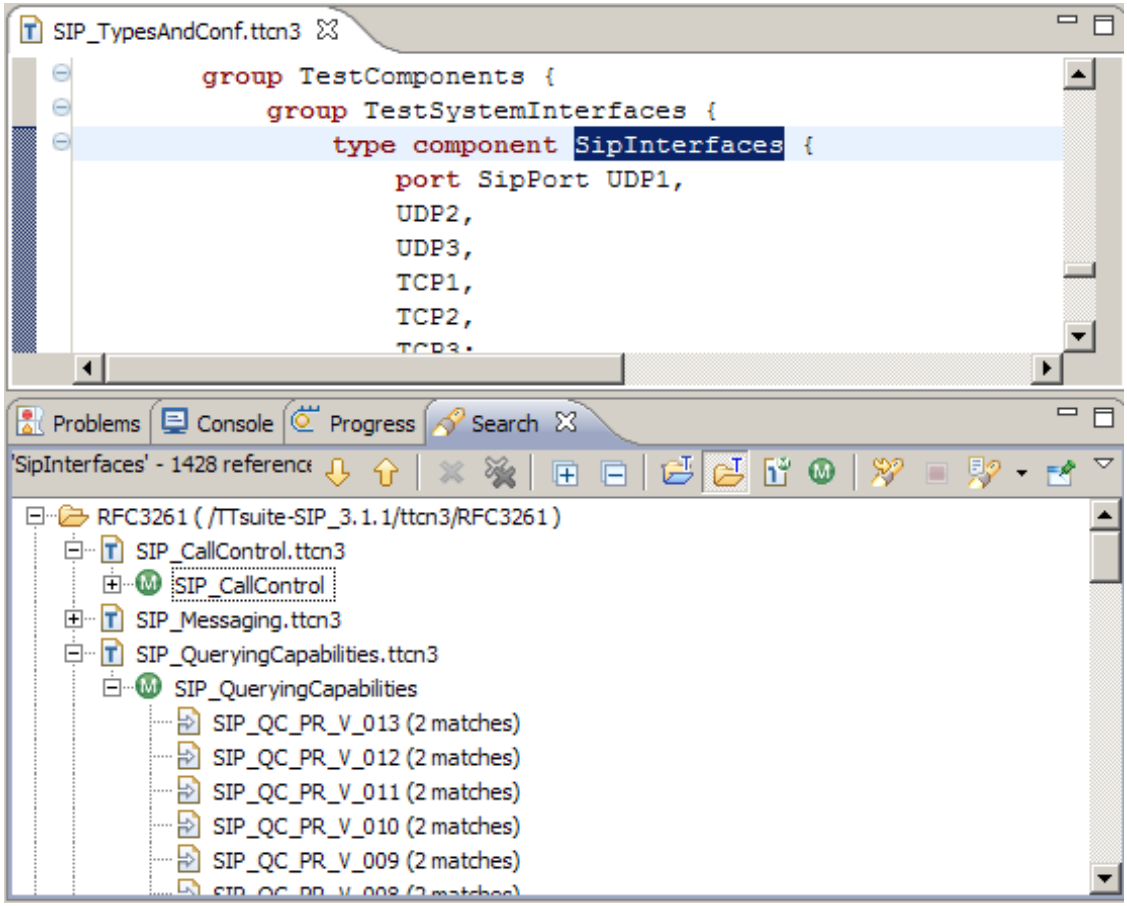
Find references in workspace

Find references of an element in a TTCN-3 file.

To use this feature:

1. Open a TTCN-3 file
2. Place the cursor before the name of a **testcase, type, template, function etc.**
3. Press **Shift+Ctrl+G**, a Search view will open displaying:
   - The project
   - The file
   - The module
   - The number of occurrences of the selected element

Figure 5.13. Find references in workspace



Outline view components

Variables, ports and timers are displayed in the Outline View

Folding support for template fields

TTCN-3 CL Editor supports folding for template fields

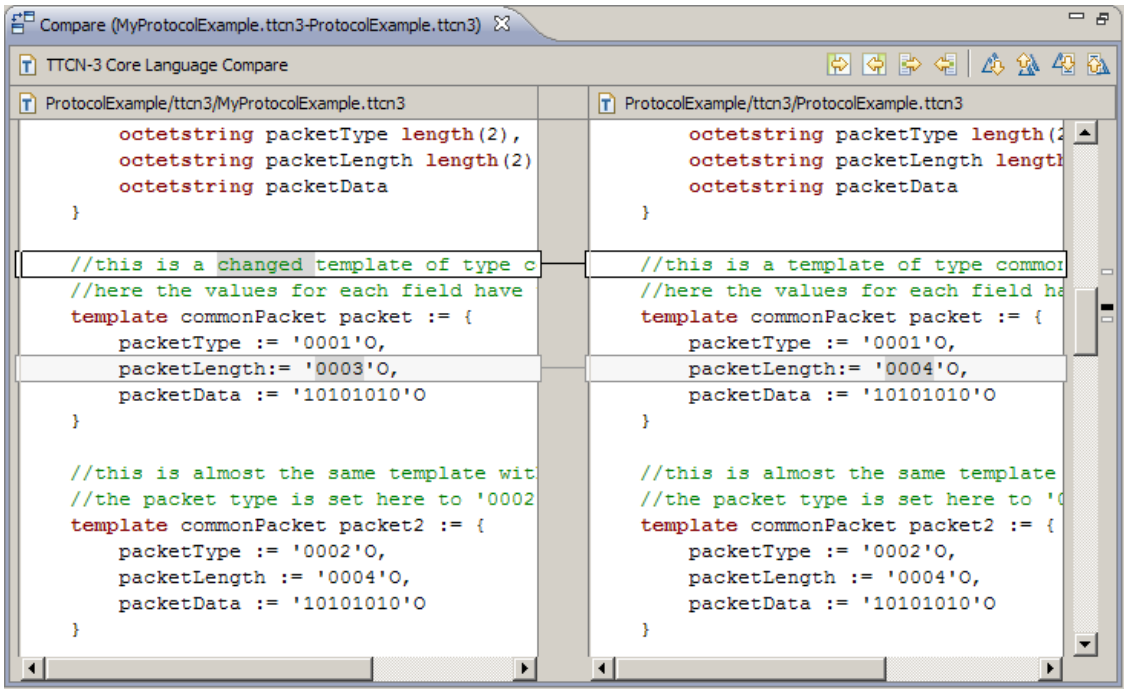
Highlighting of Keywords

The highlighting of keywords supports all TTCN-3 terminals as defined in the supported TTCN-3 Core Language standard, e.g. **module**, **type**, **template**, **component**, etc. In addition, the predefined functions such as **int2oct**, **setverdict** are also highlighted. The color of highlighted syntax can be changed using the editor preferences (see the section called “Syntax Preference”).

Comparing TTCN-3 files

TTCN-3 Core Language files can be compared with other files using Eclipse's standard text file comparison facilities. See (Eclipse) Workbench User Guide, Tasks, Comparing Resources. When using the compare editor the TTCN-3 syntax elements will be highlighted as in the CL Editor itself to maintain a consistent user experience.

Figure 5.14. Comparing a local TTCN-3 file with a revision from source code management



Mark Occurrences

You can find all occurrences of an identifier easily by using the Mark Occurrences feature. Just click on an identifier, and all occurrences will be highlighted with a yellow background.

To toggle this feature, go to the Mark Occurrences page in the CL Editor preferences.

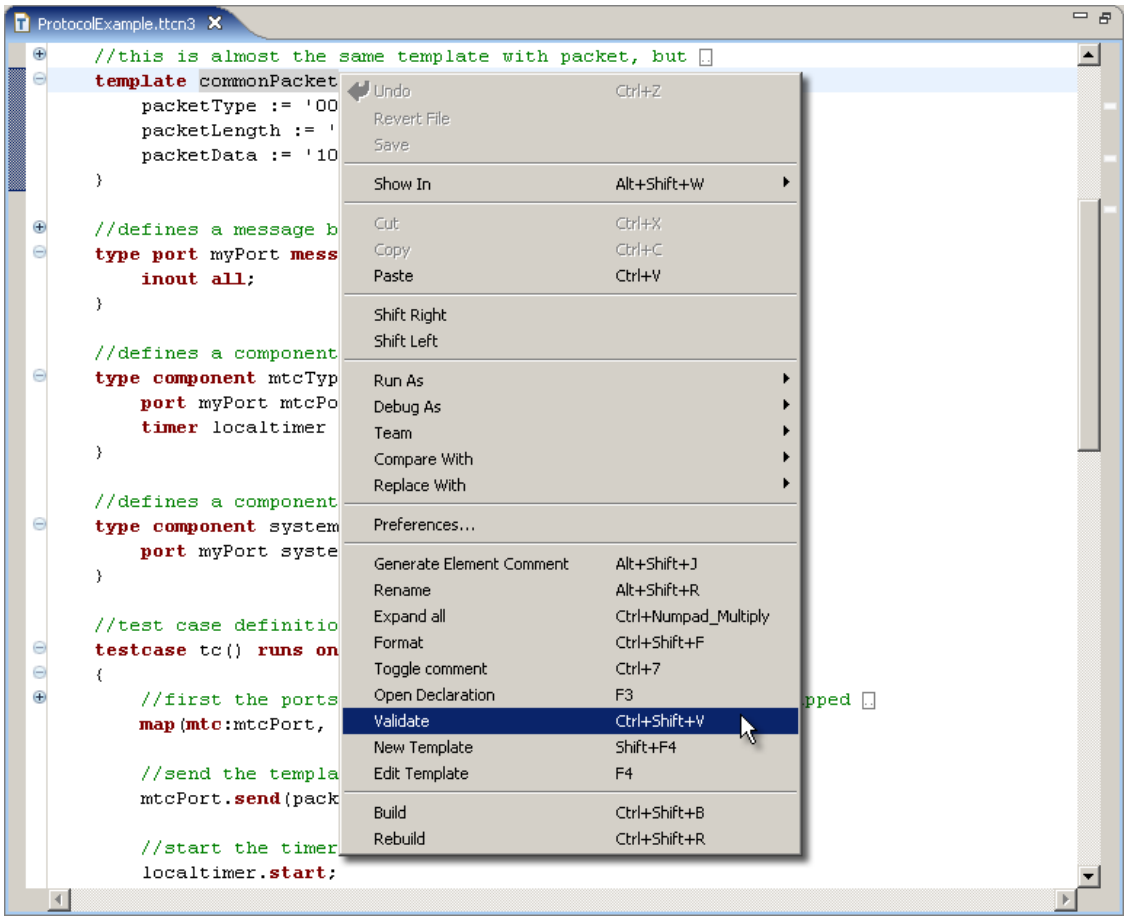
Note

At this point, Mark Occurrences does not work for fields and extensions.

Validation of Syntax

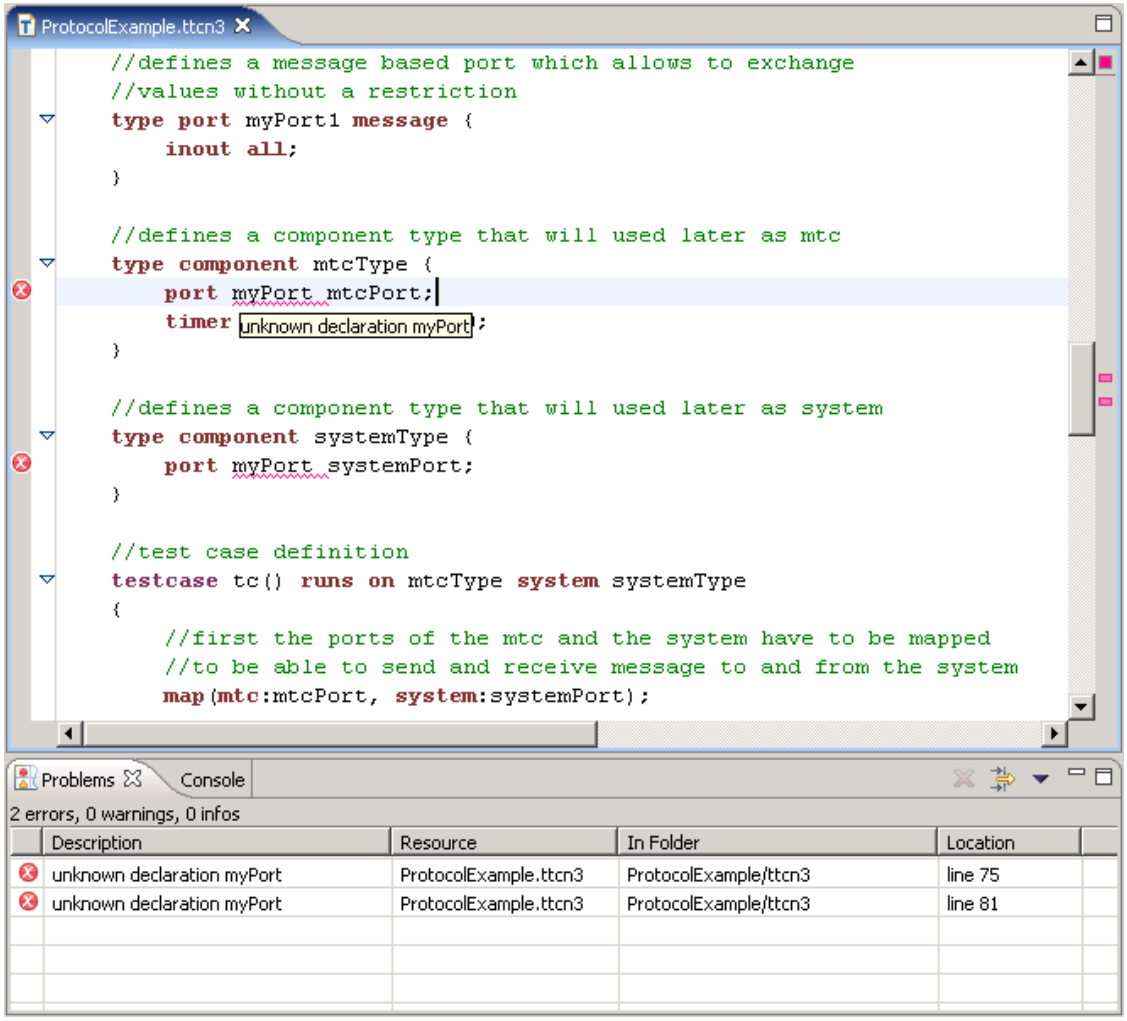
The CL Editor provides comprehensive syntax checking according to the grammar of the TTCN-3 Core Language. The validation of syntax is done automatically on opening or saving of the text file. This feature can be switched on/off by changing the editor mode in the editor preferences (see the section called “General”). A more extensive validation which includes type checking can be initiated using the  button in the window's icon bar, the menu **Source > Validation**, or the context menu **Validate** (by right-click on the editor window as shown in Figure 5.15, “Validate command”), or using the short-key **Ctrl+Shift+V**.

Figure 5.15. Validate command



The progress of syntax checking is illustrated on the progress bar. Errors or warnings found during the validation are reported to the **Problems** view, and visualized using the text annotation, as shown by the figure below. The items in the Problems view and the text annotations are useful to navigate through the test specification.

Figure 5.16. Error reporting



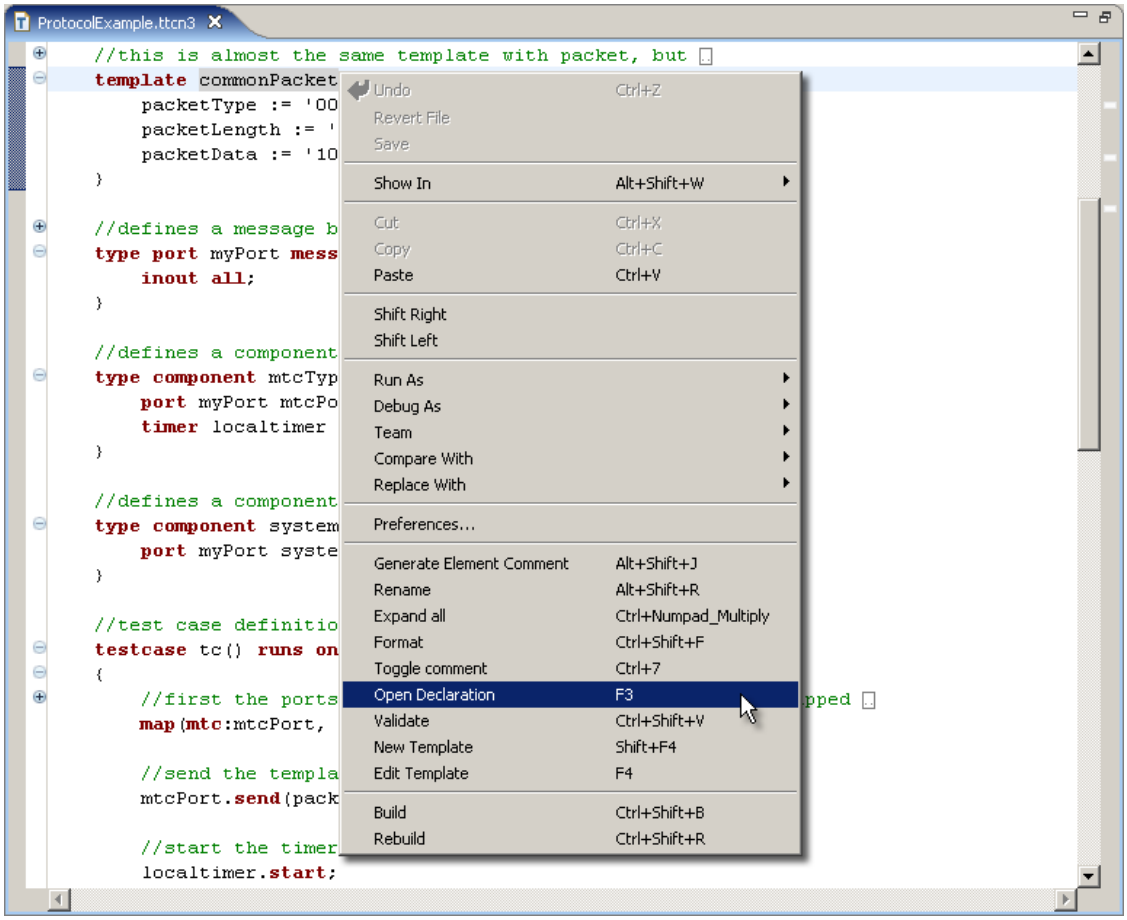
Formatting of Text

Automatic formatting of text is available using the menu **Source > Format**, or the context menu **Format** (by right-click on the editor window), or using the short-key **Ctrl+Shift+F**.

Open Declaration

Navigation of global declarations, e.g. data types, component types, templates, or test cases, is provided by the open declaration action. Doing so, place the cursor on the identifier of interest, and select either **Navigate > Open Declaration**, or the context menu **Open Declaration** (by right-click on the editor window), or the short-key **F3**. The cursor will be placed then at the position of the according declaration.

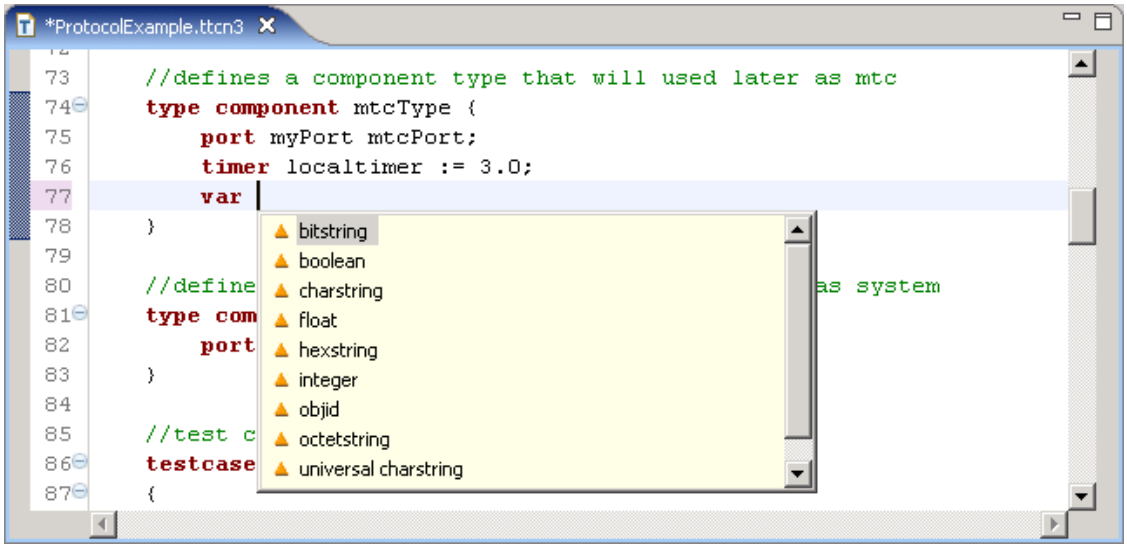
Figure 5.17. Open declaration



Completion Assistance

The completion assistance provides a context dependent selection of identifiers. For example, to declare a variable in a component type as illustrated in Figure 5.18, “Completion assistance”, type **var** and blank, and use **Ctrl+Space** to obtain a list of all defined data types or code templates for choice. Please refer to the section called “Template Preference” for availability of code templates.

Figure 5.18. Completion assistance



Progress Information

Progress information, log messages and debug information etc. that are produced by CL Editor facilities are printed on the standard console. This console is opened automatically if the appropriate environment is set. If not done so, the console can be explicitly opened by **Window > Show View > Basic > Console**.

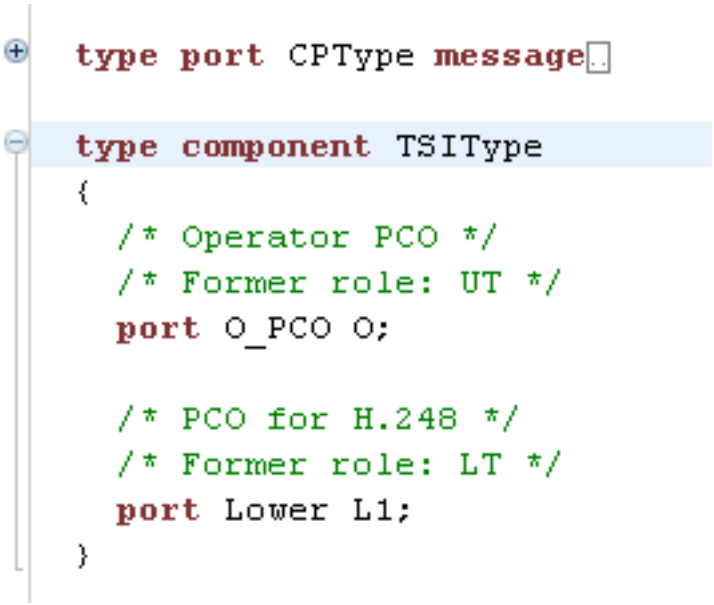
Code Folding

More clarity in your code can be reached by using code folding: Syntactical units of TTCN-3 code are foldable and expandable exactly like in the Eclipse Java editor.

If you put the mouse pointer onto the little '-' button on the left (see figure below), a line appears that shows you the code area that is summarized into one syntactical unit, which's folding is controlled by that particular button.

By pressing the button, the area is folded, which means that only it's first line remains visible and the '-' button changes into a '+' button. By pressing this button again, the area is unfolded and becomes visible again.

Figure 5.19. Code folding



Refactoring Support

The goal of TTCN-3 program refactoring is to make workspace-wide code changes without affecting the behavior of the program. This tooling provides assistance in easily refactoring TTCN-3 code.

When performing a refactoring operation, you can optionally preview all of the changes resulting from a refactoring action before you choose to carry them out. If you do not preview a refactoring operation, the change will be made in its entirety and any resultant problems will be shown after compilation or validation.

Refactoring commands are available from the context menu of the Core Language editor. Many "apparently simple" commands, such as Rename, are actually refactoring operations, since renaming TTCN-3 elements often require changes in dependent files.

Note

Before performing any refactoring operation, it is recommended to save and validate all open TTCN-3 files.

Refactoring Steps

Refactoring With/Without preview

The following instructions will lead you through the refactoring process:

- Activate a refactoring command. For example, rename a template by clicking on it's name and choosing Rename from the CL Editor's pop-up menu (or the Source menu).
- The Refactoring Parameters page prompts you for information necessary for the action. For example, the Rename Refactoring dialog asks you for a new name for the selected template.
- Provide the necessary data on the parameters page, and
  - click OK in order to start the refactoring process, or

- click Next in order to see a preview of the changes

Previewing refactoring changes

The Preview page shows the proposed effects of a refactoring action. You can use this page as follows.

- Select a node in the tree to examine a particular change.
- To examine a change inside a TTCN-3 module, expand a TTCN-3 module node in the tree and select one of its children.
- When selecting nodes, the compare viewer is adjusted only to show a preview for the selected node.
- Clear the checkbox for a node to exclude it from the refactoring.

Note

Excluding a node can result in compile errors when performing the refactoring without further warning.

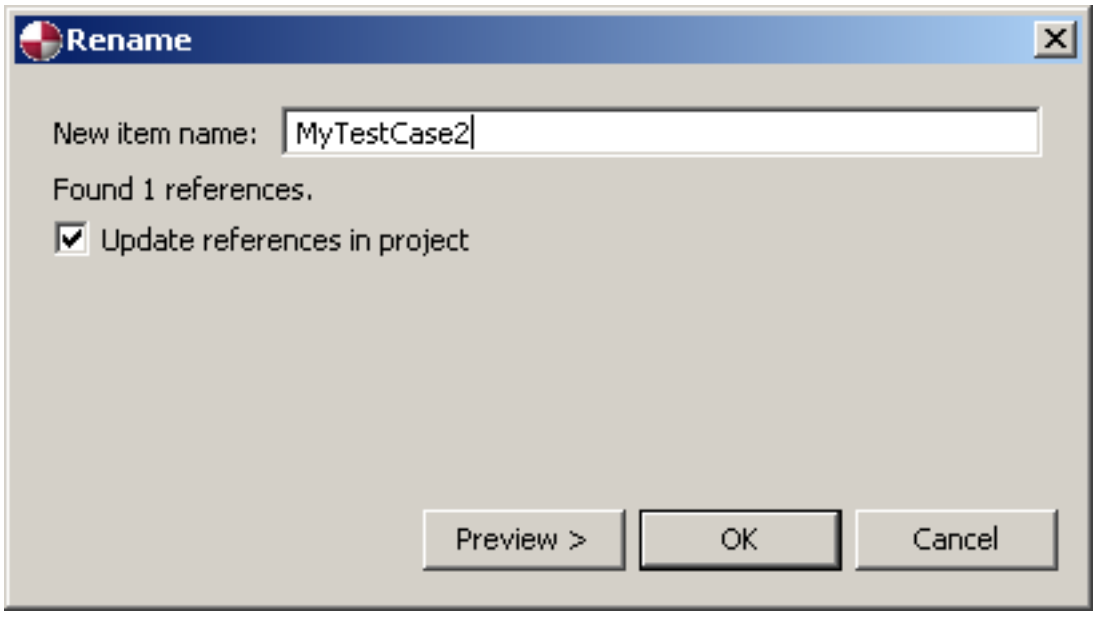
Rename

You can rename a type by modifying its declaration in the TTCN-3 module in which it is declared. However, if you also want to update all references to it, do one of the following:

- In a CL Editor, select a reference.
- From the editor's pop-up menu, select Rename or use the Source > Rename action from the global menu bar.

Parameters page

Figure 5.20. Parameters Page for the Rename Refactoring Command



- In the Enter new name text field, type a new name for the identifier that you're renaming.

- If you do not want to update references to the renamed field, deselect the Update references to the renamed element checkbox.
- Click OK to perform a quick refactoring, or click Preview to perform a controlled refactoring.

Note

References in comments and string literals are not updated.

Note

The rename operation is only allowed on elements of kind: altstep, constant, function, module parameter, template, testcase, type and variable.

Quick Fix

For certain problems or errors, CL Editor can offer corrections. This is indicated by the light bulb shown in the editor marker bar to the left. If the Quick fix is concerned with an error, then the light bulb is overlayed by the red error cross.

To see the correction proposals, use the Quick Fix action:

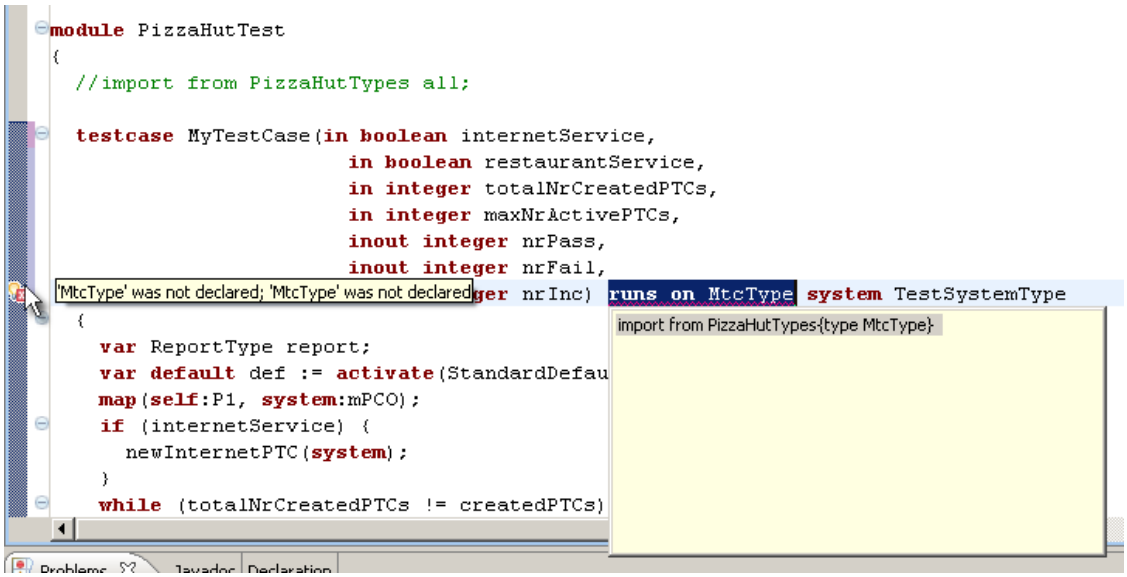
a) set the cursor inside the highlight range, and select **Quick Fix** from the Edit menu or the context menu,

b) set the cursor inside the highlight range, and press **Ctrl**+**1** or

c) click on the light bulb.

Quick fix is also available on a Quick fixable problem entry in the Problem view. The Quick Fix action will open a dialog to select the correction.

Figure 5.21. Quick Fix

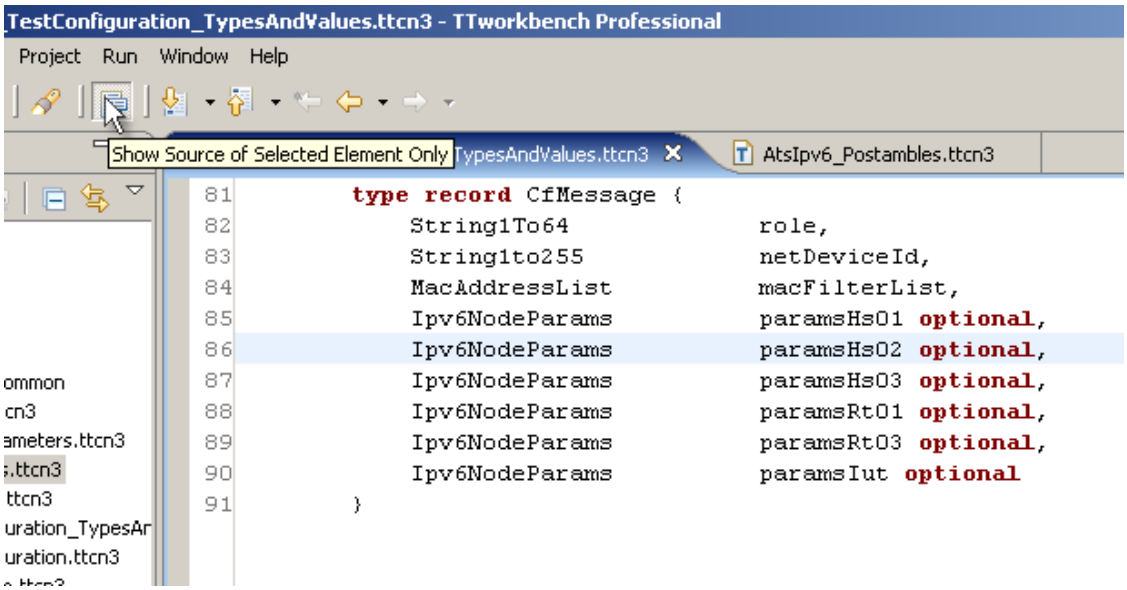


Showing single element or whole TTCN-3 file

- To display the selected TTCN-3 file in a single element view, click the Show Source of Selected Element Only button in the workbench toolbar, so that it is pressed.

- To display the selected TTCN-3 file in a whole (non-segmented) view, click the Show Source of Selected Element Only button in the workbench toolbar, so that it is not pressed.

Figure 5.22. Show Source of Selected Element Only



Note

This toolbar button is enabled only when a TTCN-3 editor is open.

Task tags inside TTCN-3 comments

- Task tags can be configured on the TTCN-3 > CL Editor > Task Tags preference page. When the tag list is not empty, the compiler will issue a task marker whenever it encounters one of the corresponding tag inside any comment in Java source code. Generated task messages will include the tag, range until the next line separator, comment ending or non-empty tag, and will be trimmed. If the same line of code carries multiple tags, they will be reported separately.
- See the CL Editor preference page for information on how to enable task tags in your source code.
- See Tasks and Markers for details.

Note

This feature is activated only for TTCN-3 projects having the TTCN-3 Compiler Nature enabled

Template Wizard

Purpose

To ease creation and modification of TTCN-3 templates a Template Wizard has been integrated into CL Editor. It is a graphical interface for certain elements while editing TTCN-3 Core Language. It can be accessed from the Edit menu, the editor context menu or via the shortcuts:

- **F4** (Edit Template) or
- **Shift**+**F4** (New Template).

Editing an existing TTCN-3 template

- Set the cursor inside the identifier of a template to be edited in the TTCN-3 source code.

Figure 5.23. Template in Core Language

```
template CommandRequest cadd_1tid_s(
    in template TerminationID loc_tid1,
    in template MediaDescriptor loc_mediaDescr) := {
    command := {
        addReq := {
            terminationID := { loc_tid1 },
            descriptors := { { mediaDescriptor := loc_mediaDescr }
            }
        }
    },
    optionalField := omit,
    wildcardReturn := omit
}
```

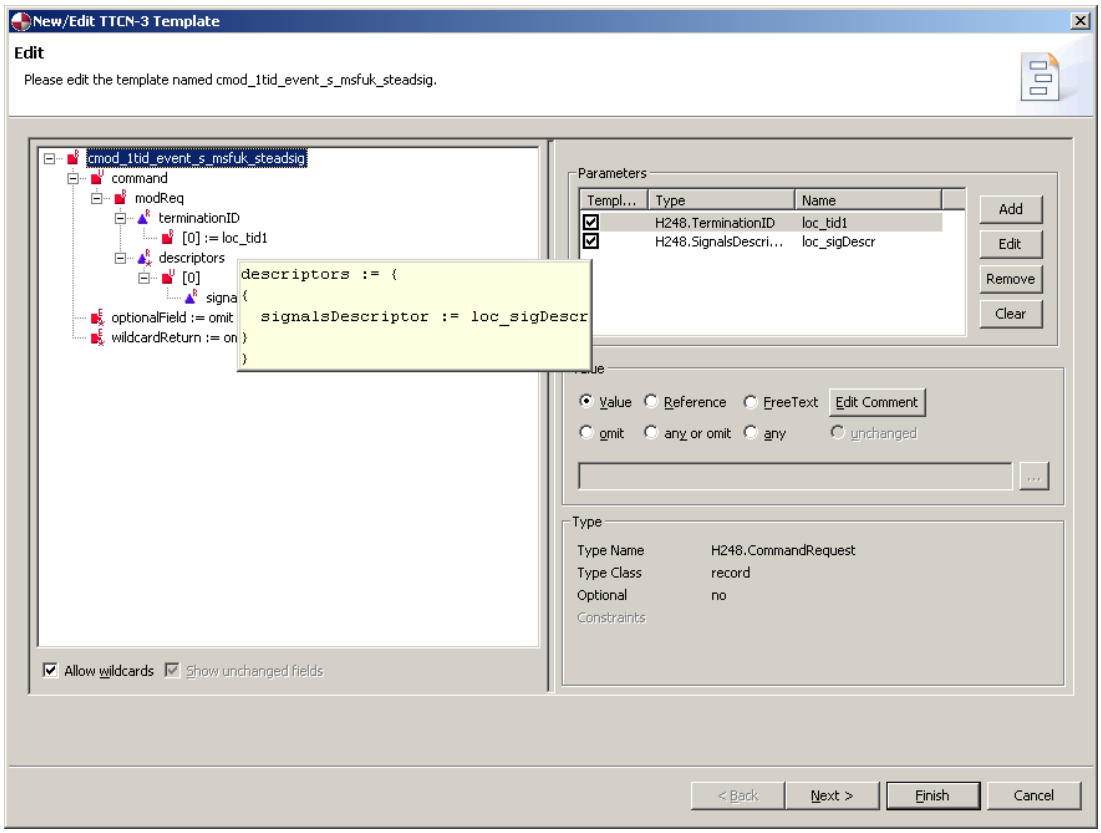
- Choose Edit > Edit Template.

  Alternatively you can press **F4** or use the context menu (Right-Click).

- In the Edit page of the Template Wizard appearing change and set values of the template.

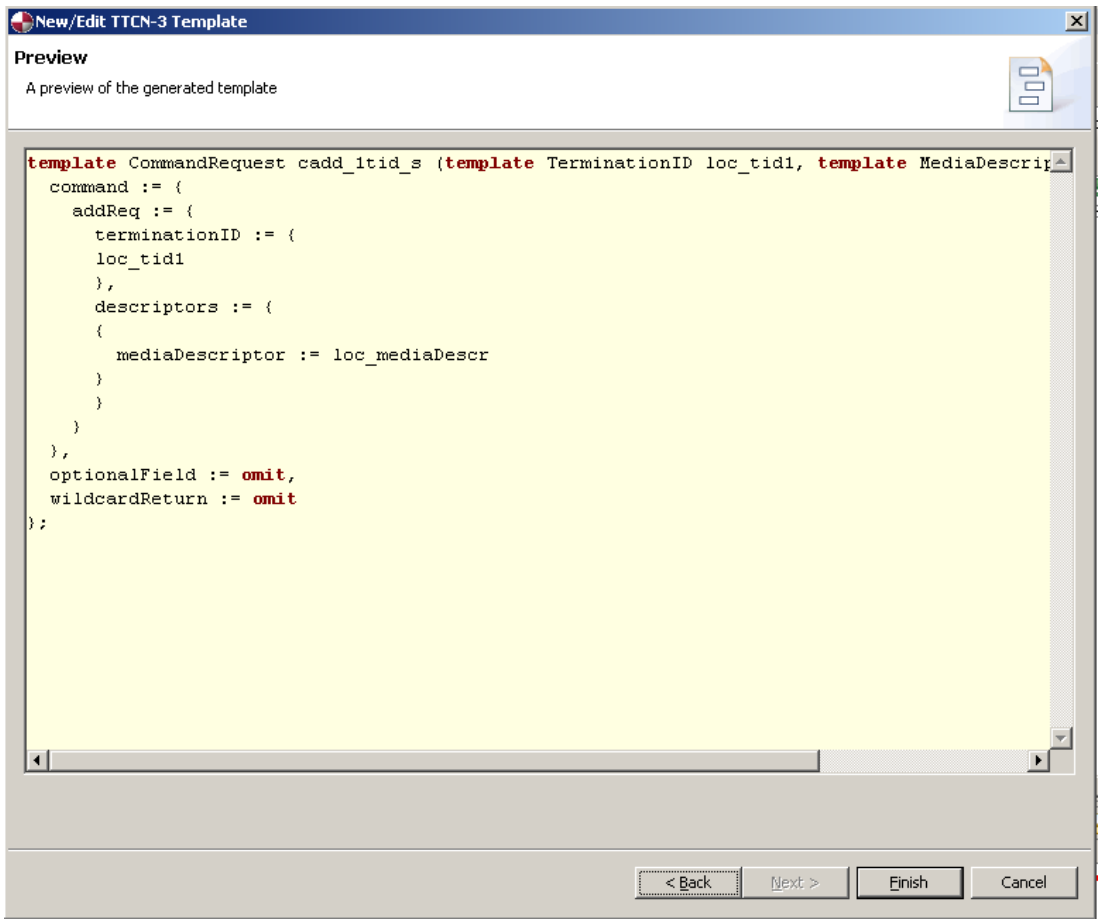
  In the left pane of this page the template's structure is shown. Select elements to be edited here. The right hand side has the following areas: the parameters of the template, the value of the selected element, and its type information. Within the the value area it is possible to edit a comment for the template by clicking the button "Edit Comment".

Figure 5.24. Edit page of the Template wizard



- The Next button leads to the preview page, where the source code of the created template can be examined. Switch back for further editing.

Figure 5.25. Preview page of the Template wizard



- To save the template back into the source code click Finish on the preview page.

Creating a new template based on a type in the same module

- Set the cursor inside the identifier of the type the new template should be created from.
- Choose Edit > New Template

  Alternatively you can press **Shift**+**F4** or use the context menu (Right-Click).
- The Template Generation Options page of the wizard appears. Here the name of the new template can be defined. It is also possible to choose a different TTCN-3 type to derive the new template from.

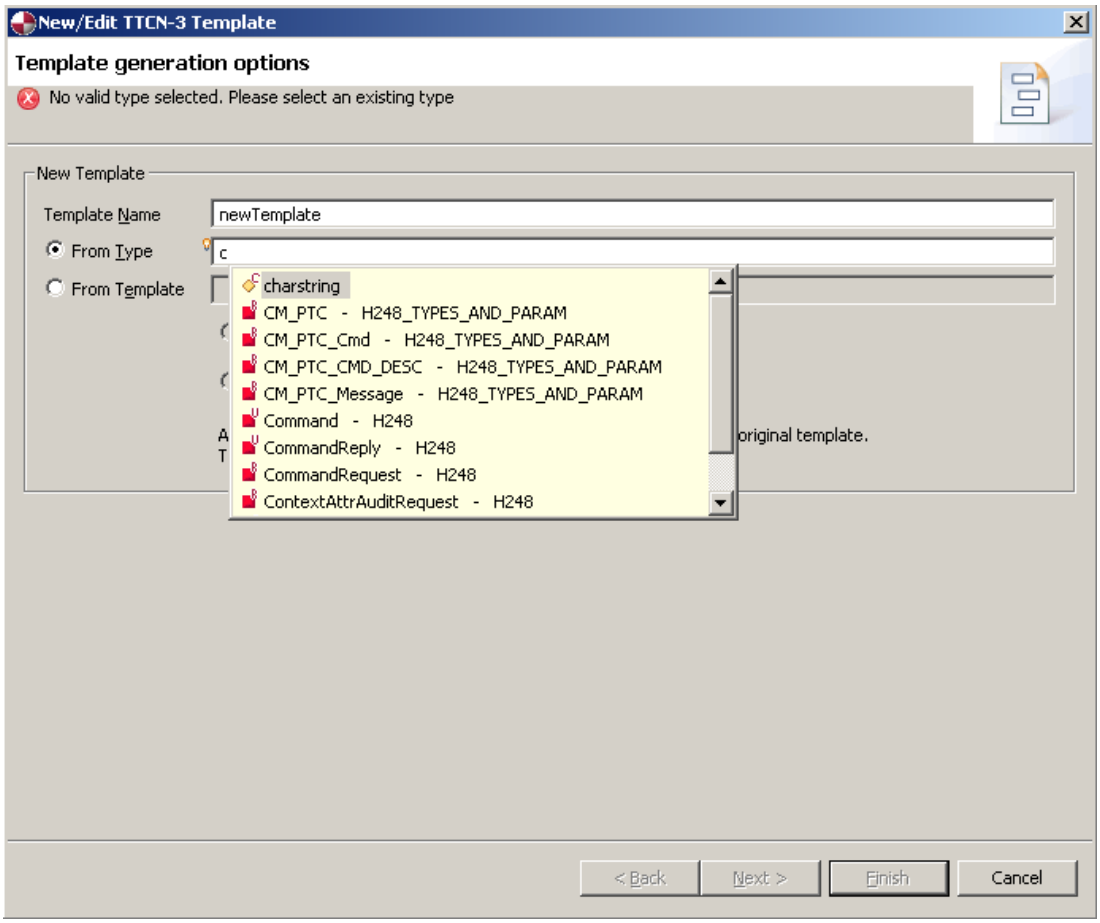
Creating a new template from a type defined elsewhere

- Set the cursor to the position where the new template should be inserted.
- Choose Edit > New Template

  Alternatively you can press **Shift**+**F4** or use the context menu (Right-Click).
- The Template Generation Options page of the wizard appears. The Template Name and From Type fields are empty and should be filled now in order to be able to go further.

In the From Type field there is content assist available. To choose from a list of all known types, hit **Ctrl**+**Space** and select the desired one. Start typing the beginning of the desired type to eliminate improper choices.

Figure 5.26. Content Assist for From Type in the Template wizard



Checking of range constraints

For language elements of types in the following list, range constraints will be checked against the type's definition:

- bitstring
- boolean
- charstring
- enumerated
- float
- hexstring
- integer

- octetstring
- verdicttype

For other types the user is responsible for observing range constraints when inserting values.

Known limitations of validation

The current implementation has certain limitations when validating values. Values of the TTCN-3 types listed below cannot be validated. In these cases values should be inserted as they shall appear in Core Language and should be marked as FreeText.

- address
- anytype
- objid
- universal charstring

Note

Values inserted as FreeText will not be validated by the wizard. In those cases the user has to make sure valid values has been entered.

Preferences

The properties of CL Editor can be parameterized using the CL Editor preferences. The preferences are presented as several preference pages under the CL Editor preference category (**Window > Preferences > TTCN-3 > CLEditor**).

In the following, the CL Editor preference pages for validation, syntax, typing and TT3 plugins are introduced.

General

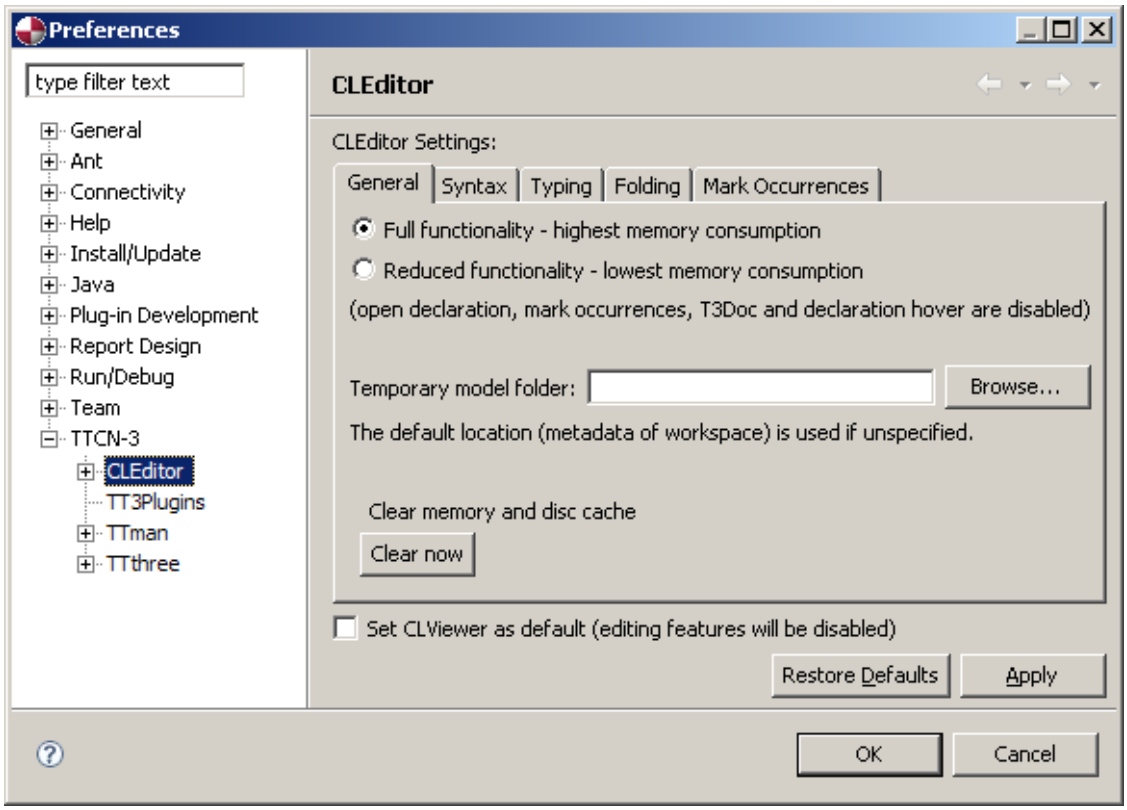
The General page allows to configure the memory handling of the CL Editor. First you can select one of two editing modes:

- **Full functionality:** if selected (default), the CL Editor runs with all supported features.
- **Reduced functionality:** if selected, some of the CL Editor's feature are disable to reduce the memory usage. The deactivated features are Open declaration (see the section called “Open Declaration”), Mark occurrences (see the section called “Mark Occurrences”), T3Doc (see the section called “TTCN-3 Documentation Generation (T3Doc)”) and declaration hover.

Temporary model folder: During the validation temporal data is produced. The data will be deleted on closing of the text file. The user can specify a preferred folder for the temporal data, e.g. `c:\tmp` for windows, or `/tmp` for Linux. If unspecified, the default location (`.metadata/.ttcn3`) is used.

Clear memory and disc chache: Immediately removes all currently cached TTCN-3 content and sets the according memory and disc space free. This does not affect your edited data!

Figure 5.27. CL Editor General preference page

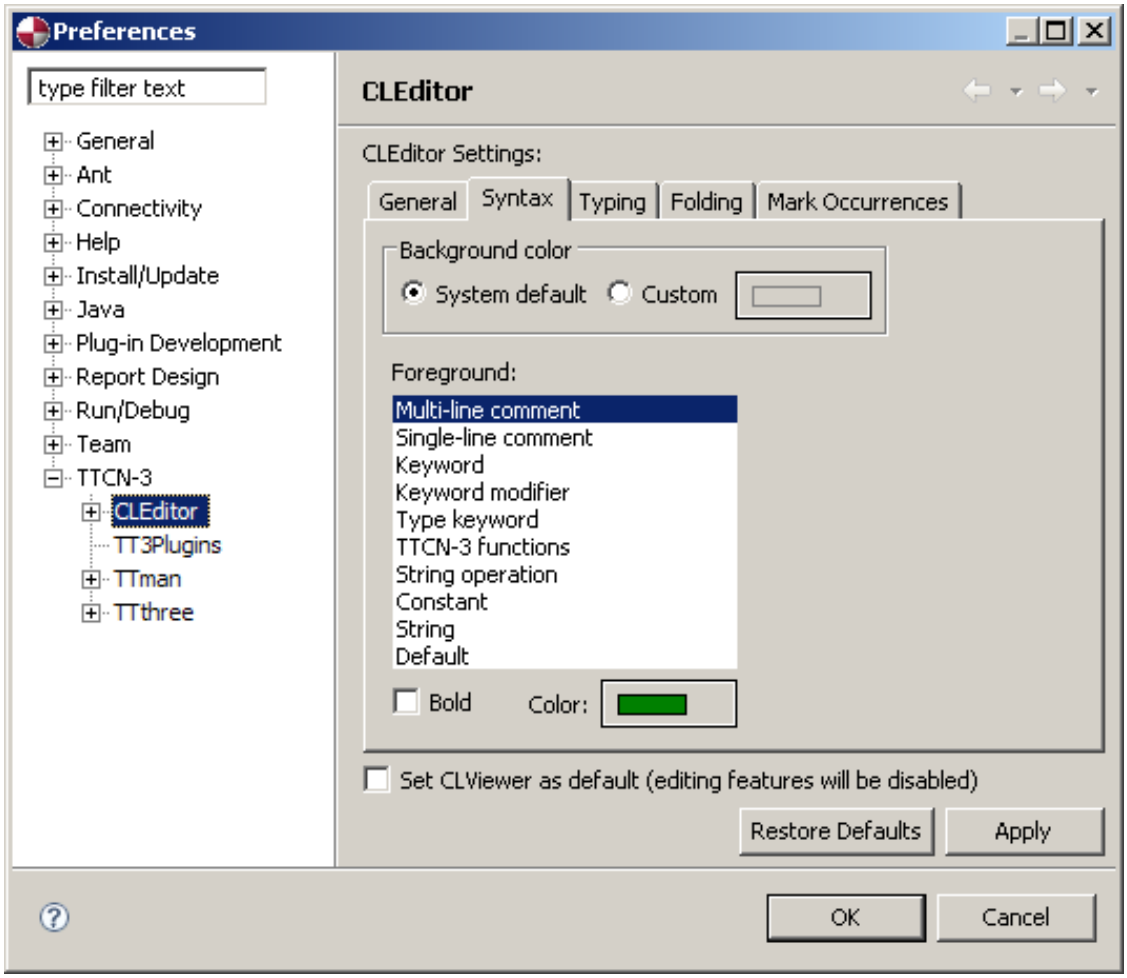


Syntax Preference

The Syntax page is used to specify the foreground and background color for different types of text fragments:

- Multi-line comment
- Single-line comment
- Keyword
- Keyword modifier
- Type keyword
- TTCN-3 functions
- String operation
- Constant
- String
- Default

Figure 5.28. CL Editor Syntax preference page

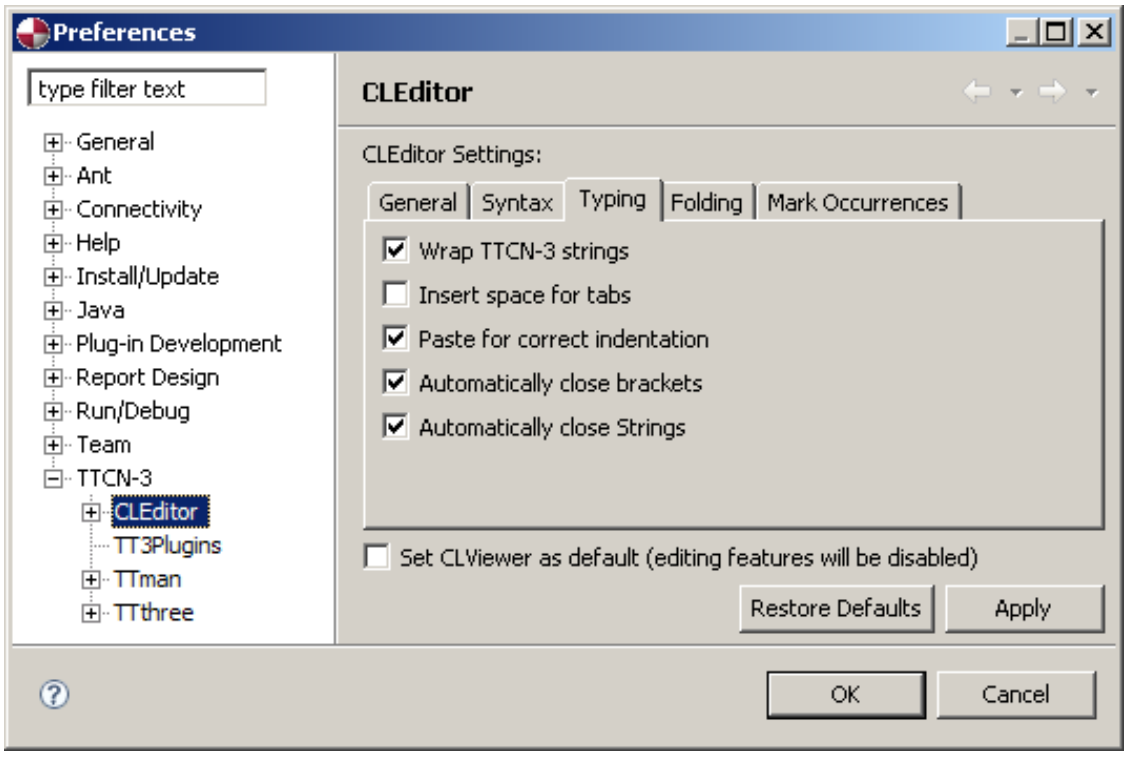


Typing Preference

The Typing page provides the following selections for typing text:

- Wrap TTCN-3 strings
- Insert space for tabs
- Past for correct indentation
- Automatically close brackets
- Automatically close Strings

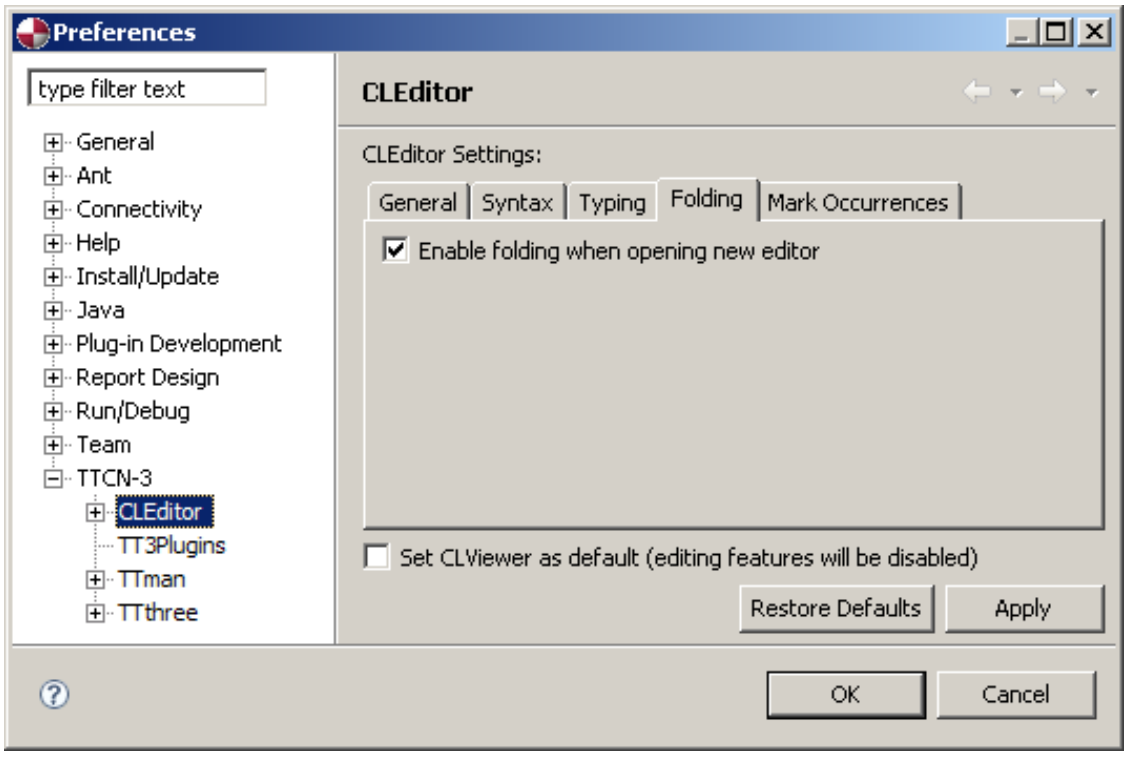
Figure 5.29. CL Editor Typing preference page



Folding Preference

The Folding page is used to enable or disable the folding feature on opening of new instance of CL Editor

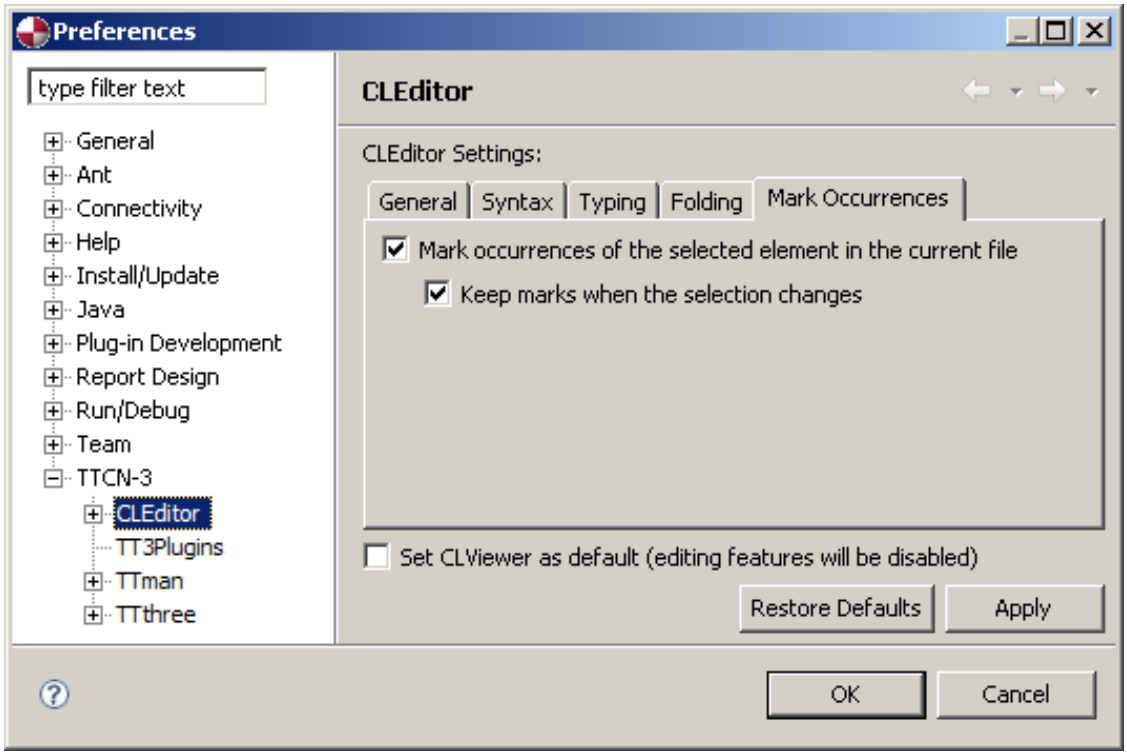
Figure 5.30. Folding preference page



Mark Occurrences

The Mark Occurrences page allows you to set if the CL Editor will automatically highlight references of the currently selected element (see the section called “Mark Occurrences”). If "Keep marks when the selection changes" is set, the reference marks will still be shown if the selection has been removed, but no other element that would cause reference highlighting was selected instead.

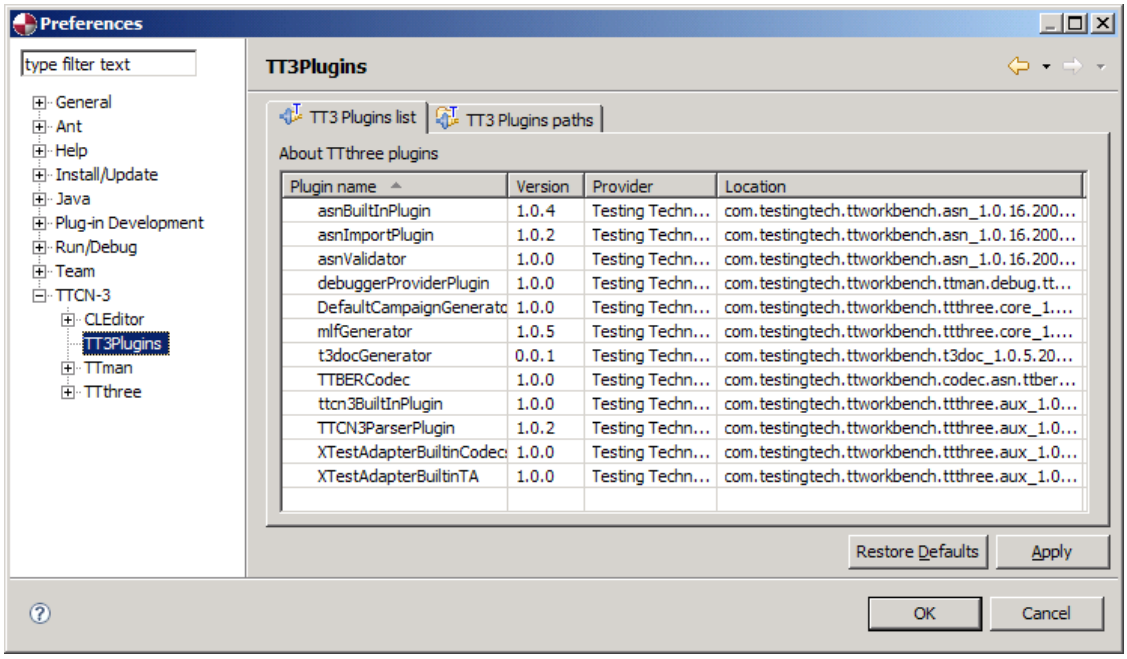
Figure 5.31. Mark Occurrences preference page



TT3 Plugins Preference

The TT3 Plugins page provides information about TTthree plugins that are found in locations specified in the main TT3 plugins preference page (see also Figure 3.1, “TTCN-3 main preferences”).

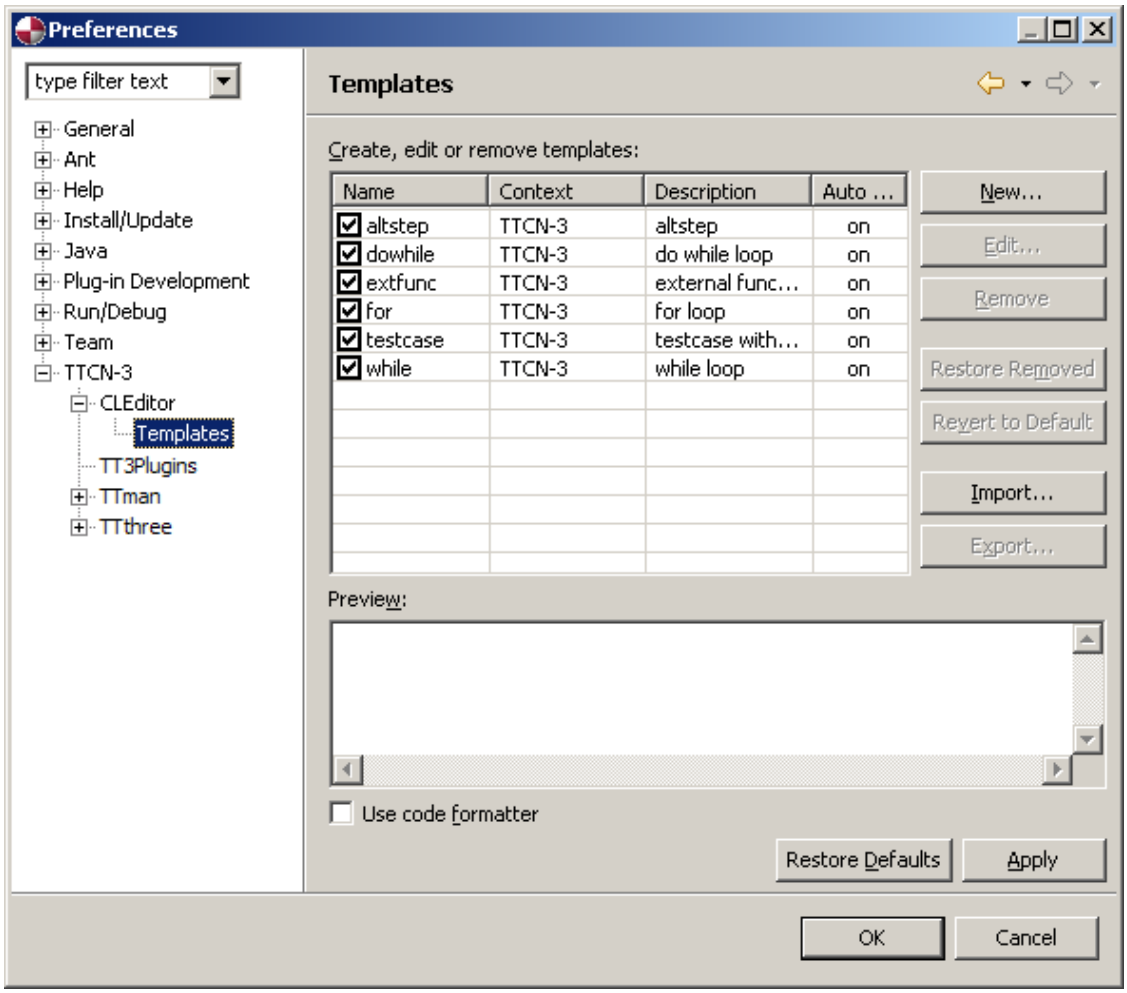
Figure 5.32. CL Editor TT3 plugins preference page



Template Preference

This page (Figure 5.33, “CL Editor Template preference page”) provides the selection and deselection of code templates used by completion assistance (see also the section called “Completion Assistance”).

Figure 5.33. CL Editor Template preference page



Chapter 6. Using TTworkbench GFT Editor

The GFT Editor is a graphical TTCN-3 editor for user-friendly graphical test specification and documentation. This feature is included within TTworkbench Professional, but not in TTworkbench Basic.

GFT Basics

Language Concepts

GFT represents graphically the behavioral aspects of TTCN-3 like the behavior of a test case or a function. This means a GFT diagram provides a graphical presentation of either:

- the control part of a TTCN-3 module
- a TTCN-3 test case
- a TTCN-3 function
- a TTCN-3 altstep

GFT does not provide graphics for data aspects like declaration of types and templates. TTCN-3 definitions and declarations without a corresponding GFT presentation may be presented in the TTCN-3 core language. TTCN-3 core language modules can be imported into GFT Editor. The referenced identifiers can be used afterwards for editing GFTs.

No graphical representation is available for:

- import definitions
- type definitions
- signature declarations
- template declarations
- constant declarations
- external constant declarations
- external function declarations

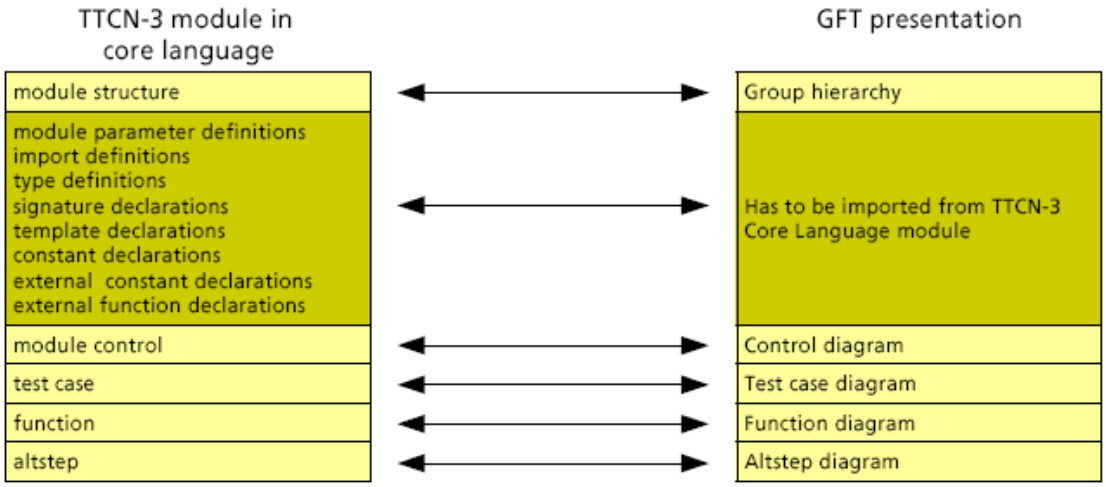
GFT defines no graphical representation for the structure of a TTCN-3 module, but GFT Editor provides respective means to define groups and group hierarchies.

Mapping between GFT and TTCN-3 Core Language

GFT provides graphical means for TTCN-3 behavior definitions. The control part and each function, altstep and test case of a TTCN-3 core language module can be mapped onto a corresponding GFT diagram and vice versa. This means:

- the module control part can be mapped onto a control diagram and vice versa
- a test case can be mapped onto a test case diagram and vice versa
- a function in core language can be mapped onto a function diagram and vice versa
- an altstep can be mapped onto an altstep diagram and vice versa

Figure 6.1. Relation between TTCN-3 Core Language and GFT



Note

GFT provides no graphical presentations for definitions of module parameters, types, constants, signatures, templates, external constants and external functions in the module definitions part. These definitions may be presented directly in core language. Each declaration, operation and statement in the module control and each test case, altstep or function can be mapped onto a corresponding GFT representation and vice versa. The order of declarations, operations and statements within a module control, test case, altstep or function definition is identical to the order of the corresponding GFT representations within the related control, test case, altstep or function diagram.

Note

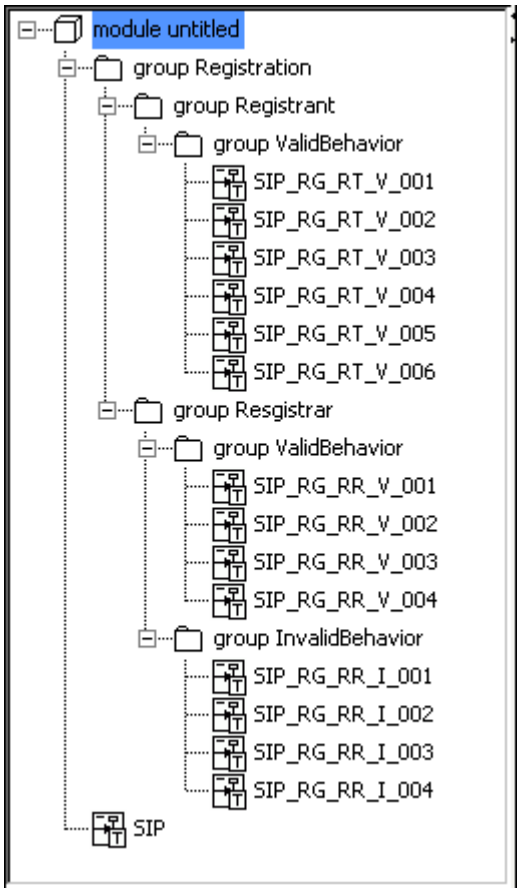
The order of GFT constructs in a GFT diagram is defined by the order of the GFT constructs in the diagram header (declarations only) and the order of the GFT constructs along the control instance (control diagram) or component instance (test case diagram, altstep diagram or function diagram).

Module Structure

A TTCN-3 module has a tree structure. TTCN-3 module is structured into a module definitions part and a module control part. The module definitions part consists of definitions and declarations that may be structured further by means of groups.

GFT provides diagrams for all "behavioral" leaves of the module tree structure, i.e. for the module control part, for functions, for altsteps and for test cases. GFT defines no concrete graphics for the module tree-structure, however GFT Editor provides the module structure in form of a navigator tree.

Figure 6.2. TTCN-3 Module Tree Structure in GFT Editor



GFT Editor UI and Workflow

Designing test cases graphically with GFT Editor, its possible to create test cases, functions, altsteps, and module controls. When the module is complete, it can be exported in a TTCN-3 core language file, if it should be used with a TTCN-3 compiler. For documentation purposes the diagrams can also be exported as GIF files.

About the GFT Editor Working Environment

Opening GFT Editor for the first time displays a menu bar and button bar across the top of the screen, a navigator tree on the left side, a drawing area in the center with tools palette on the left side of it, and a multitabbed panel on the bottom. After opening a document, GFT Editor places the diagram windows in the middle.

Displaying a GFT diagram

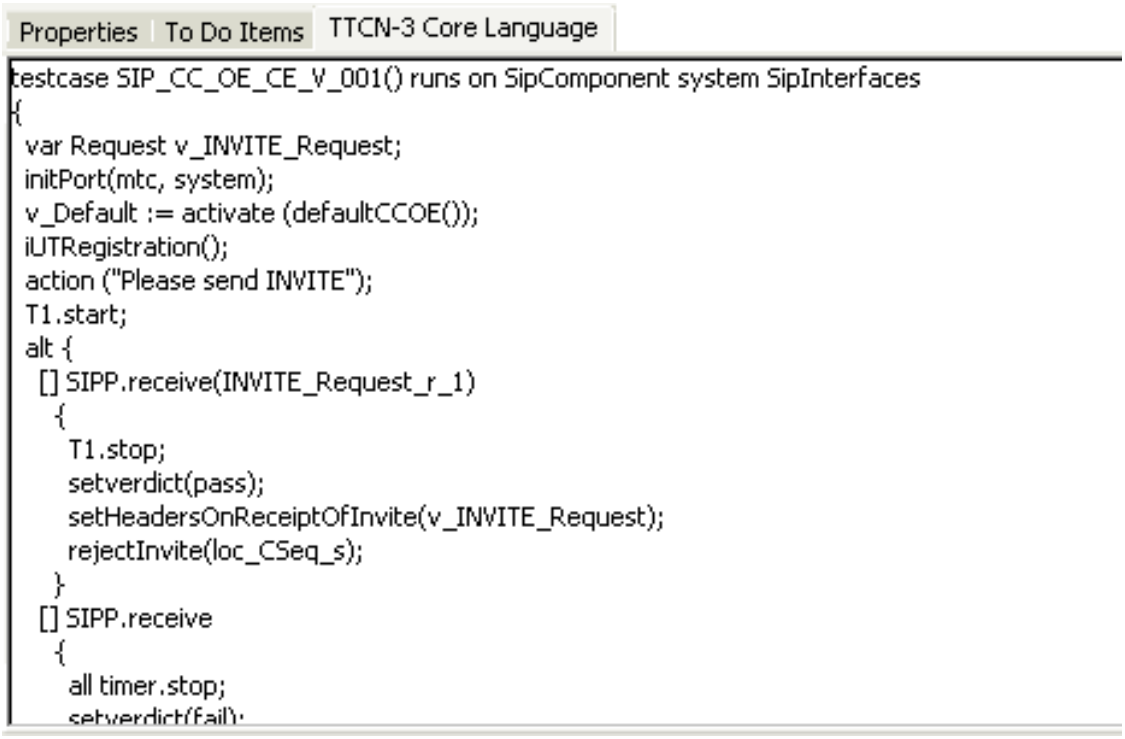
Selecting a diagram in the navigator tree, displays the respective diagram. In the diagram window, the title bar displays the diagram name.

Selecting a symbol in the navigator tree, that is part of a certain diagram, the respective diagram will be displayed. In this case the respective symbol is selected in the diagram, otherwise the diagram frame itself is selected. If the respective diagram is iconified or completely not visible it will be displayed using its last size and location.

Displaying Core Language

Every diagram and symbol in GFT has its counterpart in the TTCN-3 core language. The corresponding core language can be shown by selecting a diagram or symbol in the drawing area or in the navigator tree and clicking on the TTCN-3 Core Language tab on the bottom.

Figure 6.3. Core Language Tab



Displaying and Choosing Tools

The tools palette contains tools to create, select, and edit symbols in a certain diagram. Clicking on a tool will choose it.

Figure 6.4. Tools Palette

Note

The symbols name will be shown by holding the pointer over the tool icon in the tools palette until the tooltip displays.

The tools palette can be removed from the drawing area just by toggling the Drawing Tools button  in the button bar, Drawing Tools menu, or via **Ctrl**+**D**.

Editing Symbol Attributes

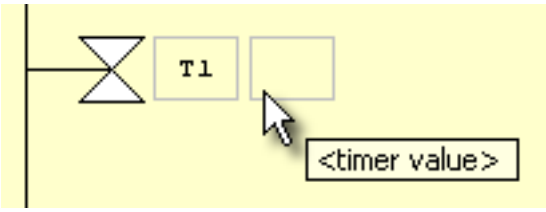
GFT symbols have attributes that can be edited, e.g. a name. In general GFT Editor provides the following ways to edit symbol attributes. For detailed description on specific symbols and their attributes, please have a look at the section called “GFT Symbols”.

GFT Attributes can be edited in two ways, directly on the symbol in the drawing area, or at the property tab on the bottom.

... in the Drawing Area

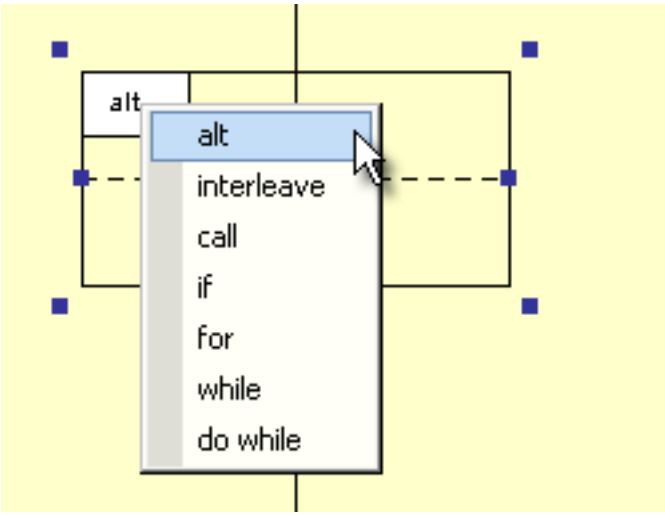
While moving the mouse over a symbol in the drawing area gray rectangles will be displayed at the places where attributes of the respective symbol may be edited. A tool tip will explain the meaning of that attribute.

Figure 6.5. Tool Tips on Symbol Attributes



Left-click in the gray box will pop-up a list or editor to change the respective attribute.

Figure 6.6. List of Attributes

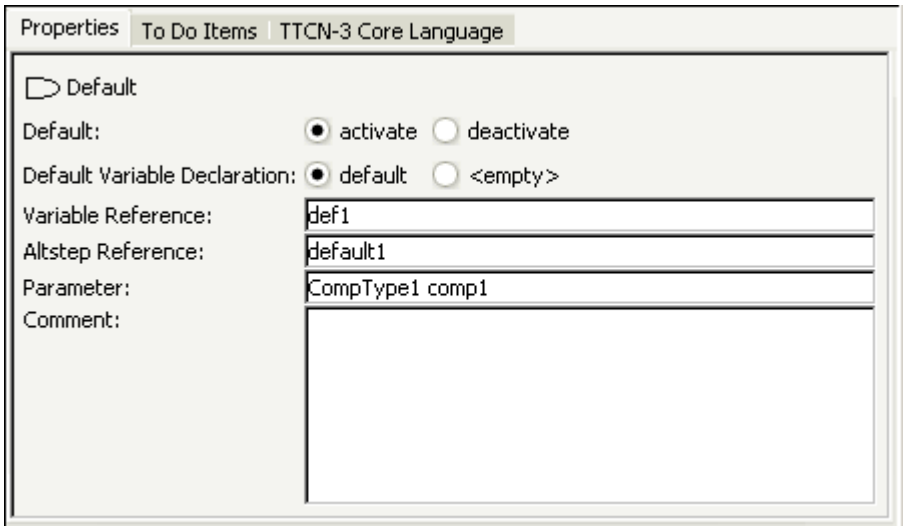


... in the Property Panel

Besides the direct editing of symbol attributes in the drawing area, it is also possible to edit them in the respective property panel.

To do so, select the respective symbol in the drawing area or in the navigator tree. Left-click on the properties tab on the bottom and edit as desired.

Figure 6.7. Property Panel



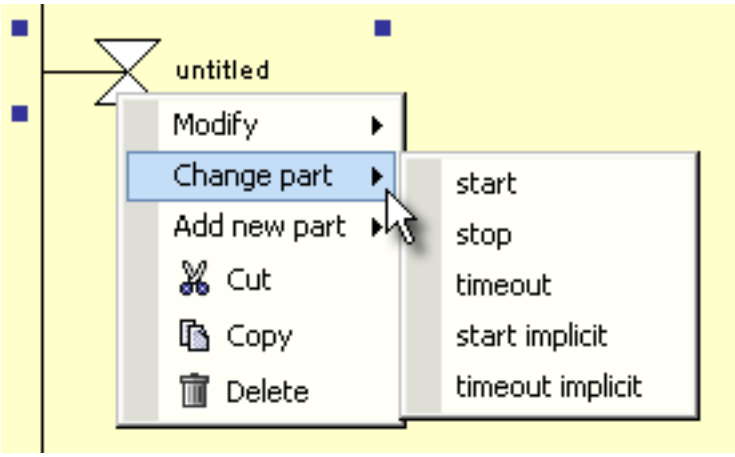
Note

It is possible to add a comment to every symbol. However the comment attribute can be edited only via the property panel.

Using Context Menus

Context menus let you quickly access commands that are relevant to the current selection. To display a context menu: Right-click a selected symbol in the drawing area.

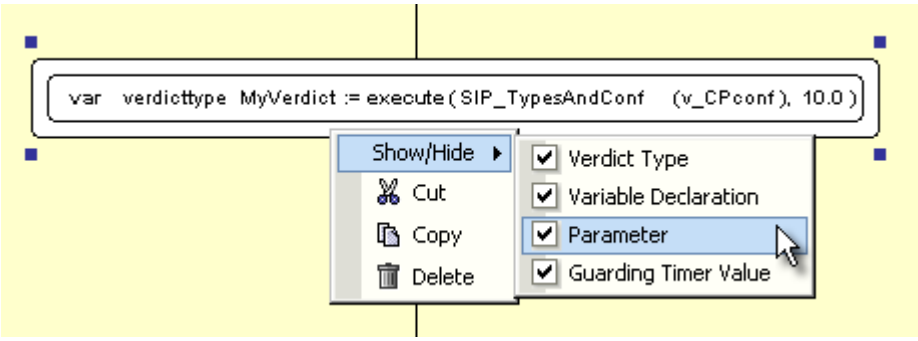
Figure 6.8. Symbol's Context Menu



Showing/Hiding Optional Symbol Attributes

For a better view it is possible to show or hide optional attributes of a symbol. To do so right-click on a symbol in the drawing area, select Show / Hide from the popup menu and toggle the respective checkbox.

Figure 6.9. Show / Hide Optional Symbol Attributes



Note

Hiding an attribute will delete the value of this attribute.

Another way to show hidden attributes is to edit them in the respective property panel.

In some cases there is a dependency between the display of two attributes.

Go to/Import Referenced Diagram

It is possible to jump to a referenced test case, function or altstep diagram. To do so, right-click on a reference , start symbol , execution or activation in the drawing area and select Go to Diagram from the popup menu.

This menu item is only available, if there exists another diagram with the referenced name in the current module, otherwise it's possible to import the referenced diagram by selecting Import Referenced Diagram from the popup menu.

Note

After loading a saved GFT file, the corresponding TTCN-3 file has to be selected, if case of importing a referenced diagram from the module.

Undoing and Repeating Multiple Actions

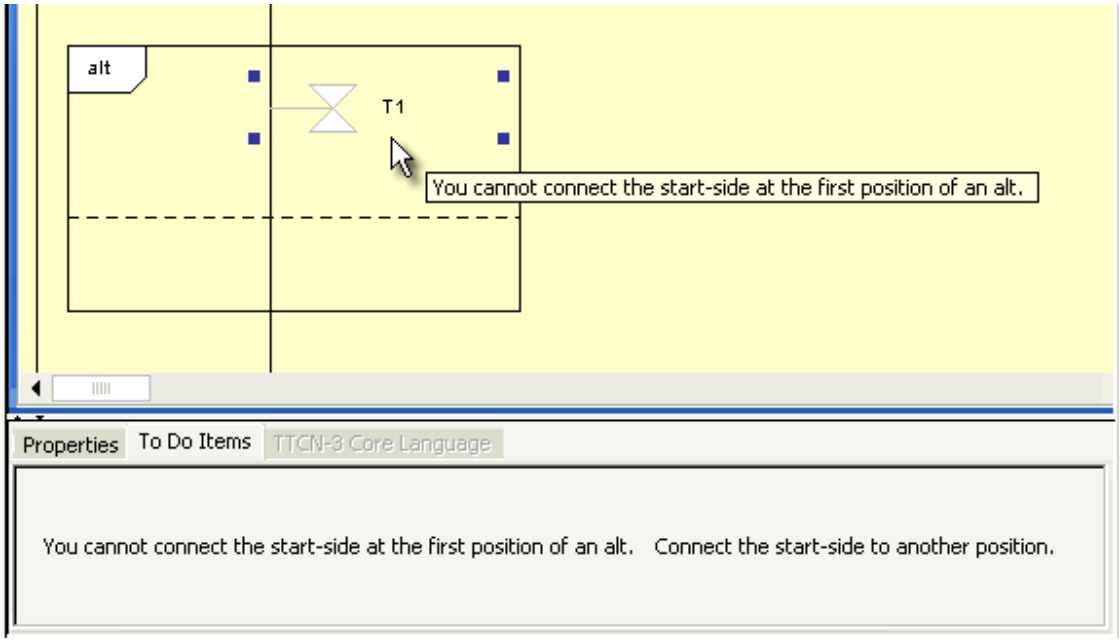
Within GFT Editor it is possible to undo and afterwards redo every action the user has done up to a number of 20. For undo use Undo button in the button bar, Undo menu, or just hit **Ctrl**+**Z**. For redo use Redo button in the button bar, Redo menu, or just hit **Ctrl**+**Y**.

To Do Support

GFT Editor has a built-in "To Do" support, in order to support the specification of correct GFT diagrams. In the case a symbol is not well positioned or the attributes of a symbol are not specified properly, the symbol gets grayed-out.

A hint with an explanation of the reason will be display at the tooltip. In addition the reason and a suggestion for a solution will be available on the respective To Do panel on the bottom.

Figure 6.10. To Do Item Panel



Importing Definitions

As GFT provides no graphical representations for definitions of module parameters, types, constants, signatures, templates, external constants and external functions, these definitions may be imported directly from TTCN-3 core language modules.

This is expressed as an import statement, which constitutes a permanent unidirectional relationship from the current GFT file to the core language module. Later, when a GFT file is used to produce an executable test suite, that relationship will also be used.

Note

This function doesn't convert a TTCN-3 core language module into a GFT module (see the section called “Importing TTCN-3 Core Language”).

Note

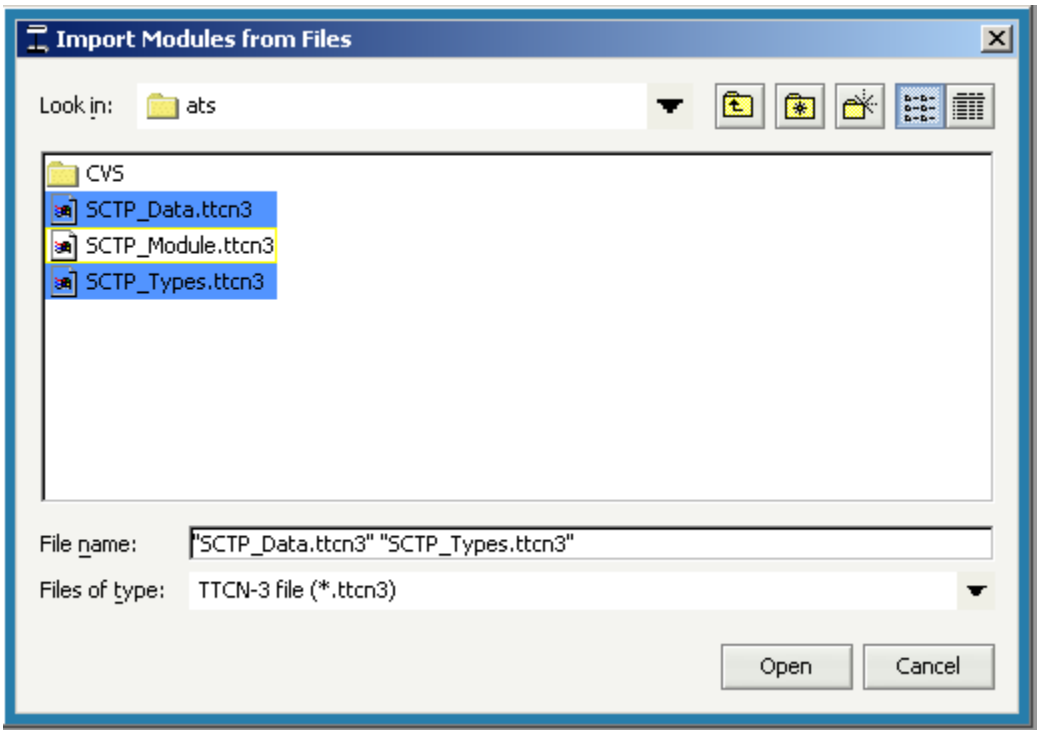
GFT Editor stores only a file reference to the module, not the imported definitions itself. So when opening a GFT file, all utilized module files must be present, because GFT Editor will re-import their definitions.

Steps for Creating an Import Statement

In order to import definitions from a TTCN-3 module:

1. Choose Import Modules... menu, or just hit **Alt**+**I**
2. Choose one or several TTCN-3 module(s) from the file system.

Figure 6.11. Choosing Multiple TTCN-3 Module Files for Import.



Note

GFT Editor supports importing from native core language files (`*.ttcn3`) as well as importing from pre-compiled TTCN-3 modules (`*.jar`).

For each chosen file an import statement will be generated in the navigator panel and an overview of all imported definitions will be shown in a window. Those imported references will now become available for selection when drawing a diagram with GFT Editor.

3. Change properties as needed via the property panel:

   - File Name:

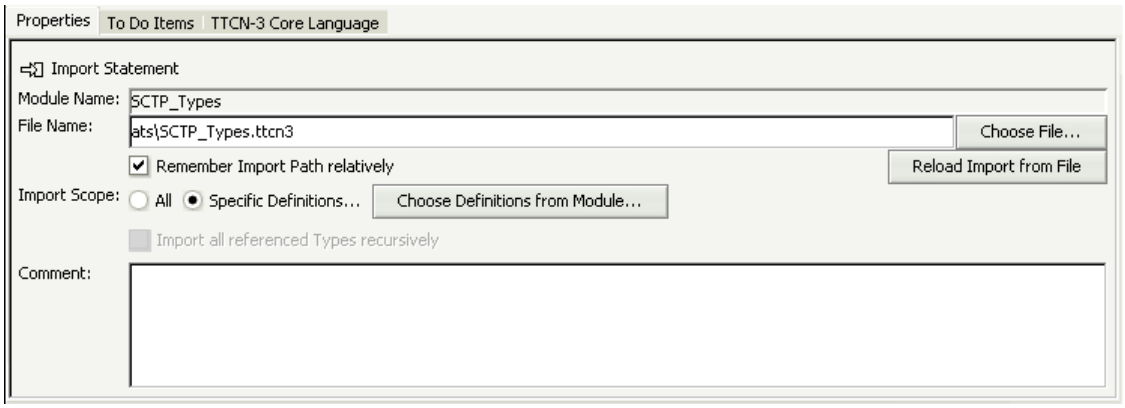
     A different file can be chosen or the current file can be re-imported after an external change.

     The checkbox Remember Import Path relatively controls how the file path of the imported module is saved in the GFT file. By default it is saved relatively to the GFT file. So references to module files will not break as long as a GFT file is moved together with its imported modules. When imported modules are stored in a different place than the GFT file it may be desirable to use an absolute path.

   - Import Scope:

     With TTCN-3 one can define precisely which definitions should be imported from a module. The default is to import all importable definitions. See the section called “Restricting the Scope of Imports” about how to import only a subset of a module's content.

Figure 6.12. Property Panel of an Import Statement

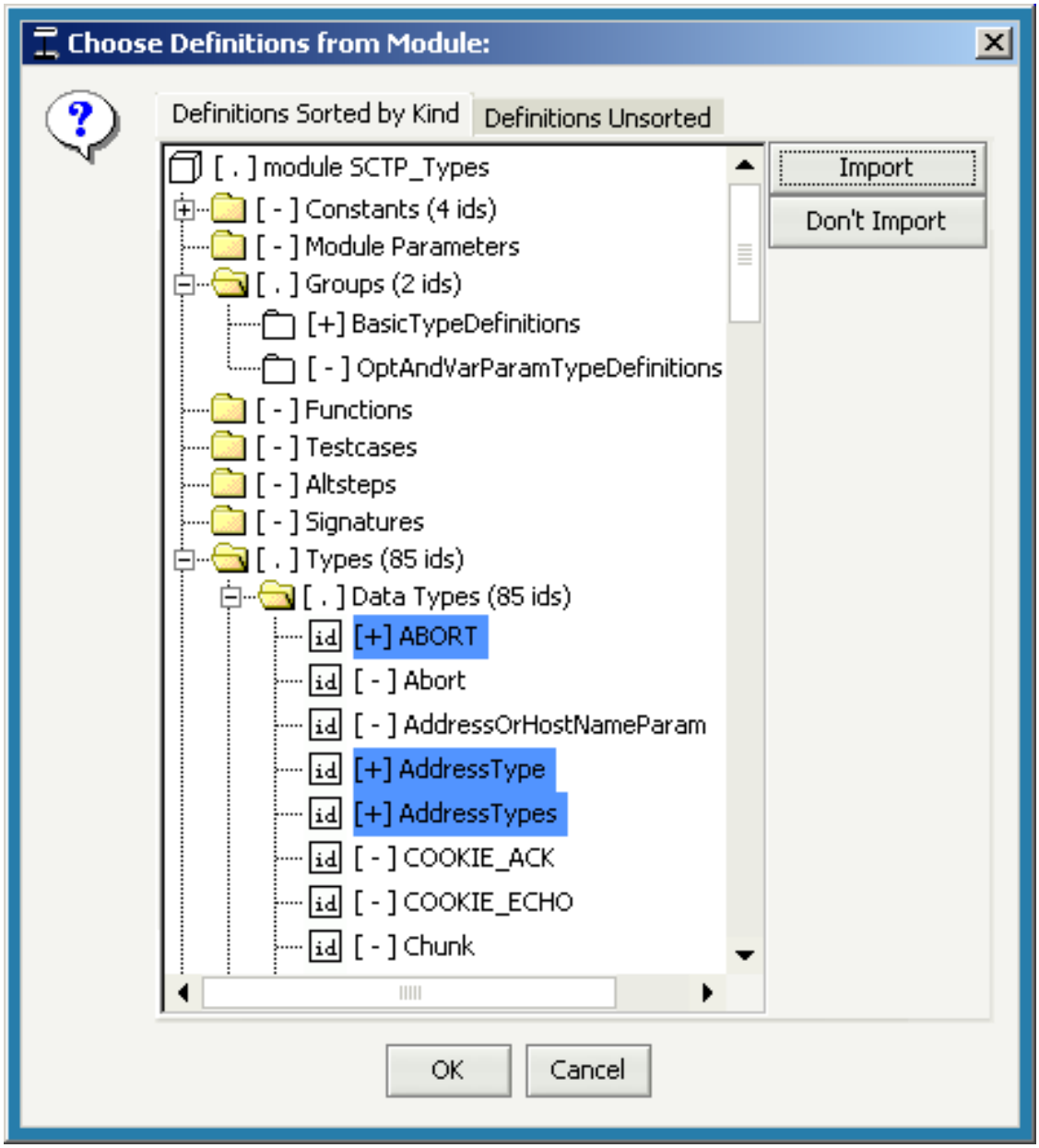


After an external change the imported TTCN-3 core language module can be re-imported (updated) via the Reload Import from File button on the import property panel. All imports can be updated at once via Reload all Imports menu, or just by hitting **Ctrl**+**R**.

Restricting the Scope of Imports

By default all importable definitions in a module will be imported and thus visible in GFT diagrams. To restrict the scope of an import, choose Specific Definitions... in the property panel. The dialog Choose Definitions from Module: will pop up, where definitions can be chosen to be imported.

Figure 6.13. Choosing the Definitions to be Imported



This dialog is similar to the overview window of imports. It shares the same two panels: the first one shows the definitions sorted by kind. Kinds are shown as folders, containing definitions. The second panel shows them unsorted as they appear in the module file.

Importable kinds are **Groups**, **Constants**, **Moduleparameters**, **Functions**, **Altsteps**, **Testcases**, **Signatures**, **Types** and **Templates**. When a **Group** is imported, all definitions in that group will be imported. Definitions selected for import have a plus-sign [+] before their name. A minus-sign [-] means no-import and containers may have a dot-sign [.], indicating that part of their content will be imported. Definitions are shown in a no-import state by default.

To import specific definitions, select them (use **Ctrl**+Left-Click or **Shift**+Left-Click to select more than one item) and click Import. To import all definitions of a kind, use the view Definitions Sorted by Kind. Then select the folder of the desired kind and click Import.

Import scope settings can be changed at any time by clicking the button Choose Definitions from Module... or by reverting the import scope back to All .

Importing TTCN-3 Core Language

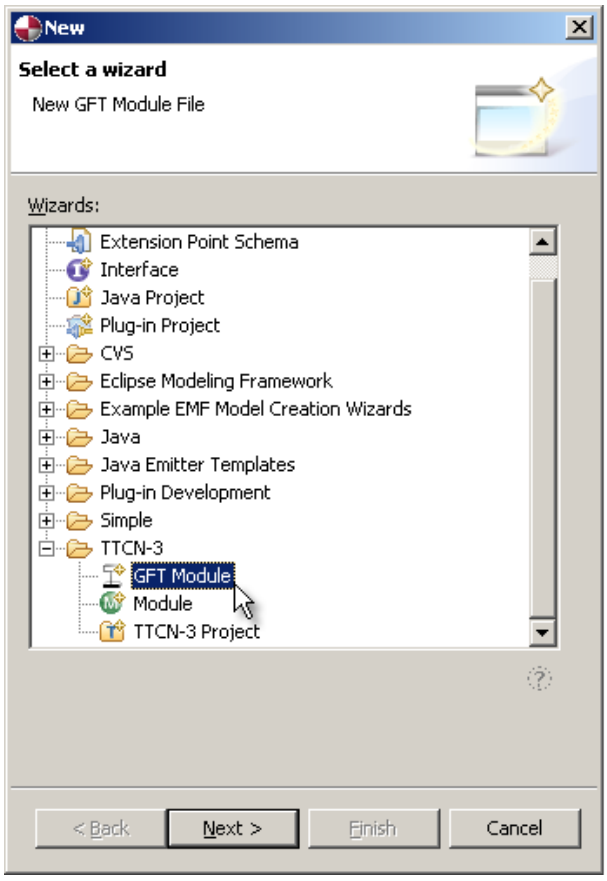
This function imports existing TTCN-3 sources into the graphical editor.

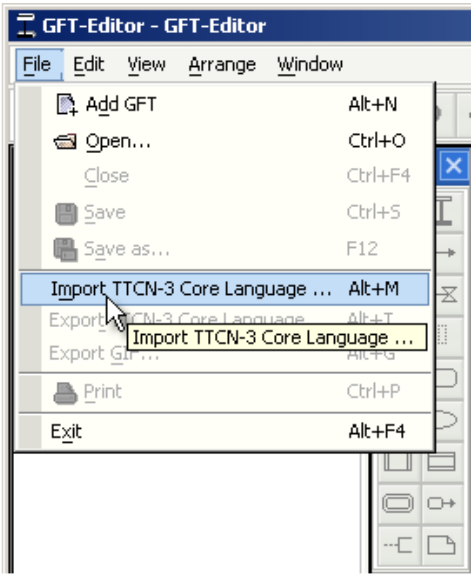
Note

GFT Editor does support importing TTCN-3 2.2.1 Core Language only, thus before importing, please make sure the Core Language to be imported is TTCN-3 2.2.1 compliant.

Therefore create a new GFT module using the wizard File/New/Other ... and choose GFT Module from the TTCN-3 group. Press Next, enter a container and a file name and press Finish. Cancel the New GFT dialog and select Import TTCN-3 Core Language... from the File menu or just hit **Alt**+**M**.

Figure 6.14. Importing TTCN-3 Core Language in GFT Editor





Note

Just to get name editing support and store a file reference to the module, see the section called “Importing Definitions”.

The generated GFT symbols will be automatic placed. This can take a long time and needs a lot of memory, particularly for large test suites.

When prompted, select the diagrams to import. It's also possible to import whole groups or even the whole module; multiple selections are possible. If diagrams already exists, the user has to decide, either to close the current module and to import the diagrams as a new GFT module or to insert the diagrams into the current module.

An information window and a status bar informs about the currently generated diagram, group, import statement and GFT symbol. A counter indicates the current and the last diagram number.

Note

If the free memory is low, an information message appears and the import stops.

Exporting TTCN-3 Core Language

The whole generated TTCN-3 core language can be seen by selecting the module control in the navigator tree and switching to the TTCN-3 Core Language tab on the bottom. To export it in a TTCN-3 core language file, select Export TTCN-3 Core Language... from the File menu or just hit **Alt**+**T**.

Exporting GIF

The diagrams can be exported as GIF files, e.g. for documentation purposes. Activate the window with the diagram and select Export GIF... from the File menu or just hit **Alt**+**G**.

Opening and Saving a GFT File

A project can be saved and loaded via a GFT file. If an older version of a GFT file is loaded, it will be converted to the current version.

Note

Upgrade of the GFT file takes effect by saving the project.

If a higher version has been detected while loading a GFT file, GFT Editor needs an update to read the file.

GFT Diagrams

Overview

GFT Editor provides the following diagram types:

- control diagram for the graphical presentation of a TTCN-3 module control part
- test case diagram for the graphical presentation of a TTCN-3 test case
- altstep diagram for the graphical presentation of a TTCN-3 altstep
- function diagram for the graphical presentation of a TTCN-3 function

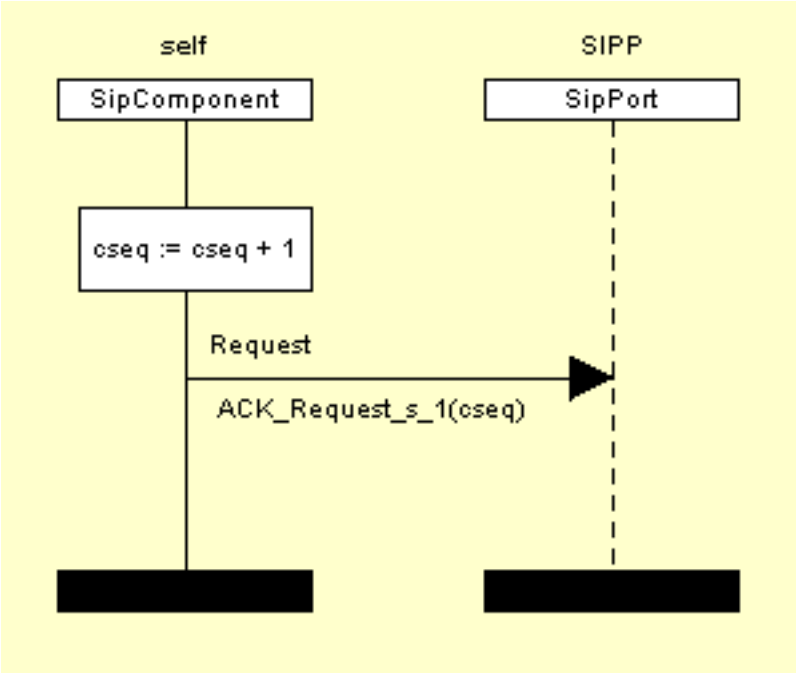
Common Properties

Each GFT control, test case, altstep and function diagram has a frame symbol (also called diagram frame) to define the diagram area. All symbols and text needed to define a complete and syntactically correct GFT diagram shall be made inside the diagram area.

Each GFT diagram has a diagram heading uniquely identify each GFT diagram type. The diagram heading has to be placed in the upper left-hand corner of the diagram frame.

Sequential behavior is represented by the order of events placed upon a test component instance. The ordering of events is taken in a top-down manner, with events placed nearest the top of the component instance symbol being evaluated first.

Figure 6.15. Sequential Behavior

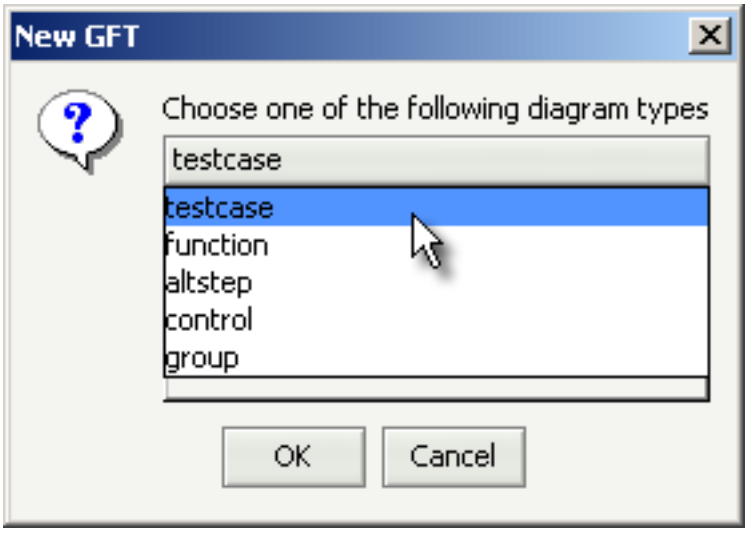


Creating a Diagram

To create a diagram choose Add GFT button  in the button bar, Add GFT menu, or just hit **Alt**+**N**.

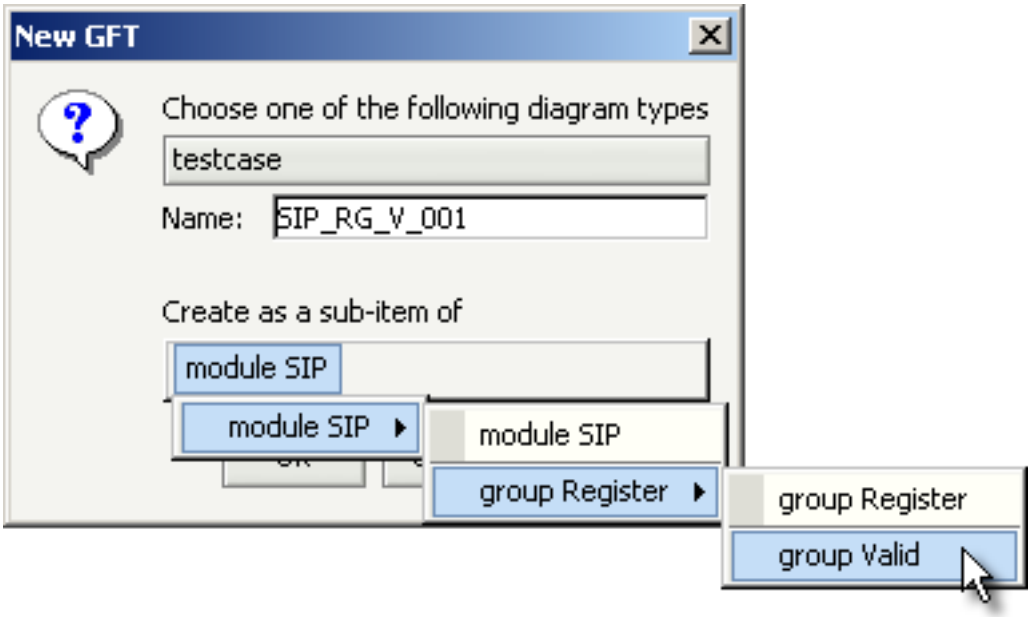
1. At the new diagram dialog box select a diagram type

Figure 6.16. Diagram Type Selection



2. Set a name
3. Select the target group

Figure 6.17. Target Group Selection



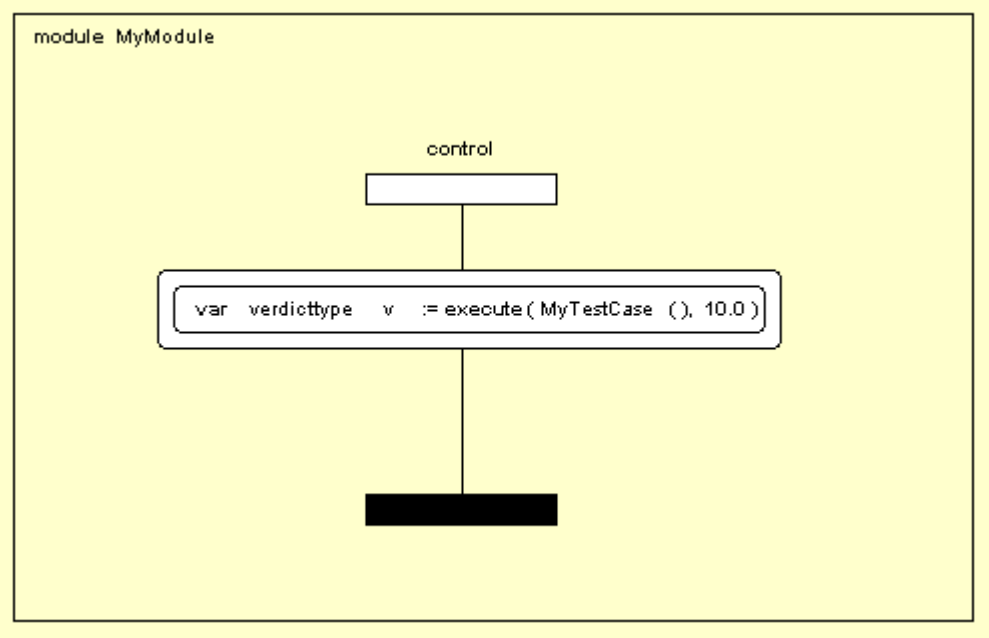
Note

If the diagram to be created is the very first diagram, a module containing the diagram will created automatically.

Control Diagram

A GFT control diagram provides a graphical presentation of the control part of a TTCN-3 module. The heading of a control diagram is the keyword **module** followed by the module name. A GFT control diagram shall only include one component instance (also called control instance) with the instance name control without any type information. The control instance describes the behavior of the TTCN-3 module control part. Attributes associated to the TTCN-3 module control part shall be specified within a text symbol in the control diagram. On creation the control diagram has the default name 'untitled'. Only by saving the GFT file it is possible to change this name. Anytime there is consistency between the module name and the control diagram name.

Figure 6.18. Control Diagram



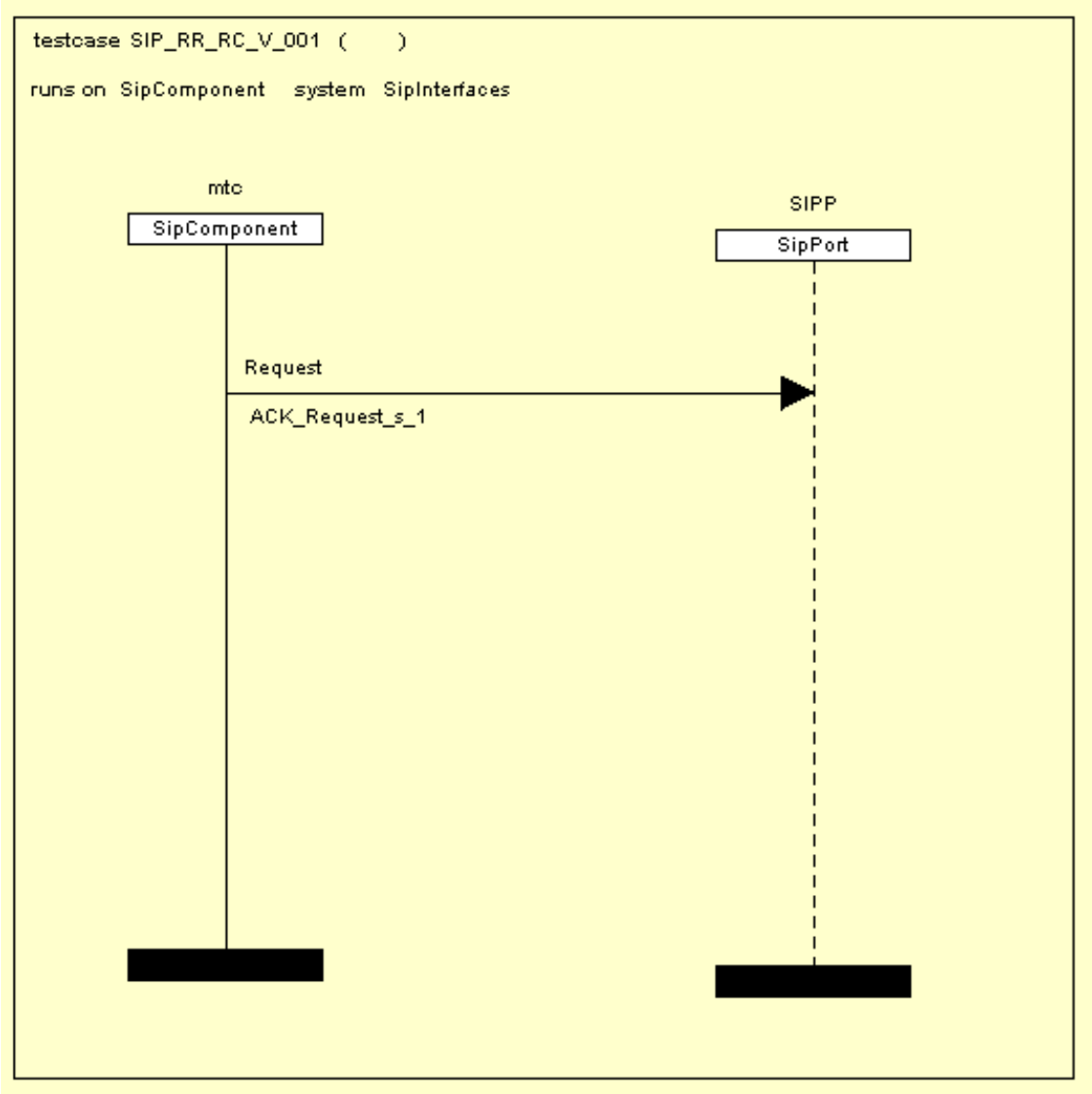
Within the control part, test cases can be selected or deselected for the test case execution with the use of Boolean expressions. The GFT symbols are attached to the control instance.

Test Case Diagram

A GFT test case diagram provides a graphical presentation of a TTCN-3 test case. The heading of a test case diagram shall be the keyword **testcase** followed by the complete signature of the test case.

A GFT test case diagram shall include one test component instance describing the behavior of the **mtc** (also called **mtc** instance) and one port instance for each port owned by the **mtc**. The type associated with the **mtc** instance is optional, but if the type information is present, it shall be identical to the component type referred to in the **runs on** clause of the test case signature. The names associated with the port instances shall be identical to the port names defined in the component type definition of the **mtc** . The associated type information for port instances is optional. If the type information is present, port names and port types shall be consistent with the component type definition of the **mtc** . The **mtc** and port types are displayed in the component or port instance head symbol.

Figure 6.19. Test Case Diagram



A test case represents the dynamic test behavior and can create test components. A test case may contain declarations, statements, communication and timer operations and invocation of functions or altsteps.

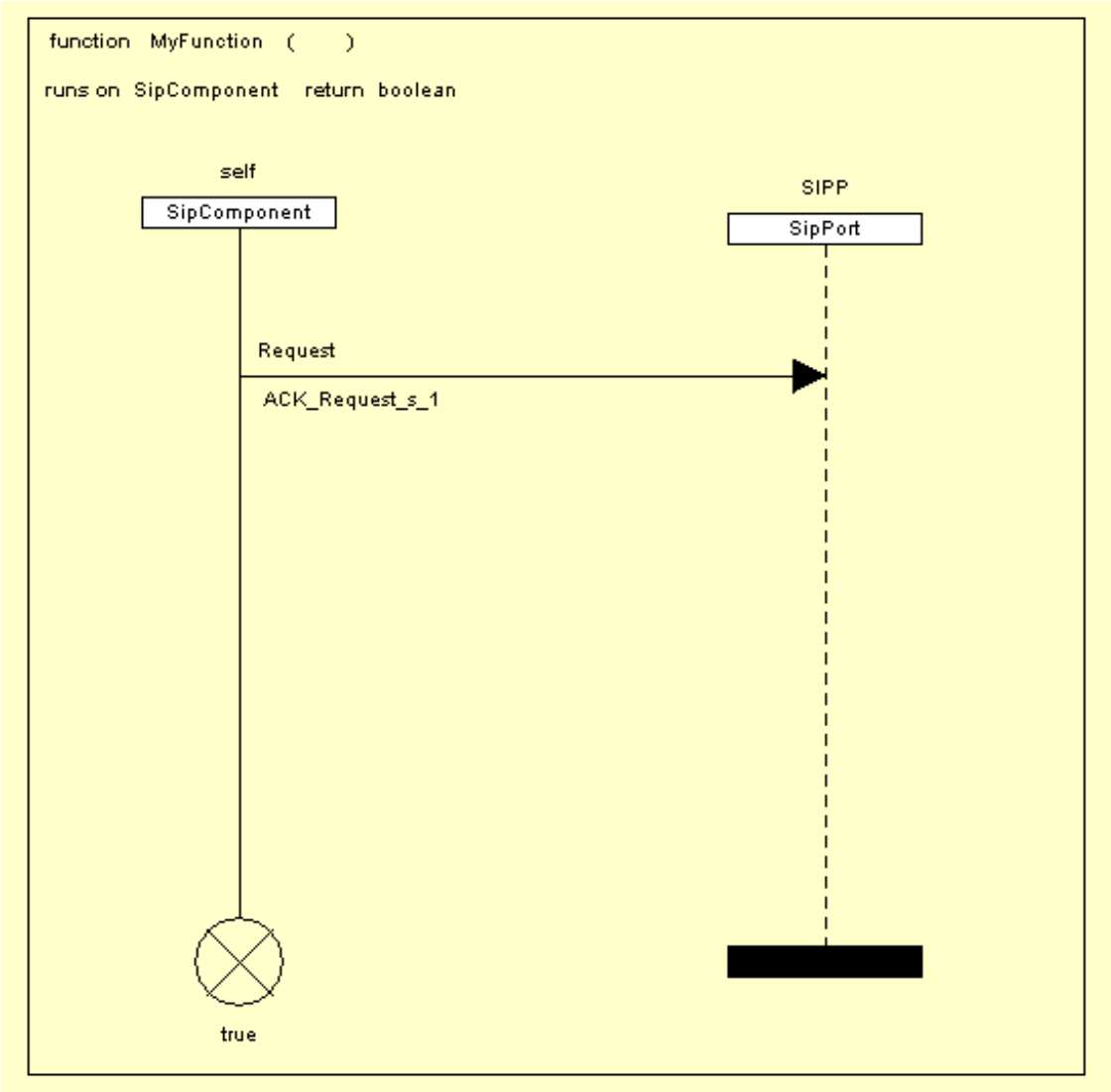
Function Diagram

GFT presents TTCN-3 functions by means of function diagrams. The heading of a function diagram shall be the keyword **function** followed by the complete signature of the function.

A GFT function diagram shall include one test component instance describing the behavior of the function and one port instance for each port usable by the function.

The name associated with the test component instance shall be **self**. The type associated with the test component instance is optional, but if the type information is present, it shall be consistent with the component type in the **runs on** clause.

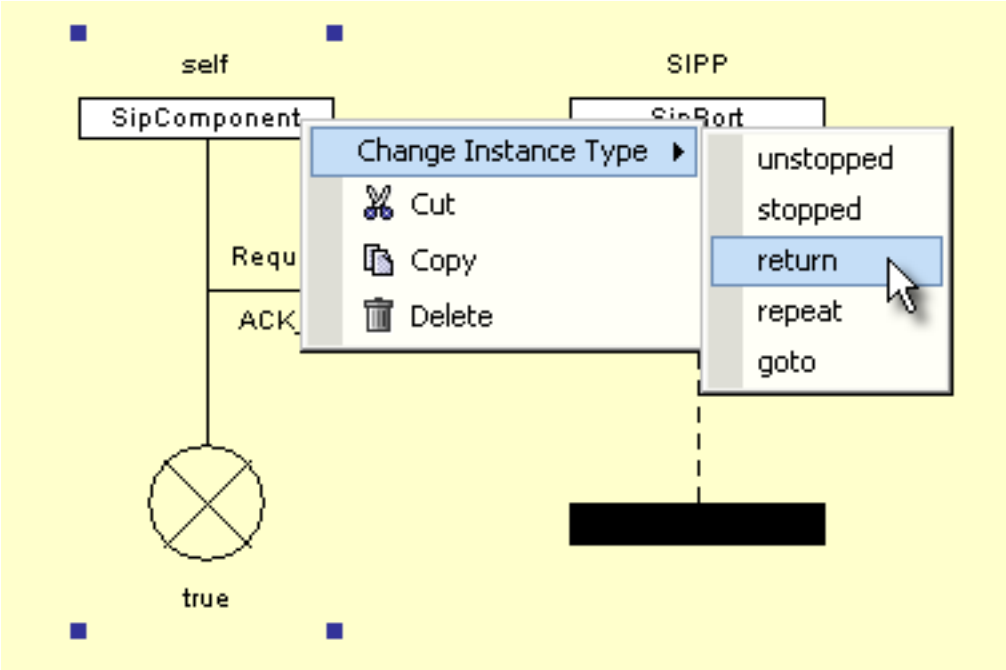
Figure 6.20. Function Diagram



A function is used to specify and structure test behavior, define default behavior or to structure computation in a module. A function may contain declarations, statements, communication and timer operations and invocation of function or altsteps and an optional **return** statement.

The **return** statement shall be represented by a return symbol on the respective instance. This may be optionally associated with a return value. To do so right-click on the component instance and choose return from the Change Instance Type popup menu.

Figure 6.21. Add a Return Symbol to a Function Component Instance



Note

A return symbol shall only be used in a GFT function diagram. It shall only be used as last event of a component instance or as last event of an operand in an inline expression symbol.

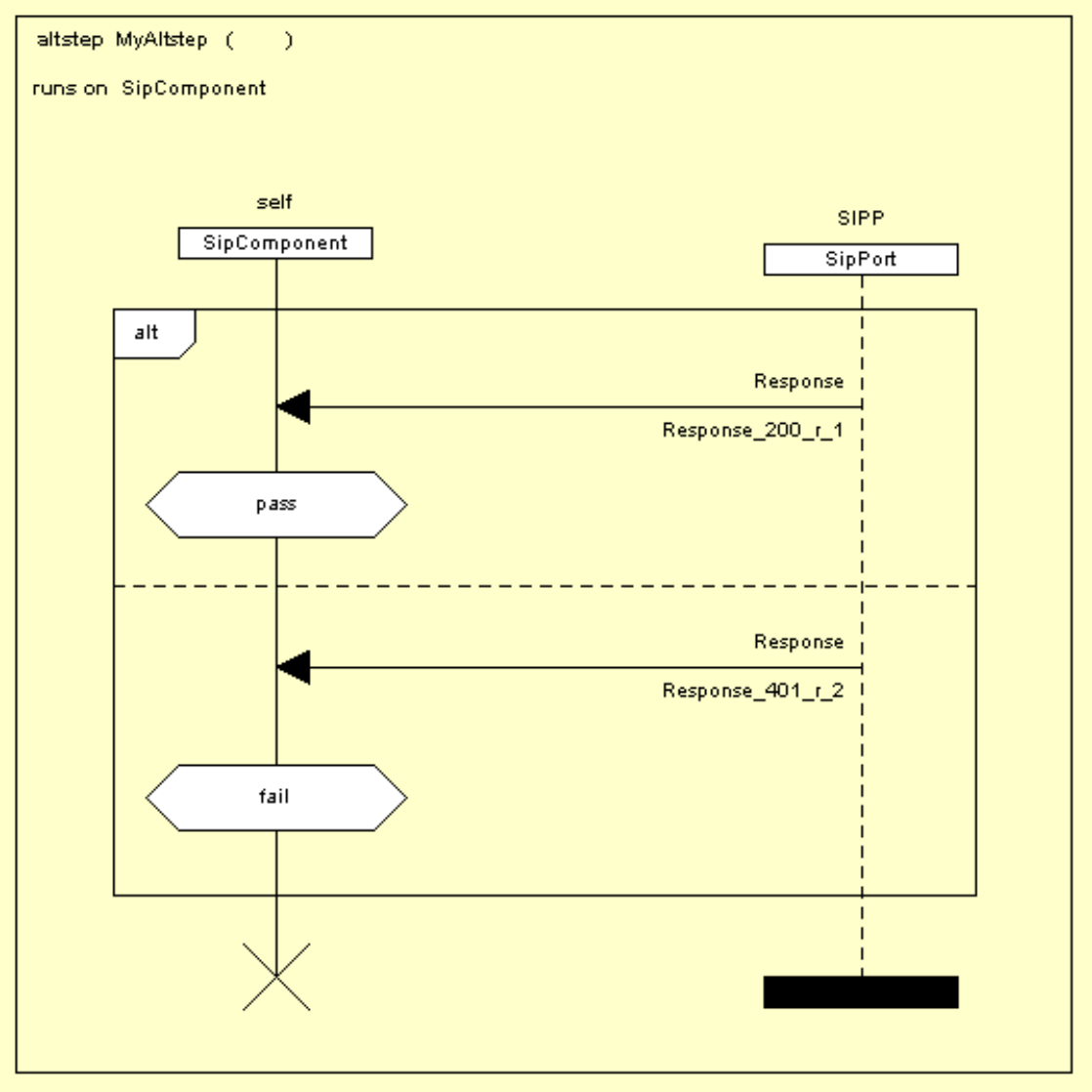
Altstep Diagram

GFT presents TTCN-3 altsteps by means of altstep diagrams. The heading of an altstep diagram shall be the keyword **altstep** followed by the complete signature of the altstep.

A GFT altstep diagram shall include one test component instance describing the behavior of the altstep and one port instance for each port usable by the altstep.

The name associated with the test component instance shall be **self**. The type associated with the test component instance is optional, but if the type information is present, it shall be consistent with the component type in the **runs on** clause.

Figure 6.22. Altstep Diagram



An altstep is used to specify default behavior or to structure the alternatives of an **alt** statement. An altstep may contain statements, communication and timer operations and invocation of function or altsteps.

GFT Symbols

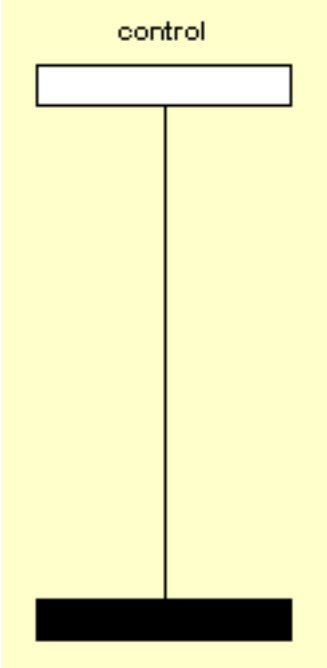
Instances

GFT diagrams include the following kinds of instances:

- control instances describing the flow of control for the module control part
- test component instances describing the flow of control for the test component that executes a test case, function or altstep
- port instances representing the ports used by the different test components

Only one control instance shall exist within a GFT control diagram. A control instance describes the flow of control of a module control part. A GFT control instance shall graphically be described by a component instance symbol with the mandatory name **control** placed on top of the instance head symbol. No instance type information is associated with a control instance.

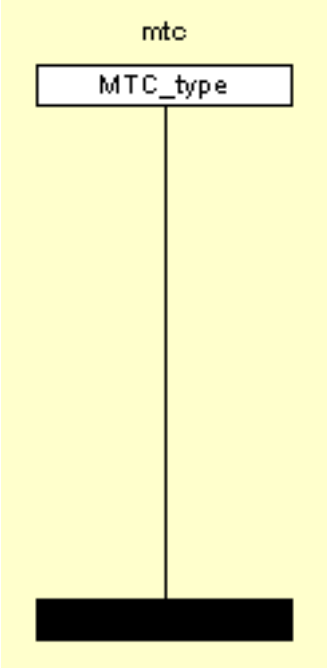
Figure 6.23. Control Instance Symbol



Each GFT test case, function or altstep diagram includes one test component instance that describes the flow of control of that instance. A GFT test component instance shall graphically be described by an instance symbol with:

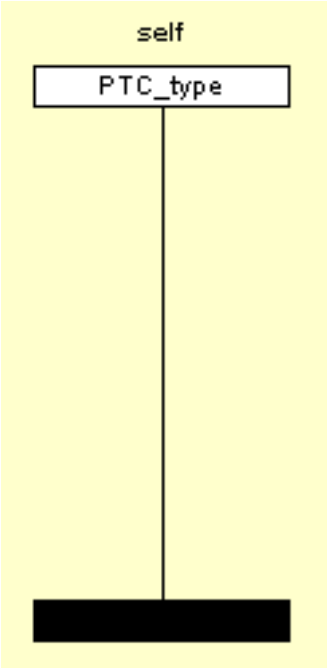
- the mandatory name **mtc** placed on top of the instance head symbol in the case of a test case diagram

Figure 6.24. Test Component Instance in Test Cases



- the mandatory name self placed on top of the instance head symbol in the case of a function or altstep diagram

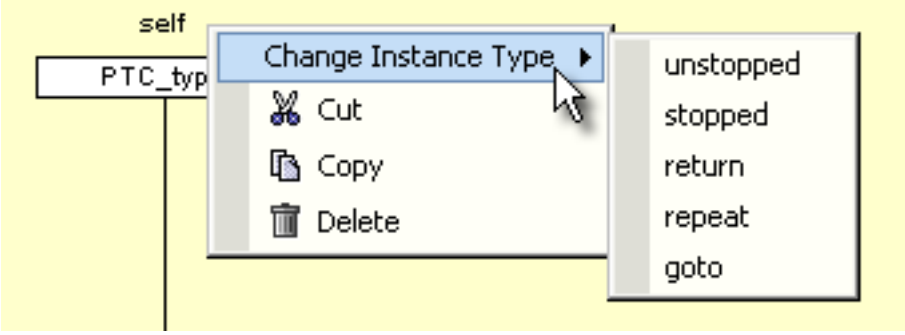
Figure 6.25. Test Component Instance Symbol in Functions and Altsteps



The optional test component type may be provided within the instance head symbol. It has to be consistent with the test component type given after the runs on keyword in the heading of the GFT diagram.

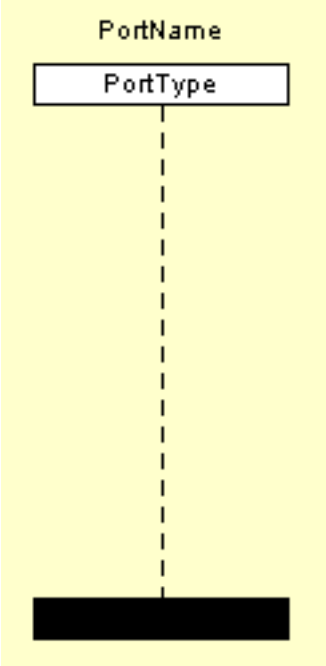
The type of a test component instance may be changed in order to stop behavior execution or to repeat inside an alternative or to return a function.

Figure 6.26. Changing the Type of Test Components



GFT port instances may be used within test case, altstep and function diagrams. A port instance represents a port that is usable by the test component that executes the specified test case, altstep or function. A GFT port instance is graphically described by a component instance symbol with a dashed instance line. The name of the represented port is mandatory information and shall be placed on top of the instance head symbol. The port type (optional) may be provided within the instance head symbol.

Figure 6.27. Port Instance Symbol



Actions

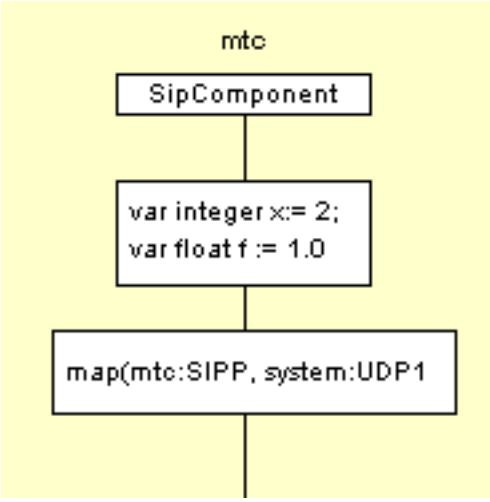
GFT actions may be used within all types of diagrams and may contain sequences of statements in TTCN-3 core language. The following TTCN-3 declarations, statements and operations can only be specified within action symbols:

- **log**

- **connect**
- **disconnect**
- **map**
- **unmap**
- **action**
- external functions
- predefined functions
- declarations, with the exception of those specified in create symbols, default symbols, reference symbols, execute symbols
- assignments, with the exception of those specified in create symbols, default symbols, reference symbols, execute symbols

The semicolon is optional if a GFT symbol includes only one statement in TTCN-3 core language. Semicolons shall separate the statements in a sequence of statements within an action symbol. The semicolon is optional for the last statement in the sequence.

Figure 6.28. Action Symbol



Note

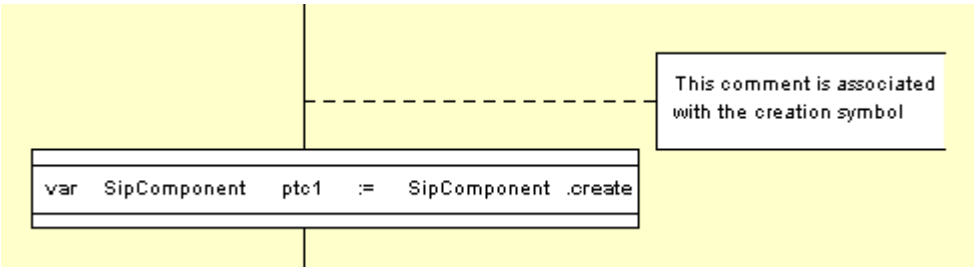
GFT symbols and action symbols containing TTCN-3 core language may arbitrarily be mixed to focus on dedicated parts of a TTCN-3 specification.

Comments

GFT provide three possibilities to put comments into GFT diagrams:

- The event comment symbol can be used to associate comments to GFT symbols. It has to be attached directly before or directly after the respective symbol on the component axis. A comment in an event comment symbol can be provided in form of free text, i.e. the comment delimiter "/*", "*/" and "//" of the core language need not to be used.

Figure 6.29. Event Comment Symbol

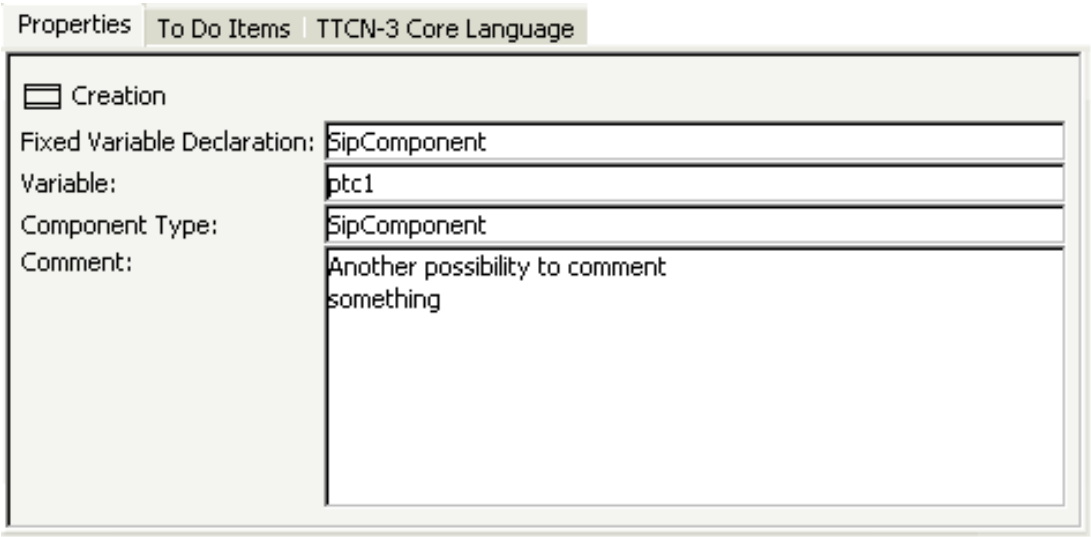


- Comments in the syntax for comments of the TTCN-3 core language can be put into text symbols (see the section called “Text Symbols”) and freely placed in the GFT diagram area.

Figure 6.30. Text Symbol Comment

- Comments may be added to GFT symbols in the respective property panel. A comment can be provided in form of free text, i.e. the comment delimiter "/*", "*/" and "//" of the core language need not to be used.

Figure 6.31. Comment in Property Panel

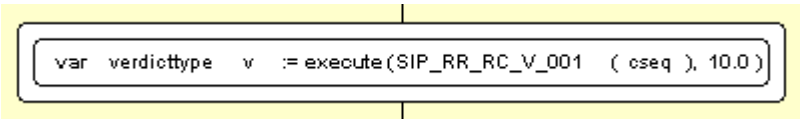


Execute Test Cases

The execution of test cases is represented by use of the execute test case symbol. The syntax of the execute statement is placed within that symbol. The symbol may contain:

- an execute statement for a test case with optional parameters
- optionally, time supervision
- optionally, the assignment of the returned verdict to a **verdicttype** variable
- optionally, the inline declaration of the **verdicttype** variable

Figure 6.32. Execute Symbol



References

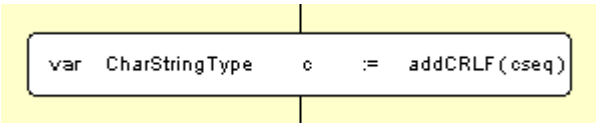
The invocation of functions and altsteps is represented by the reference symbol.

The syntax of the function invocation is placed within the reference symbol. The symbol may contain:

- the invocation of a function with optional parameters
- an optional assignment of the returned value to a variable
- an optional inline declaration of the variable

In case of an altstep invocation the syntax of the altstep invocation is placed within that symbol. The symbol may contain the invocation of an altstep with optional parameters. It shall be used within alternative behavior only, where the altstep invocation shall be one of the operands of the alternative statements.

Figure 6.33. Function Reference



Note

The reference symbol is only used for user defined functions and altsteps defined within the current module. It shall not be used for invocation of external functions or predefined TTCN-3 functions, which shall be represented in the TTCN-3 core language within an action symbol (see the section called “Actions”). Furthermore it should not be used where a function is called inside a TTCN-3 language element that has its own GFT symbol.

Note

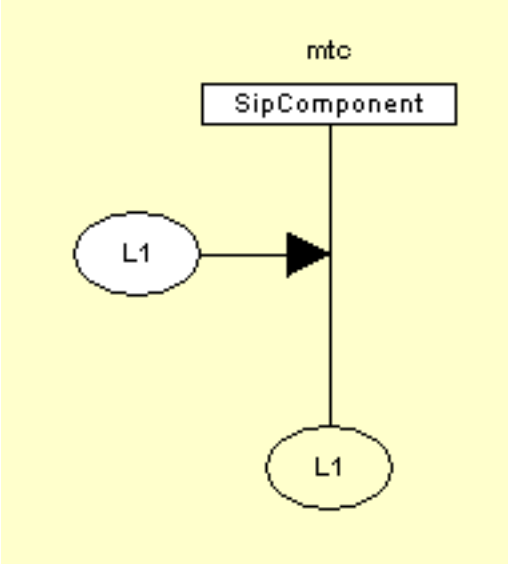
Another possibility of altstep invocation is the implicit invocation of altsteps via activated defaults (see the section called “Defaults”).

Labels and Goto

The **label** statement shall be represented with a label symbol, which is connected to a component instance.

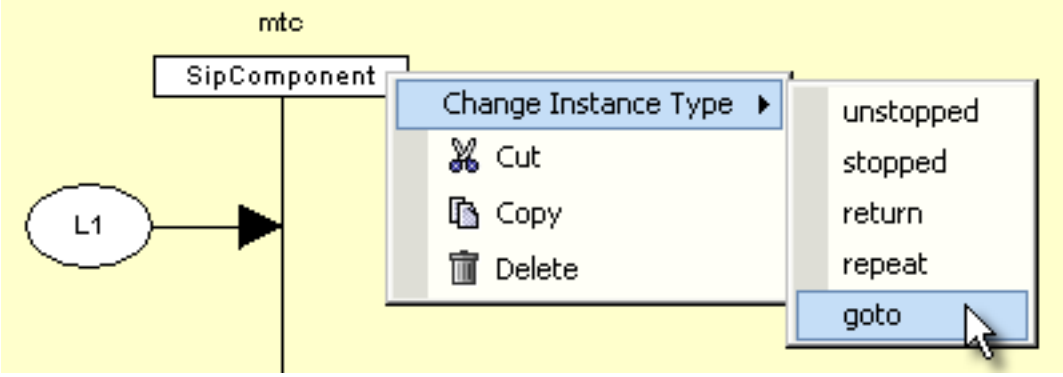
The **goto** statement shall be represented with a goto symbol. It shall be placed at the end of a component instance or at the end of an operand in an inline expression symbol.

Figure 6.34. Label and Goto



Setting the goto symbol right-click on the respective component instance and change the instance type to goto.

Figure 6.35. Setting the Goto Type



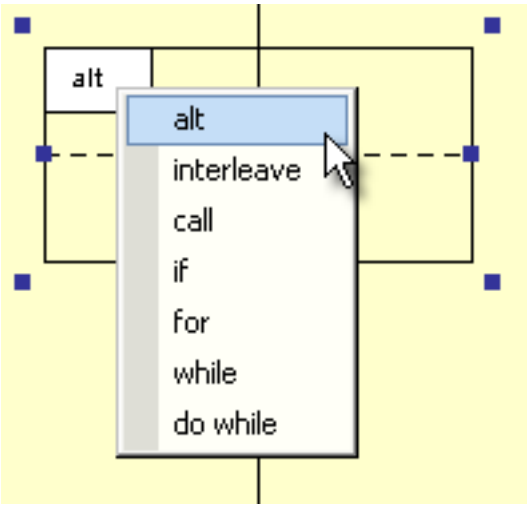
Inline Expressions

GFT inline expression symbols may be used for TTCN-3

- **if-else**
- **for**
- **while**
- **do-while**
- **alt**
- **interleave**
- **call**

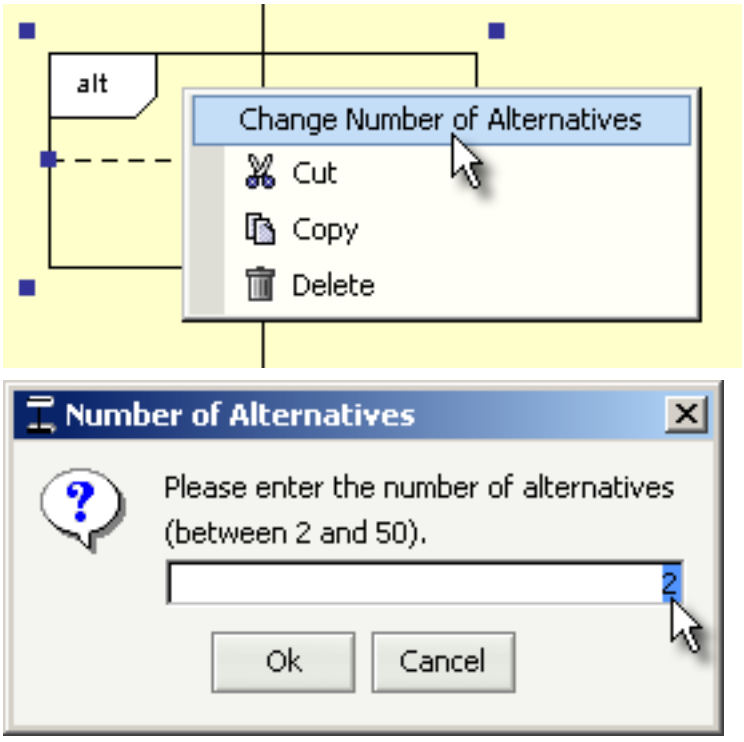
To change the type, left-click on the inline expression label and choose the desired one.

Figure 6.36. Type Change of an Inline Expression



For **alt**, **interleave**, **if-else**, and **call** it is also possible to change the number of operands, alternatives, etc. To do so right-click on the inline expression label and set the desired number.

Figure 6.37. Change the Number of Alternatives



If-Else Statements

The **if-else** statement shall be represented by an inline expression symbol labeled with the **if** keyword and a boolean expression. The if-else inline expression symbol may contain one or two operands, separated by a dashed line.

Figure 6.38. If

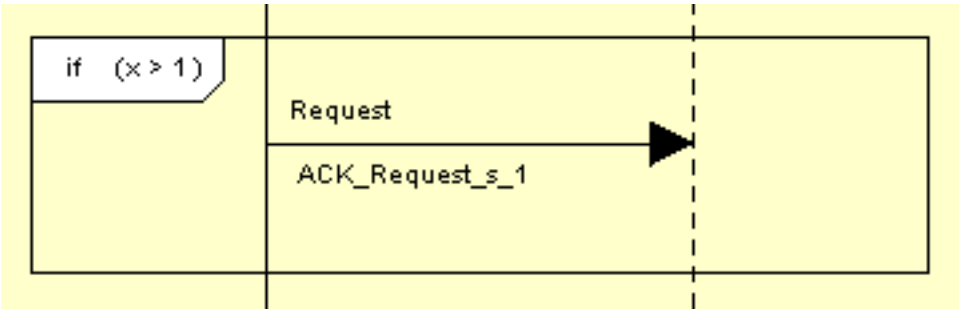
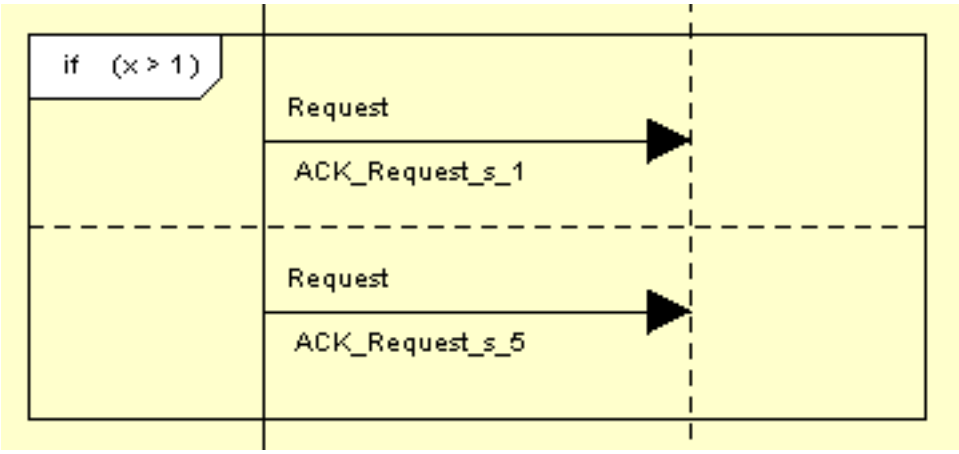


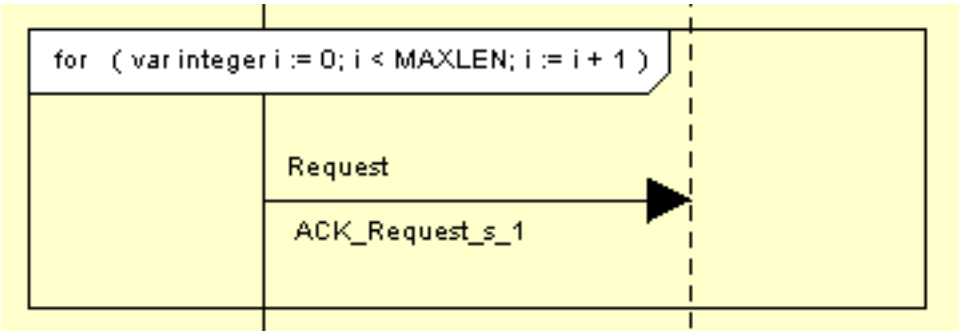
Figure 6.39. If-Else



For Statements

The **for** statement shall be represented by an inline expression symbol labeled with a **for** definition. The **for** body shall be represented as the operand of the for inline expression symbol.

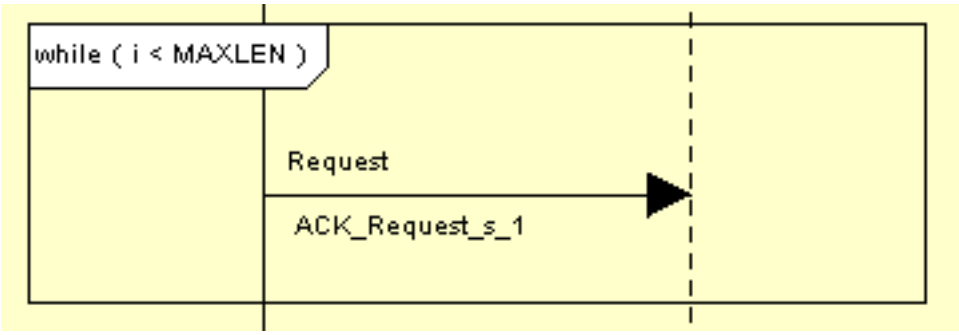
Figure 6.40. For



While Statements

The **while** statement shall be represented by an inline expression symbol labeled with a **while** definition. The **while** body shall be represented as the operand of the while inline expression symbol.

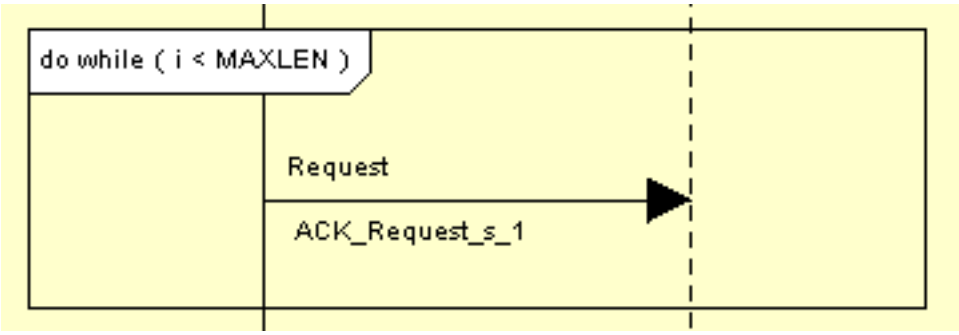
Figure 6.41. While



Do-While Statements

The **do-while** statement shall be represented by an inline expression symbol labeled with a **do-while** definition. The **do-while** body shall be represented as the operand of the do-while inline expression symbol.

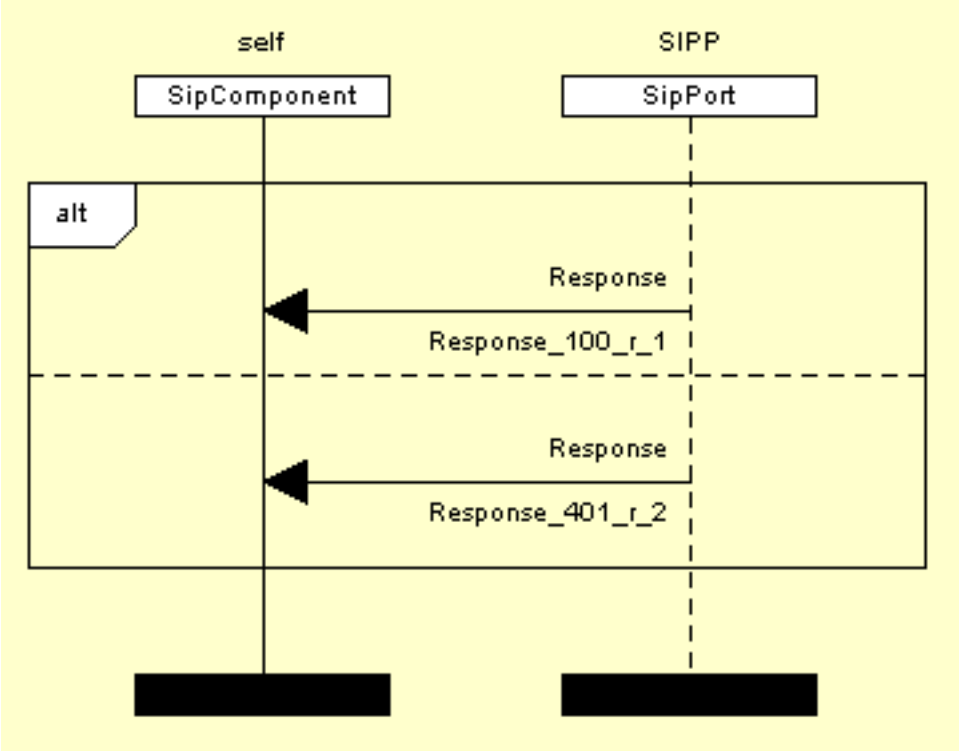
Figure 6.42. Do-While



Alt Statements

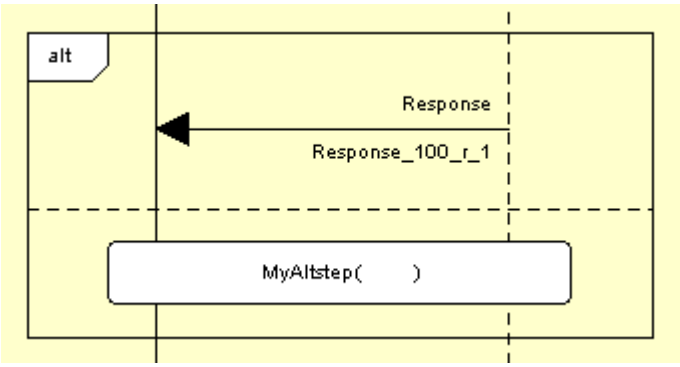
Alternative behavior shall be represented using inline expression symbol with the **alt** keyword placed in the top left hand corner. Each operand of the alternative behavior shall be separated using a dashed line. Operands are evaluated top-down.

Figure 6.43. Alt



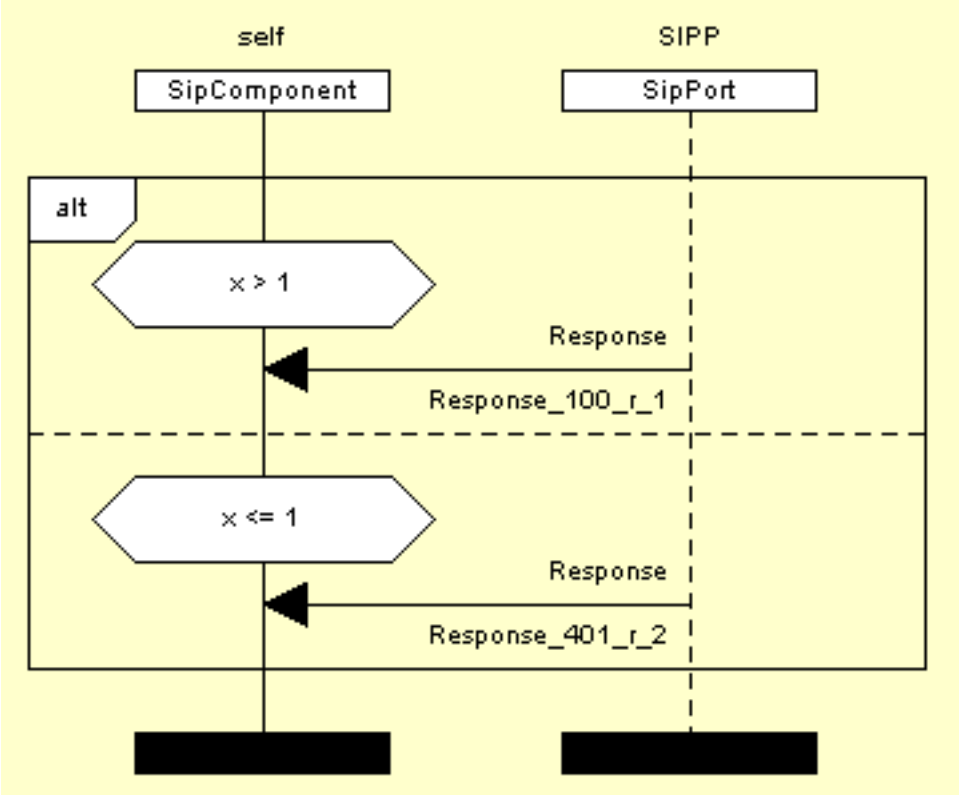
In addition, it is possible to call an altstep as the only event within an alternative operand.

Figure 6.44. Alt With Altstep Invocation



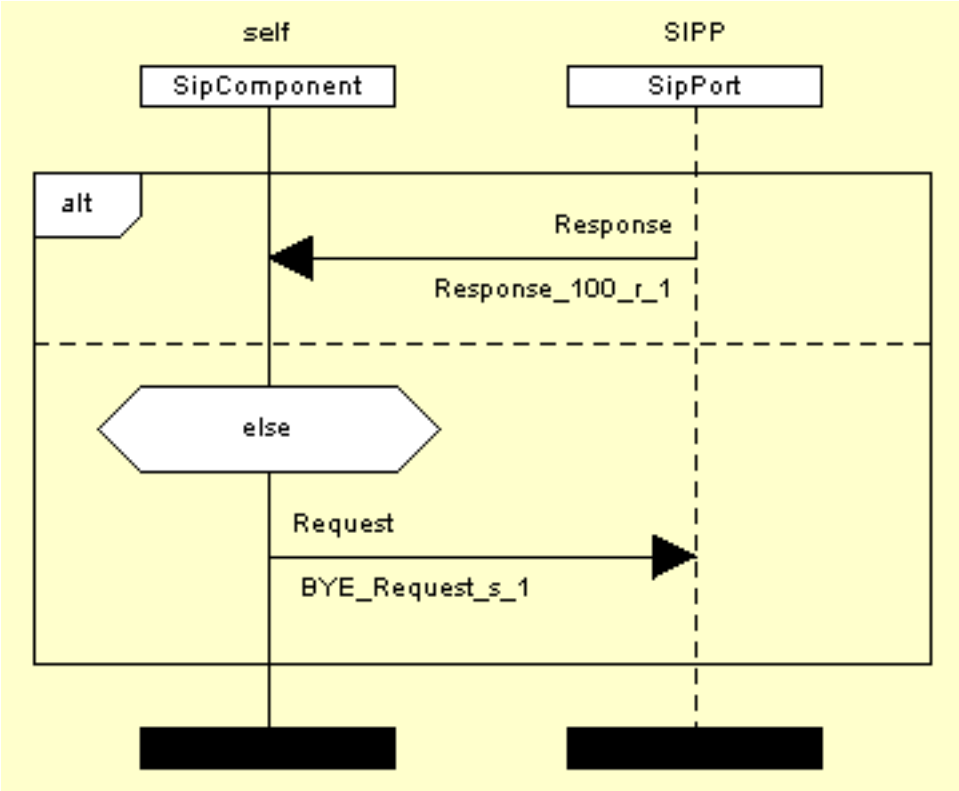
It is possible to disable/enable an alternative operand by means of a boolean expression contained within a condition symbol placed upon the test component instance.

Figure 6.45. Selecting/Deselecting an Alt



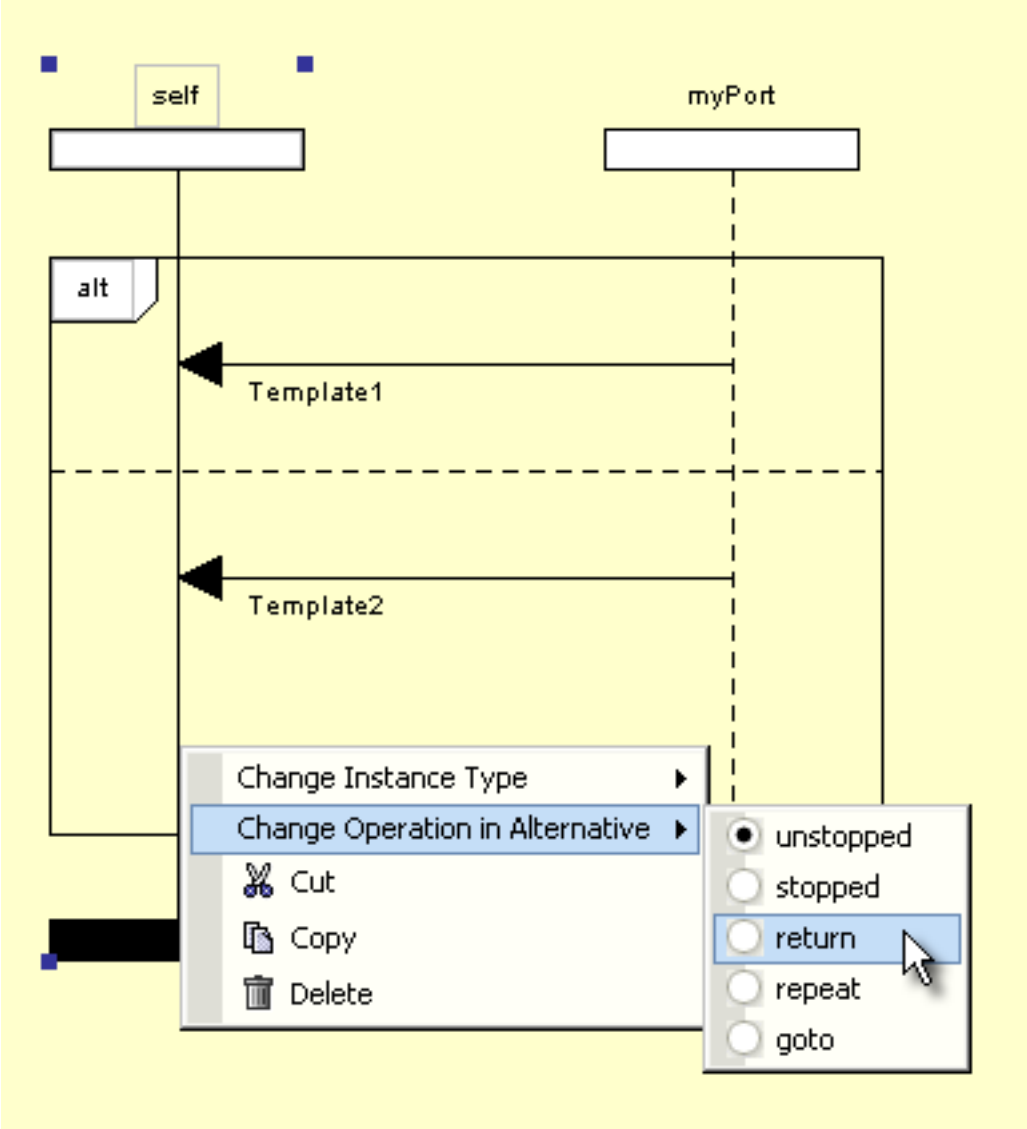
The **else** branch shall be denoted using a condition symbol placed upon the test component instance axis labeled with the **else** keyword.

Figure 6.46. Alt With Else Branch



To represent a **stop**, **return**, **repeat** or **goto** statement inside an inline expression, right-click on the component instance axis short above a dashed separator line or above the bottom of the inline expression, select Change Operation in Alternative from the popup menu and select stopped, return, repeat or goto; to switch back, select unstopped.

Figure 6.47. Change Operation in Alternative

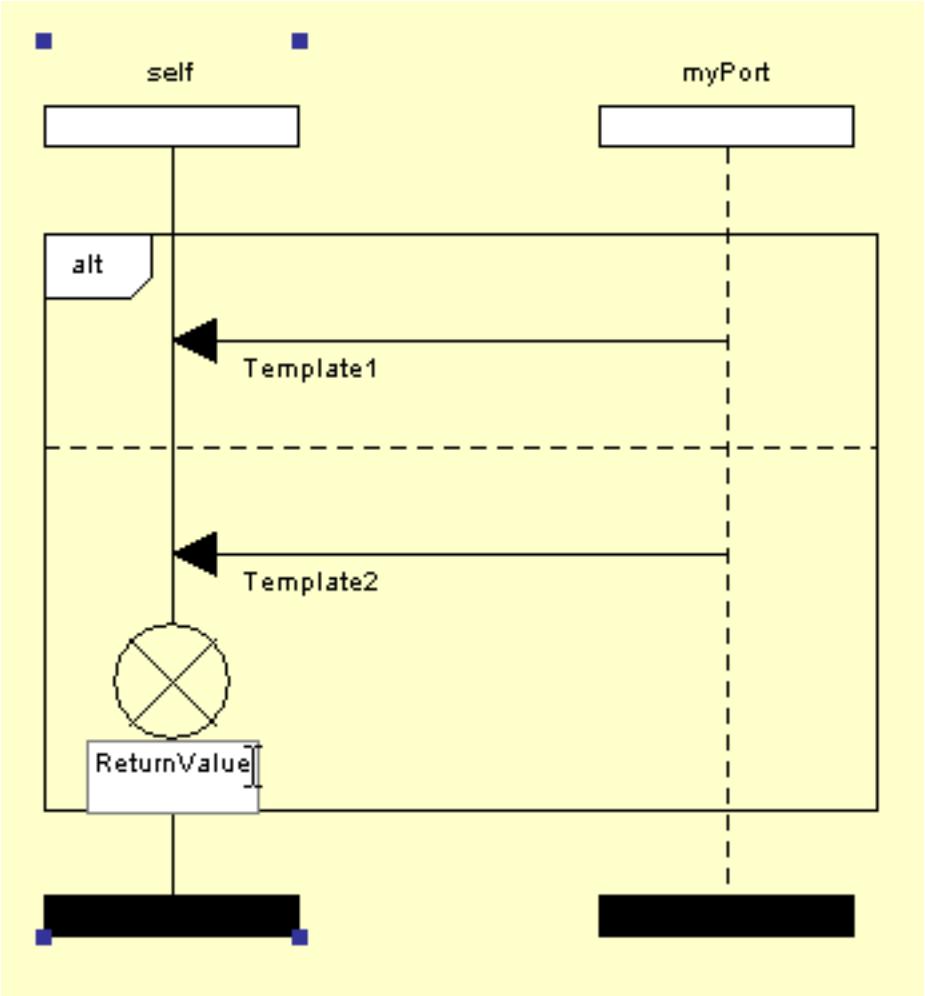


Note

A **return** statement is only possible in a GFT function diagram.

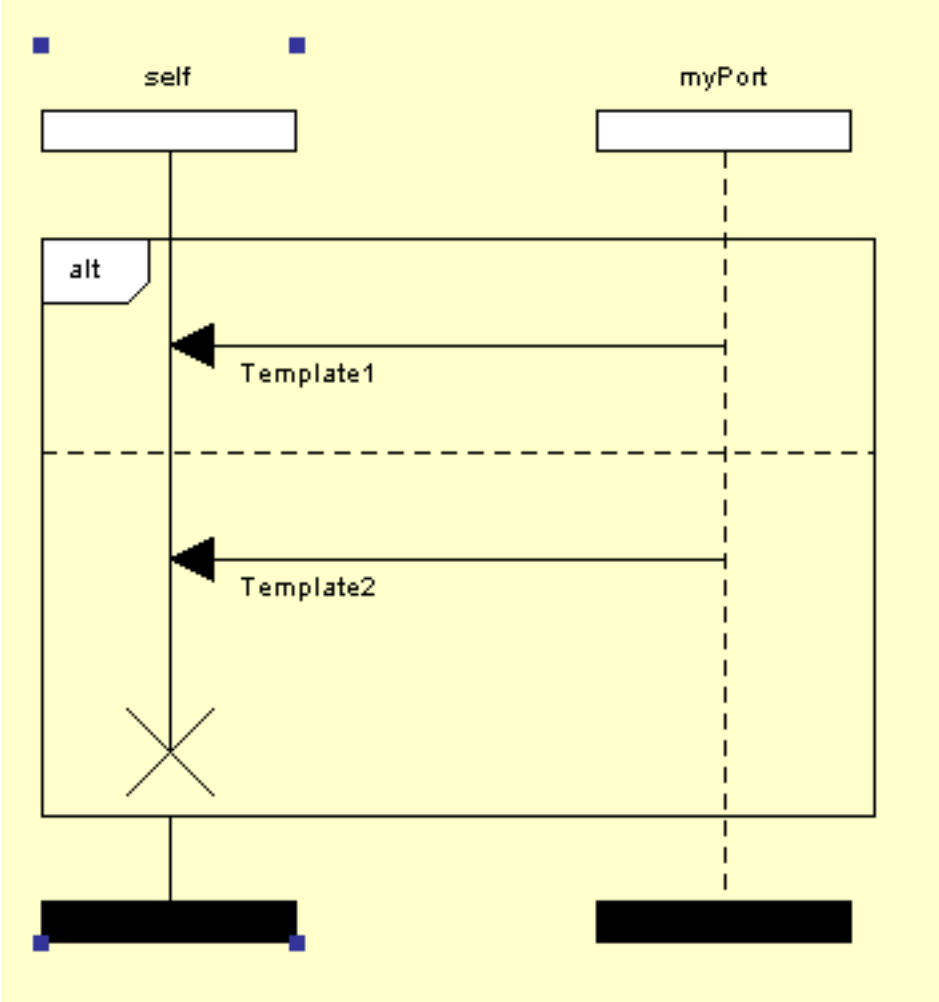
A **return** statement may be optionally associated with a return value. To specify a return value or a goto label, left-click on the text field of this symbol and enter the value.

Figure 6.48. Associate a Return Statement with a Return Value



A **stop** execution operation shall be represented by a stop symbol, which is attached to the test component instance, which performs the **stop** execution operation. It shall only be used as last event of a component instance or as last event of an operand in an inline expression.

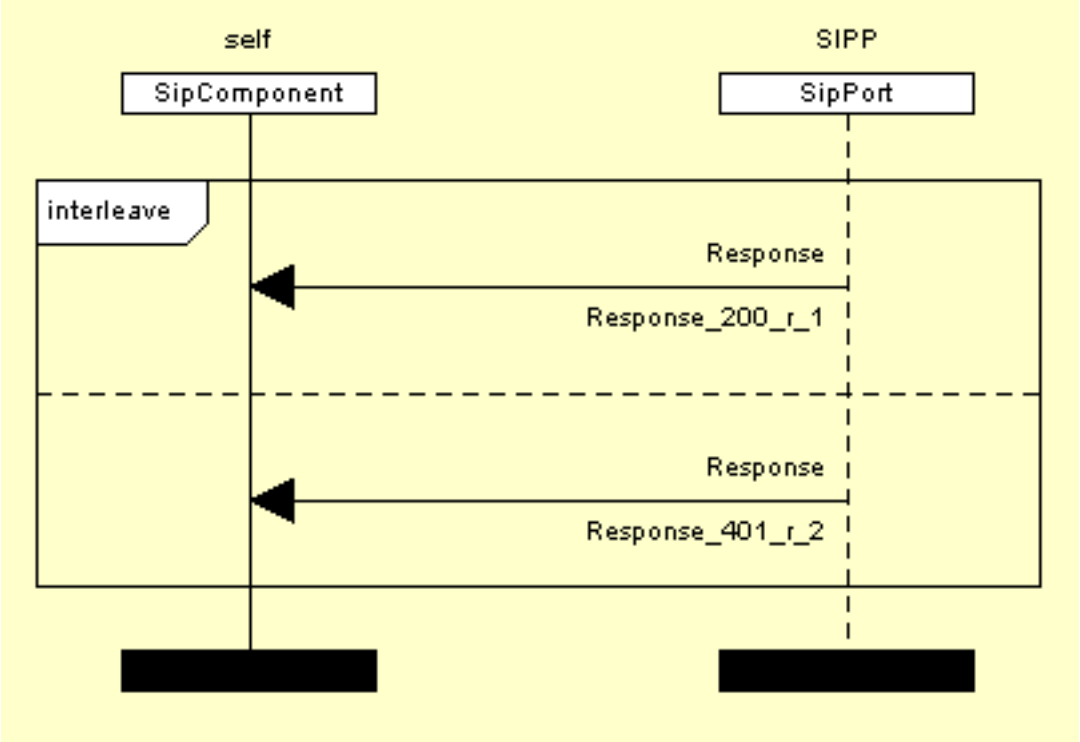
Figure 6.49. Stop Execution Operation



Interleave Statements

Interleave behavior shall be represented using an inline expression symbol with the **interleave** keyword placed in the top left hand corner. Each operand shall be separated using a dashed line. Operands are evaluated in a top-down order.

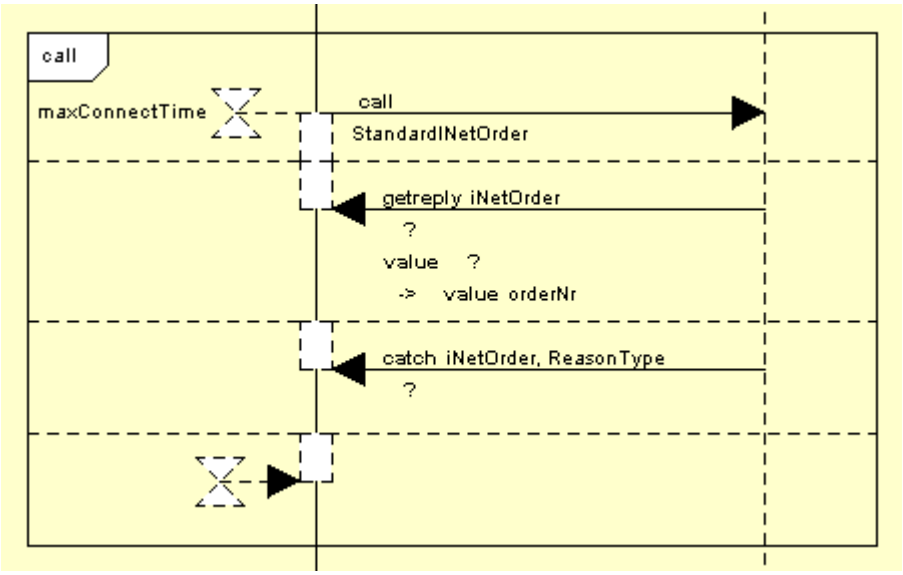
Figure 6.50. Interleave



Call Statements

Blocking call behavior shall be represented using an inline expression symbol with the **call** keyword placed in the top left hand corner. Each alternative shall be separated using a dashed line. On every dashed line a suspension region symbol should be attached to the component instance.

Figure 6.51. Call



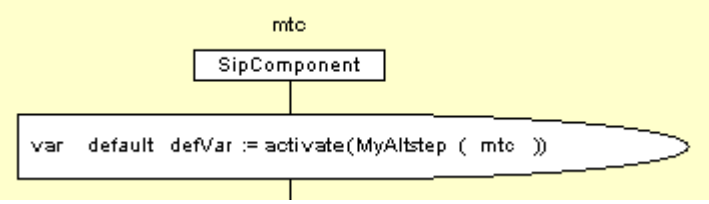
Defaults

GFT provides graphical representation for the activation and deactivation of defaults.

The activation of defaults shall be represented by the placement of the **activate** statement within a default symbol. As default behavior an altstep shall be referenced.

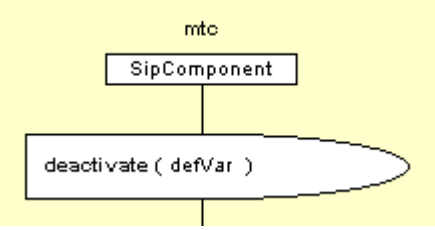
The return value of the default activation may be saved for subsequent deactivation. Variables of type **default** can either be declared directly within a default symbol as part of an **activate** statement or outside, i.e. in an imported module or in an action symbol.

Figure 6.52. Activate



The deactivation of defaults shall be represented by the placement of the **deactivate** statement within a default symbol. If no operands are given to the **deactivate** statement then all defaults are deactivated, otherwise only the referenced default will deactivated.

Figure 6.53. Deactivate

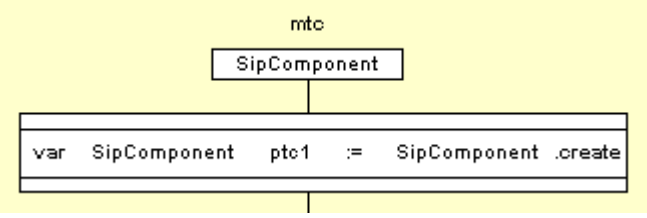


Creates

The create operation shall be represented within the create symbol, which is attached to the test component instance which performs the create operation. The create symbol contains the **create** statement.

The return value of the create operation may be saved for subsequent use, e.g. starting the component behavior. Variables of the respective component type can either be declared directly within a create symbol as part of a **create** statement or outside, i.e. in an imported module or in an action symbol.

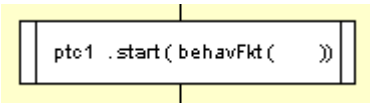
Figure 6.54. Create



Start Components

The **start** test component operation shall be represented within the start symbol, which is attached to the test component instance that performs the start operation. The start symbol contains the **start** statement. As behavior a TTCN-3 function has to be referenced.

Figure 6.55. Start



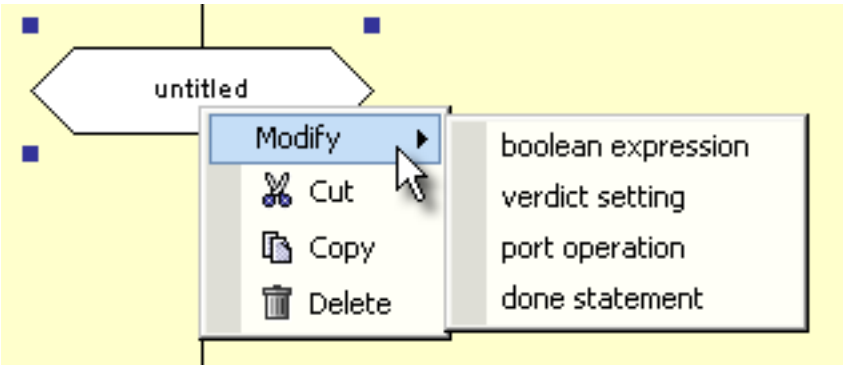
Conditions

The GFT condition symbol may be used for TTCN-3

- guarding boolean expressions (for **alt** and **call** inline expressions)
- verdict setting
- port operations (**start**, **stop** and **clear**)
- **done** statement

To change the type of condition, right-click the condition symbol and choose a condition type from the Modify popup menu.

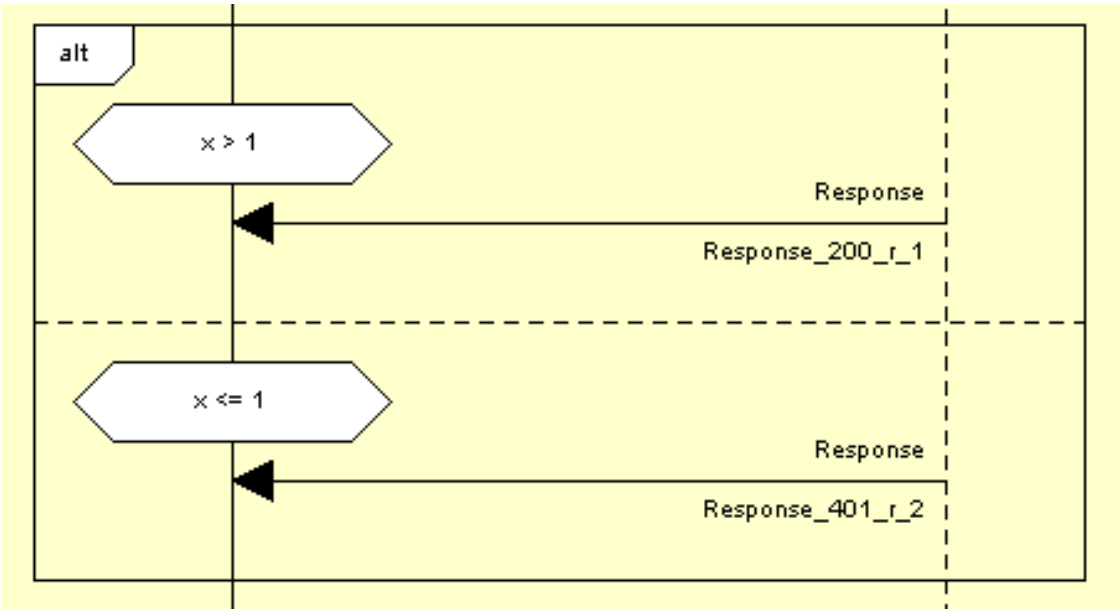
Figure 6.56. Modify the Condition Type



Guarding Boolean Expression Statements

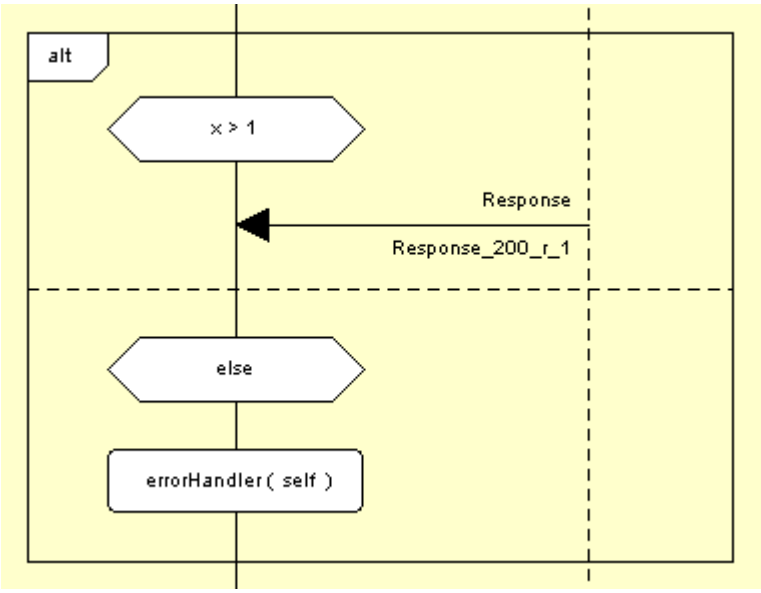
It is possible to disable/enable an alternative operands by means of a boolean expression contained within a condition symbol placed upon the test component instance.

Figure 6.57. Disable/Enable an Alternative



A special usage of guarding boolean expressions is the **else** branch of an alternative. It shall be denoted using a condition symbol placed upon the test component instance axis labeled with the **else** keyword.

Figure 6.58. Else Branch of an Alternative



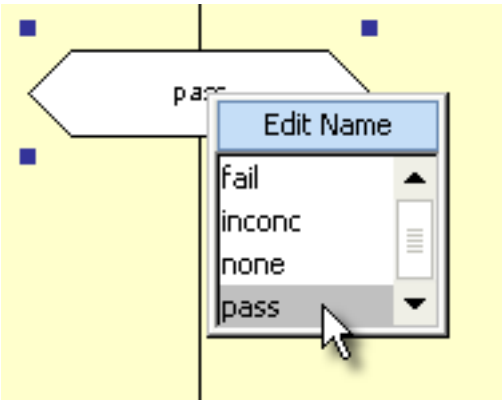
Verdict Setting Statements

The verdict set operation **setverdict** is represented in GFT with a condition symbol within which

- the values **pass**, **fail**, **inconc** or **none** are denoted.
- a TTCN-3 core language **setverdict** statement is denoted.

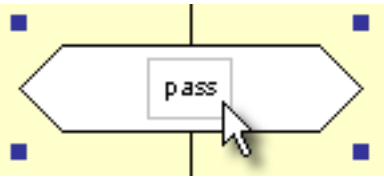
To choose one of the verdict to be set, left-click the verdict editing area of the verdict condition symbol (gray rectangle), and choose one of **pass**, **fail** , **inconc**, or **none** from the popup menu.

Figure 6.59. Popup Menu for Verdict Setting

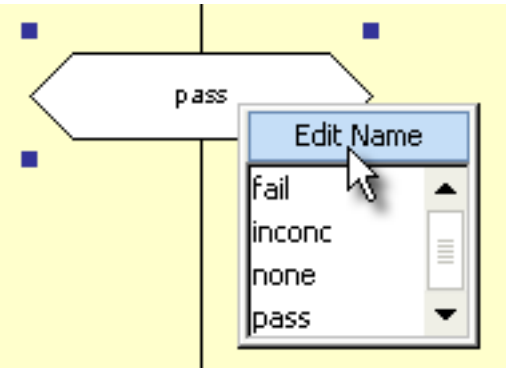


To set a verdict based on a TTCN-3 single expression

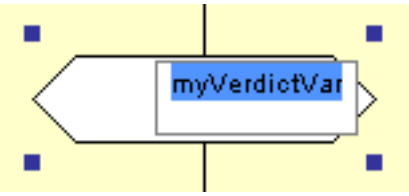
1. Left-click the verdict editing area of the verdict condition symbol (gray rectangle)



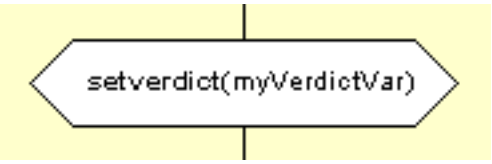
2. Select Edit Name from the popup menu



3. Specify a single expression that results to a valid TTCN-3 verdict



4. Type **Enter** to complete your input



Port Operation Statements

The port operation statements are represented in GFT with a condition symbol within which the following values are denoted:

- **start**

  The **start** port operation shall be represented by a condition symbol with the keyword **start** . It is attached to the test component instance, which performs the start port operation, and to the port that is started.

- **stop**

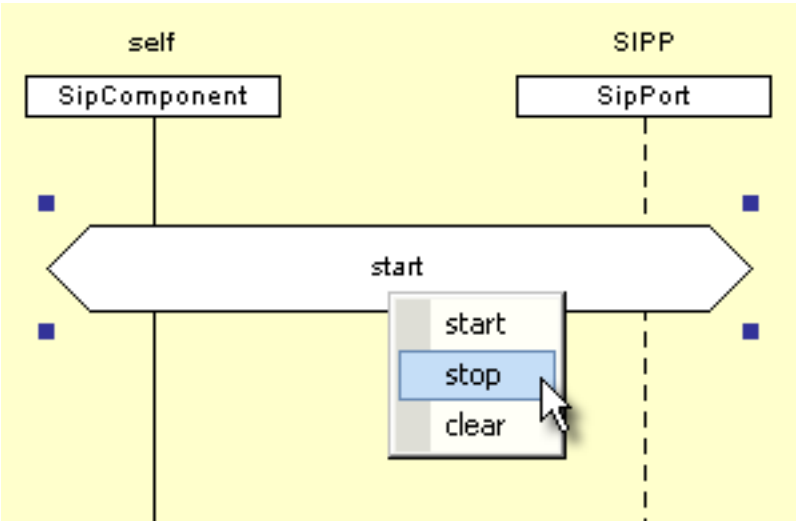
  The **stop** port operation shall be represented by a condition symbol with the keyword **stop** . It is attached to the test component instance, which performs the stop port operation, and to the port that is stopped.

- **clear**

  The **clear** port operation shall be represented by a condition symbol with the keyword **clear** . It is attached to the test component instance, which performs the clear port operation, and to the port that is cleared.

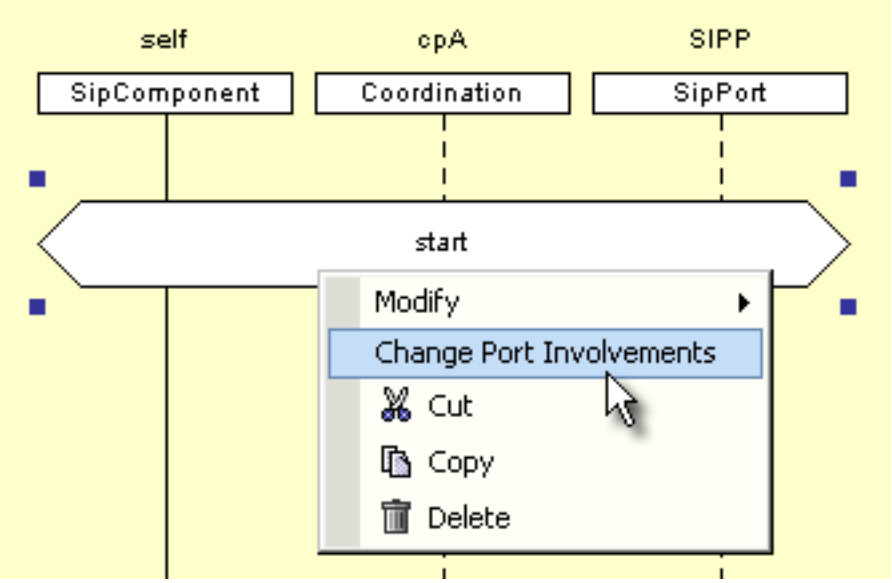
To choose one of the port operations to be set, left-click the port operation editing area of the port operation condition symbol (gray rectangle), and choose one of **start**, **stop** , or **clear** from the popup menu.

Figure 6.60. Popup Menu for Port Operation Setting



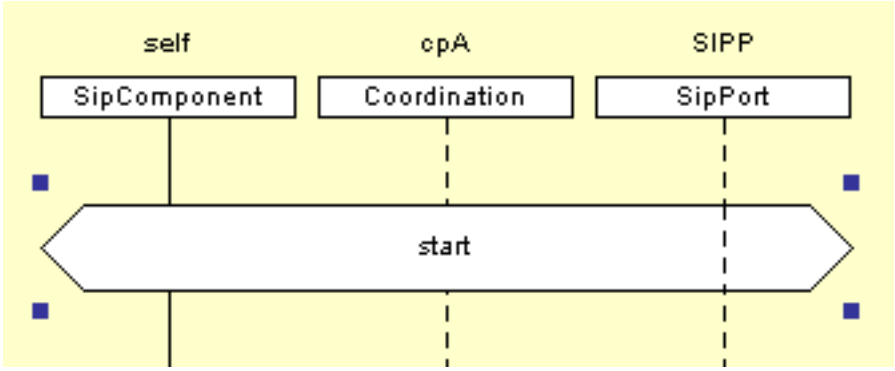
To change the involvement of a port into a port operation perform the following steps:

1. Right-click the port operation condition symbol and select Change Port Involvements from the popup menu



2. Select which ports should involved in the port operation

3. The dashed port instance axis indicates which ports are not involved in this port operation



Note

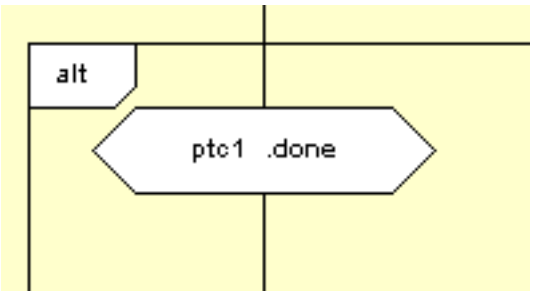
The **all** keyword for ports together with the **clear**, **start** and **stop** operation is represented by attaching the condition symbol containing the **clear**, **start** or **stop** operation to all port in-

stances represented in the GFT diagram for a testcase, function or altstep, if all port instances of the corresponding component type declaration are drawn.

Done Statements

The **done** operation shall be represented within a condition symbol, which is attached to the test component instance, which performs the **done** operation. The condition symbol contains the **done** statement referencing the component whose termination is awaited.

Figure 6.61. Done



Note

The **any** and **all** keywords can be used for the **running** and **done** operations but from the MTC instance only. They have no graphical representation, but are textually denoted at the places of their use, i.e. in GFT action symbols.

Messages

Generally GFT messages are used to specify all kinds of communication operations. Communication operations are structured into two groups:

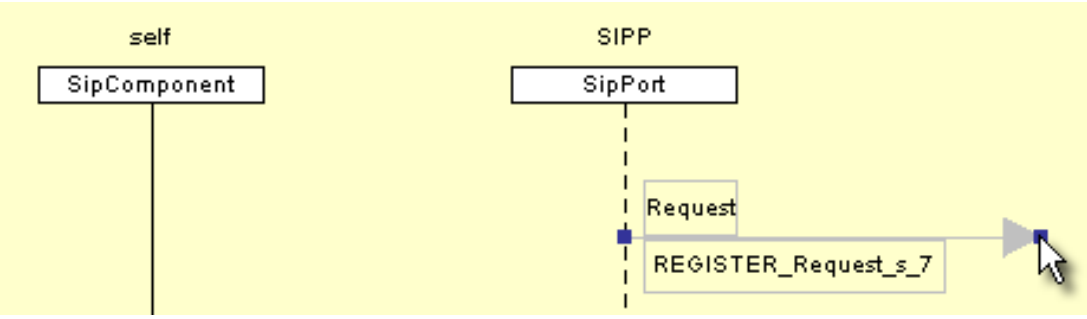
- Sending operations

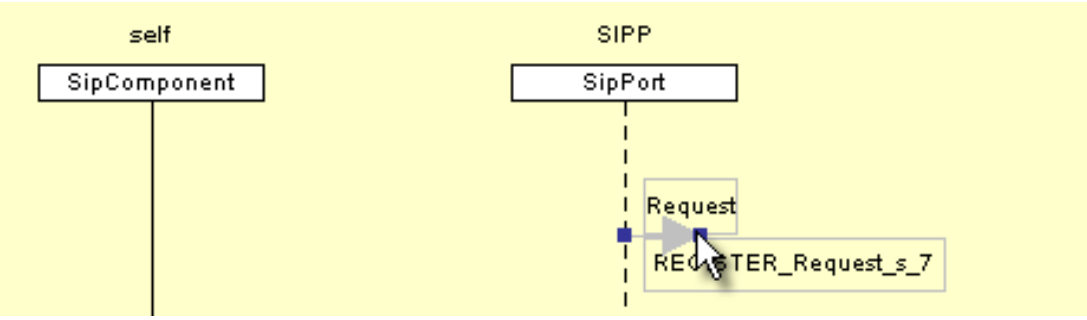
  a test component sends a message (send operation), calls a procedure (call operation), replies to an accepted call (reply operation) or raises an exception (raise operation)

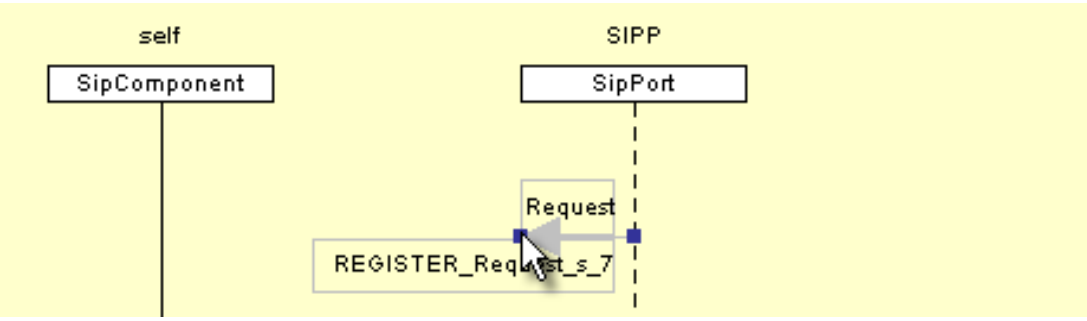
- Receiving operations

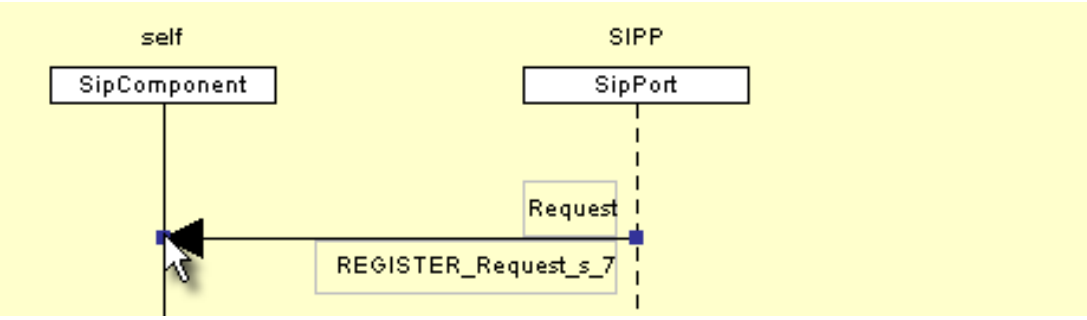
  a component receives a message (receive operation), accepts a procedure call (getcall operation), receives a reply for a previously called procedure (getreply operation) or catches an exception (catch operation)

The direction of a message symbol can be changed by dragging the arrowhead of the message symbol and drag it in the opposite direction.

1. 

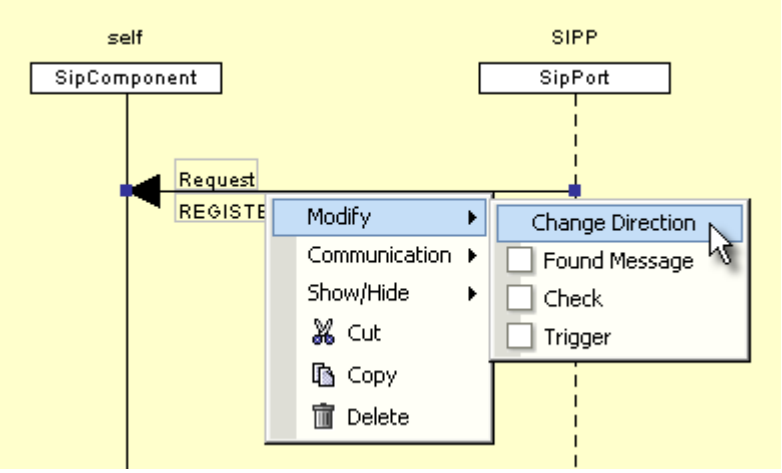
2. 

3. 

4. 

It is also possible to change the direction of a message symbol via popup menu. To do so:

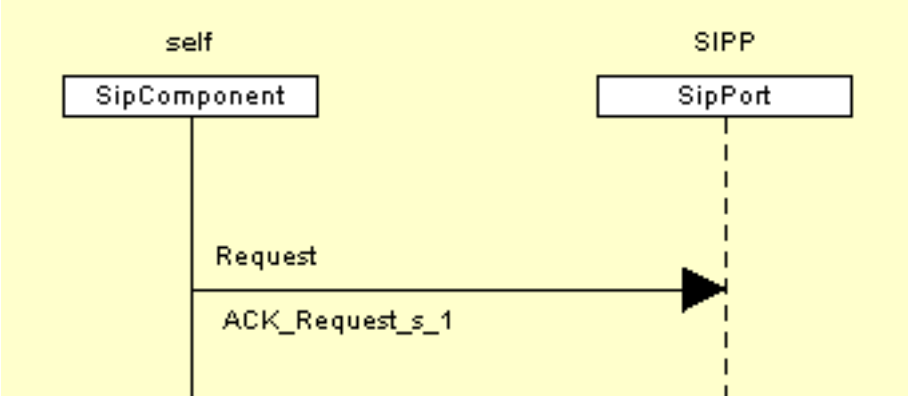
1. Right-click the message symbol
2. Select Modify followed by Change Direction from the popup menu



Message-based Communication

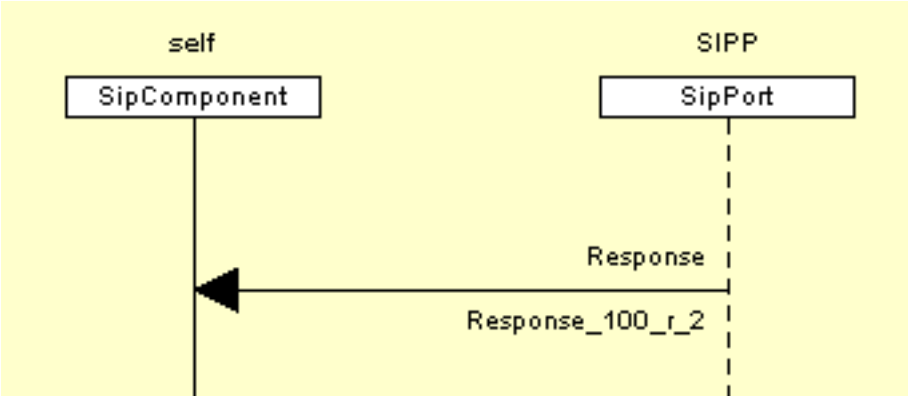
- The **send** operation shall be represented by an outgoing message symbol from the test component to the port instance. The optional type information shall be placed above the message arrow. The (inline) template shall be placed underneath the message arrow.

Figure 6.62. Send Operation



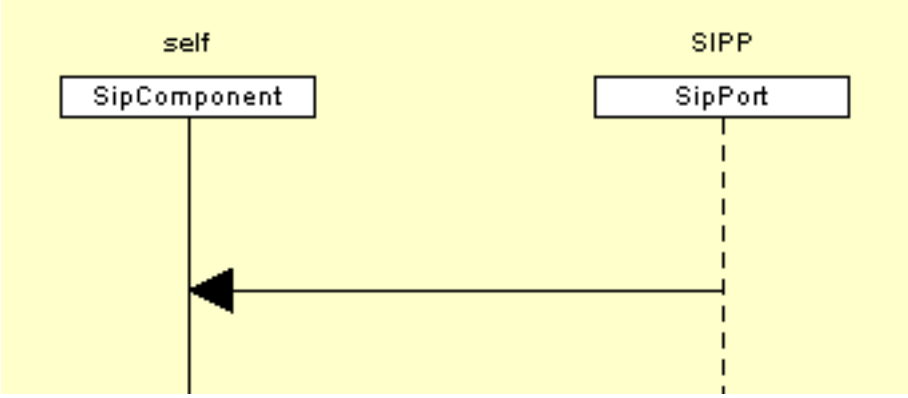
- The **receive** operation shall be represented by an incoming message arrow from the port instance to the test component. The optional type information shall be placed above the message arrow. The (inline) template shall be placed underneath the message arrow.

Figure 6.63. Receive Operation



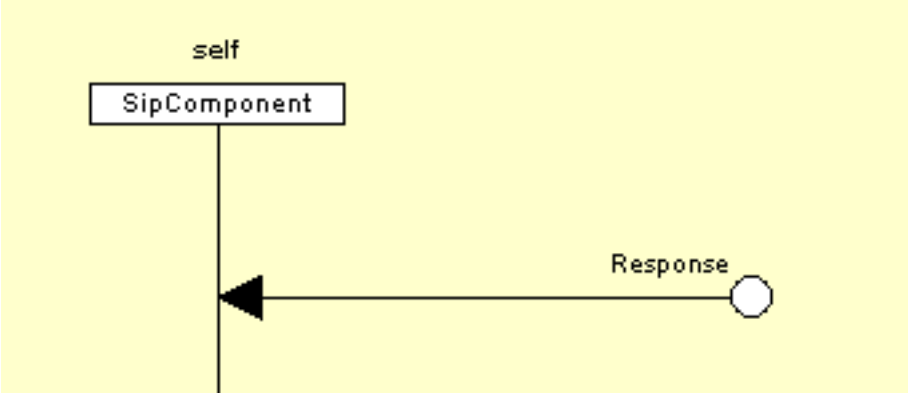
The receive any message operation shall be represented by an incoming message arrow from the port instance to the test component without any further information attached to it.

Figure 6.64. Receive Any Message Operation



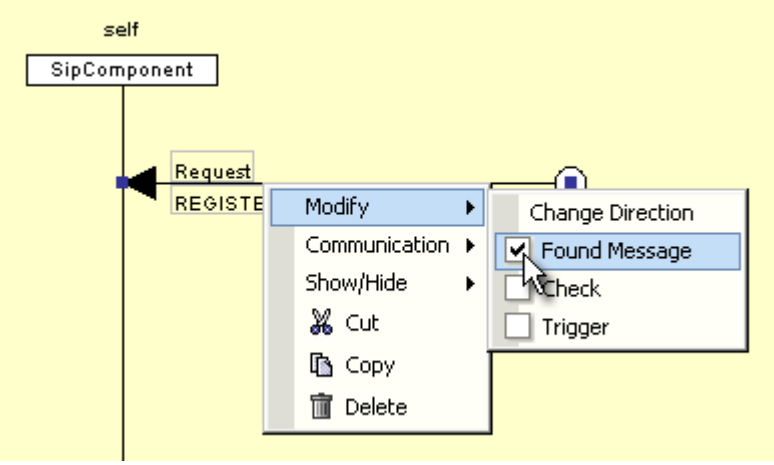
The receive on any port operation shall be represented by a found message symbol representing any port to the test component

Figure 6.65. Receive on Any Port



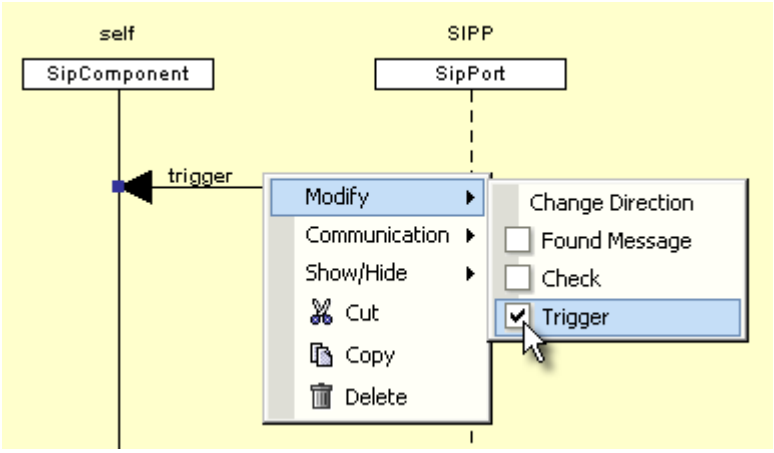
To change the message symbol to found message symbol:

1. Right-click the message symbol
2. Select Modify from the popup menu
3. Check Found Message



- The **trigger** operation shall be represented by an incoming message arrow from the port instance to the test component and the keyword **trigger** above the message arrow preceding the type information if present. The optional type information is placed above the message arrow subsequent to the keyword **trigger**. The (inline) template is placed underneath the message arrow.

  For switching to a **trigger** operation, right-click the message symbol, select Modify from the popup menu and check Trigger.



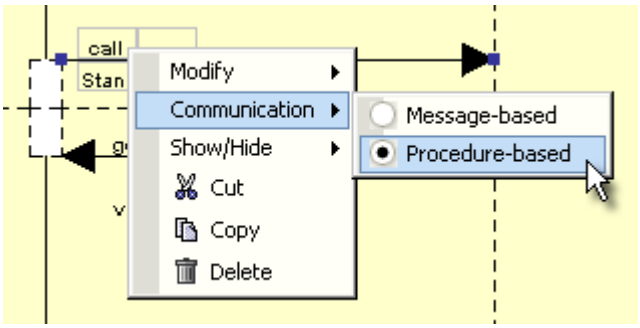
Note

The message has to be in message-based communication mode for a **trigger** operation.

Procedure-based Communication

To toggle between message-based and procedure-based communication:

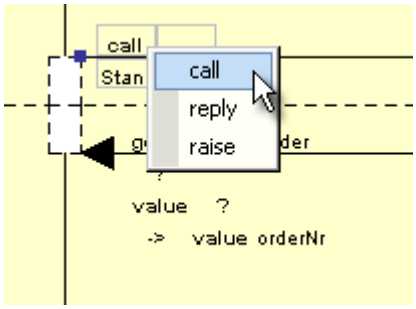
1. Right-click the message symbol
2. Select Communication followed by Message-based or Procedure-based from the popup menu.



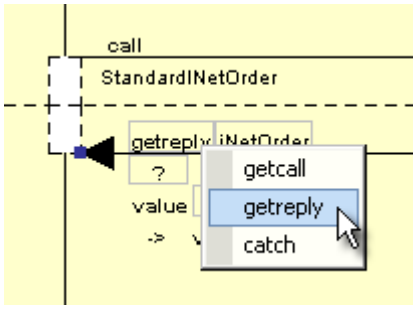
The following port operations are possible, depending on the situation:

- **call**
- **reply**
- **raise**

To change the port operation left-click on the message port operation label and choose the desired one.



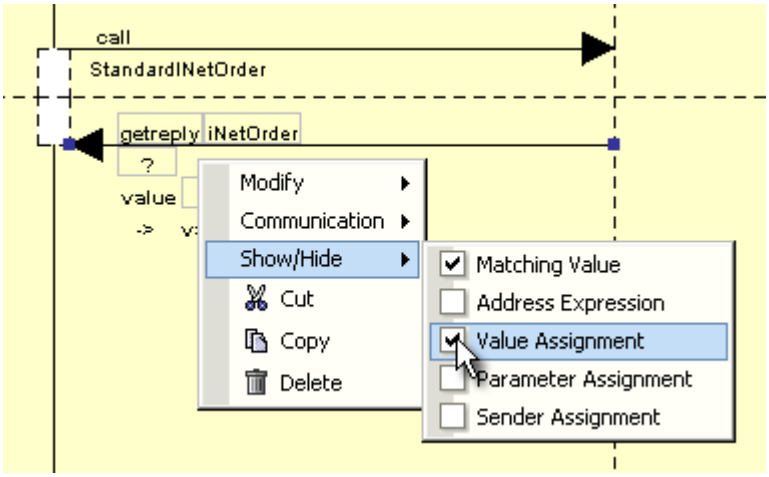
- **getcall**
- **getreply**
- **catch**



Note

It is possible to show or hide the optional matching value, address expression, value, parameter and sender assignment. To do so right-click on the message, select Show / Hide from

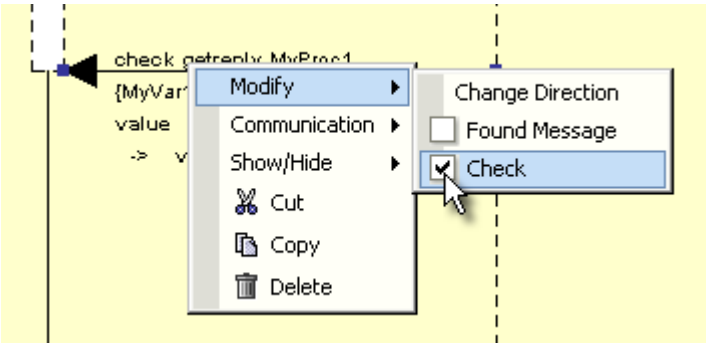
the popup menu and toggle the respective checkbox (see the section called “Showing/Hiding Optional Symbol Attributes”).



Check Operation

The **check** operation shall be represented by an incoming message arrow from the port instance to the test component. The keyword **check** shall be placed above the message arrow. The attachment of the information related to the **receive**, **getcall**, **getreply** and **catch** follows the check keyword and is according to the rules for representing those operations.

For switching to a **check** operation, right-click the message symbol, select Modify from the popup menu and select Check.

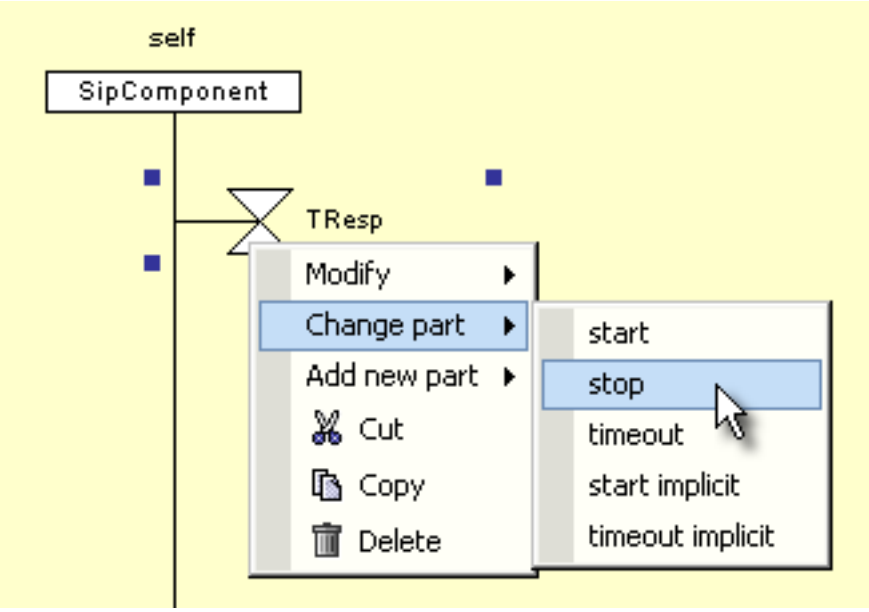


Timers

In GFT, there are two different timer symbols: one for identified timers and one for call timers (implicit timers). They differ in appearance as solid line timer symbols are used for identified timers and dashed timer symbols for call timers. An identified timer shall have its name attached to its symbol.

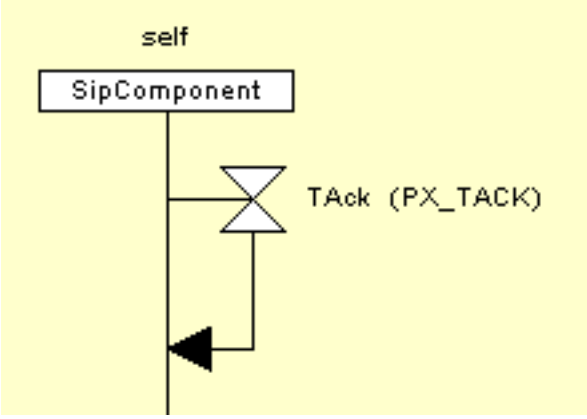
To change the timer symbol type:

1. Right-click the timer symbol
2. Select Change Part from the popup menu
3. Select one of the timer symbol types



For combinations of start timer operation with stop/timeout timer operation the connected timer symbol can also be used. There the start operation and the stop/timeout operation is connected with a vertical line.

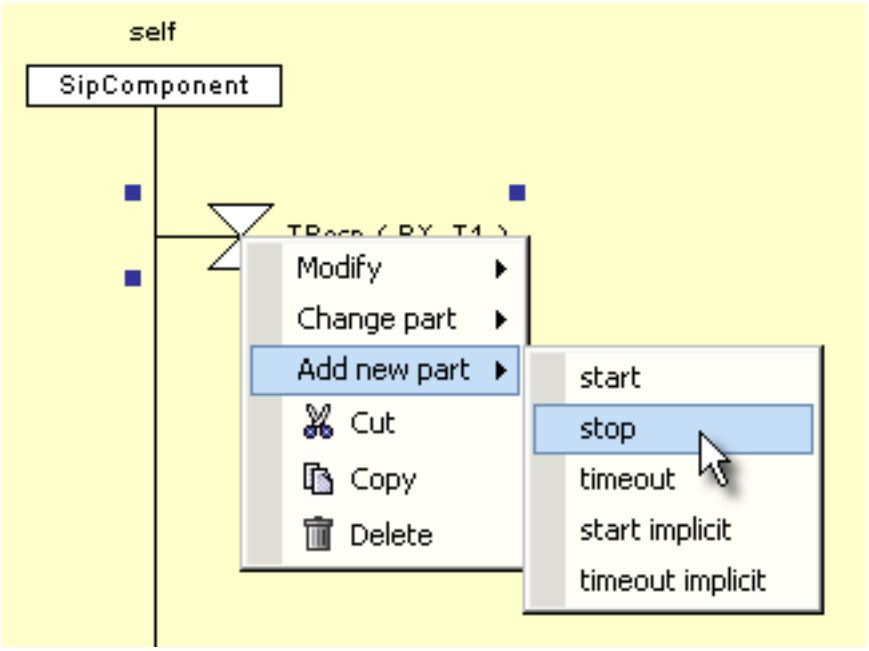
Figure 6.66. Connected Start Timer Stop/Timeout Timer Operation



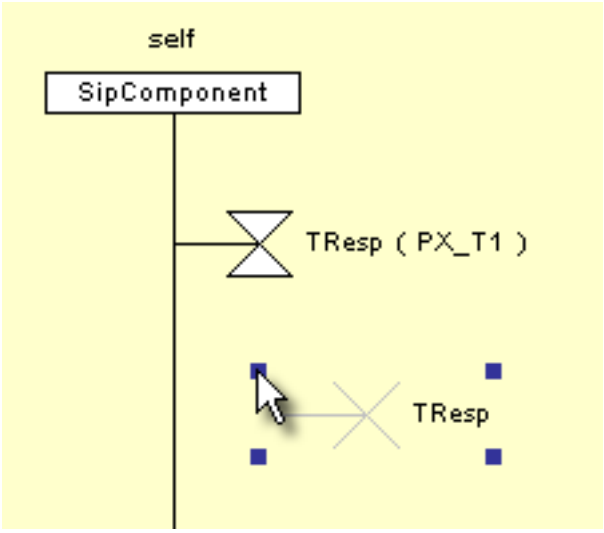
Note

Additional timer symbols for the same timer may be added via:

1. Right-click the timer symbol
2. Select Add New Part from the popup menu
3. Select one of the timer symbol types



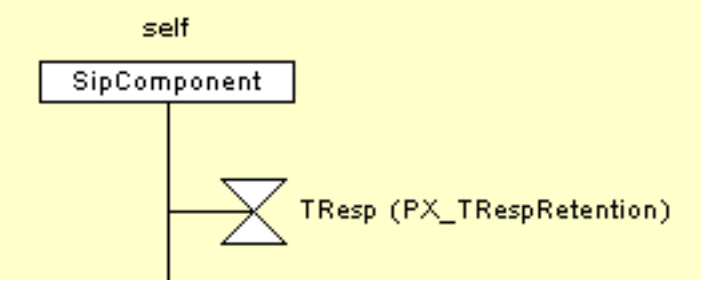
4. Left-click to add the timer symbol



Start Timer Operation

For the start timer operation, the start timer symbol shall be attached to the component instance. A timer name and an optional duration value (within parentheses) may be associated.

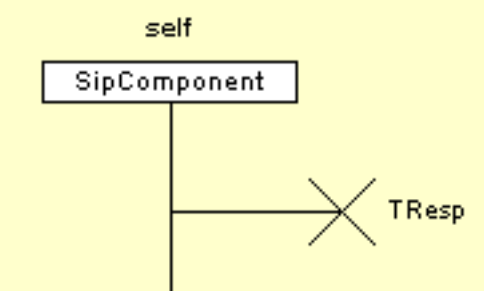
Figure 6.67. Start Timer Operation



Stop Timer Operation

For the stop timer operation, the stop timer symbol shall be attached to the component instance. An optional timer name may be associated.

Figure 6.68. Stop Timer Operation

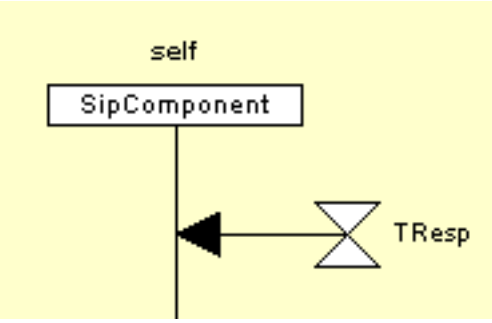


The stop timer operation can be applied to **all** timers, if the timer name is omitted

Timeout Timer Operation

For the timeout operation, the timeout symbol shall be attached to the component instance. An optional timer name may be associated.

Figure 6.69. Timeout Timer Operation



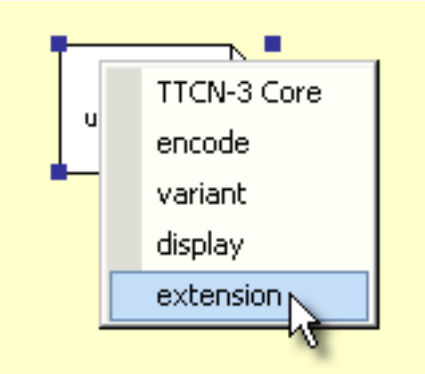
The timeout operation can be applied to **any** timer, if the timer name is omitted.

Text Symbols

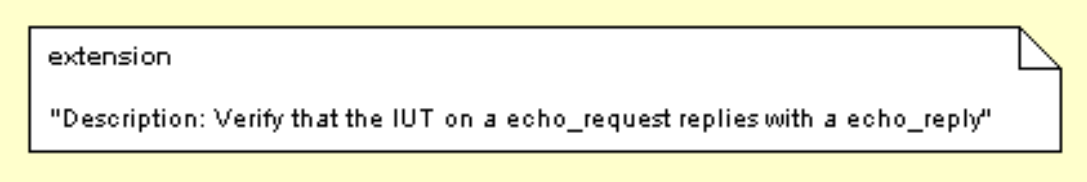
The attributes defined for the module control part, testcases, functions and altsteps are represented within the text symbol. The syntax of the **with** statement is placed within that symbol.

1. Left-click the upper text field of the text symbol

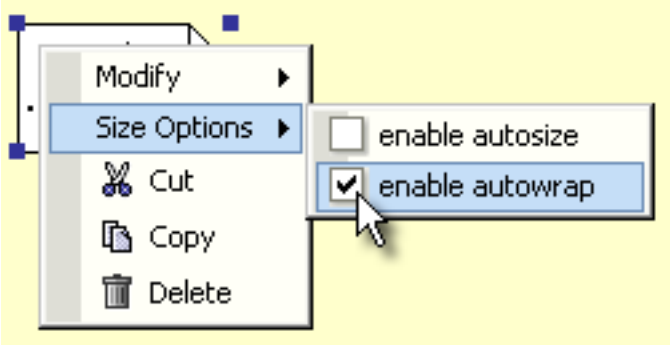
2. Select one of the attributes from the popup menu



3. Specify the attribute



If an attribute is selected, autosize and autowrap can be enabled. To do so right-click the text symbol, select Size Options from the popup menu and toggle the respective checkboxes.



Note

Text symbols may also be used to specify comments (see the section called “Comments”).

GFT Example

This chapter will give an example (restaurant example) on how to use GFT Editor. It will guide the user to create graphically a complete test suite on the basis of GFT (Graphical Presentation Format for TTCN-3).

Control Diagram

First of all a control diagram  is needed. A control diagram provides a graphical presentation of the control part of a TTCN-3 module. Therefore the new TTCN-3 Module wizard can be used, which is accessible over **File > New > Other > TTCN-3 > GFT Module**. Press Next, enter a container and a file name and press Finish .

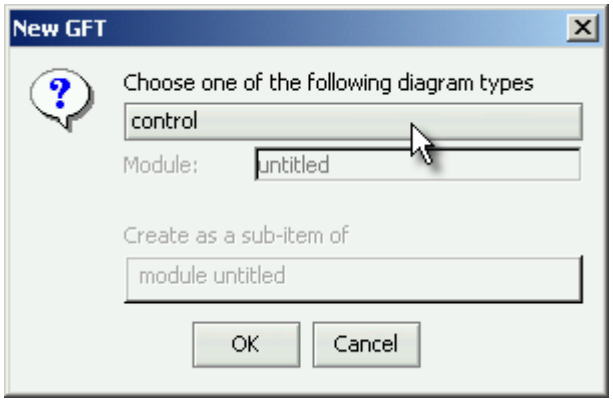
Choose **control** from the diagram dialog box and press Ok.

Note

The module name will be given later on by saving the project.

A control diagram cannot be placed inside a group.

Figure 6.70. Control



A new module containing a control diagram was created.

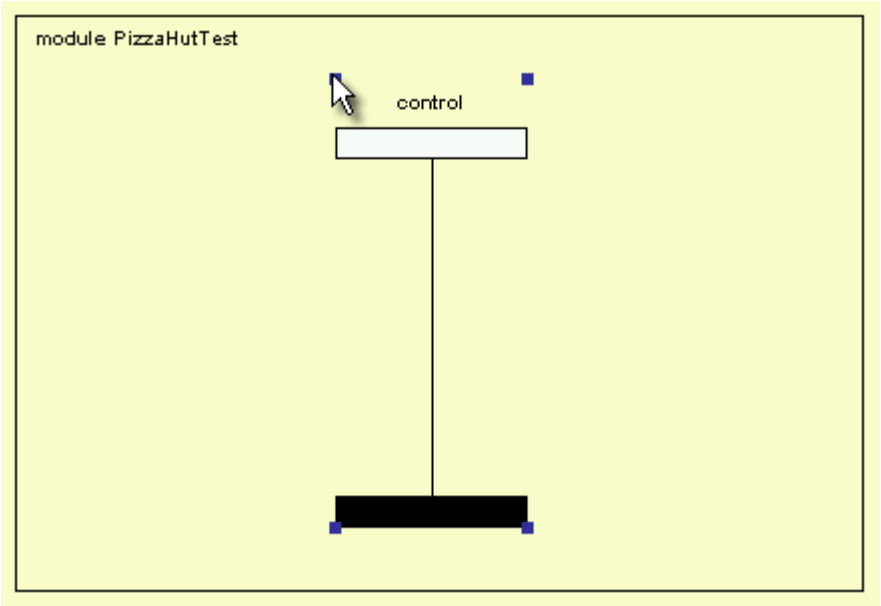
To save the project, press the Save button  in the button bar, Save menu, or just hit **Ctrl**+**S**. When asked for the file name, type in "PizzaHutTest" and save. The module name and the control diagram name changes to "PizzaHutTest". Maximize the internal frame of the control diagram.

To describe the behavior of the TTCN-3 module control part, a control instance is needed. Click on the Instance button  in the tools palette to choose a control instance. For placing it, left-click in the control diagram.

Note

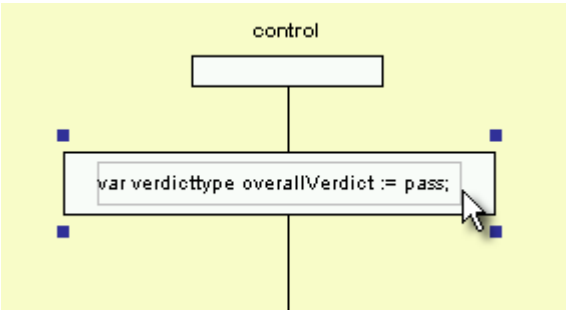
Only one control instance shall exist within a GFT control diagram. A control instance describes the flow of control of a module control part. A GFT control instance shall graphically be described by a component instance symbol with the mandatory name **control** placed on top of the instance head symbol. This name cannot be changed. No instance type information is associated with a control instance.

Figure 6.71. Control Instance



We now declare a variable overallVerdict of type **verdicttype** with initialization, that represents the overall result of the test suite. The type **verdicttype** is a TTCN-3 specific predefined basic type, that can have five possible values: **pass**, **fail**, **inconc**, **none** and **error**. This declaration shall be made within an action symbol. Select an action symbol  from the tools palette and attach it to the control instance. Left-click in the action symbol and enter the following declaration: "**var verdicttype** overallVerdict := **pass**;". To center the action symbol, left-click on it and drag it.

Figure 6.72. Declaration



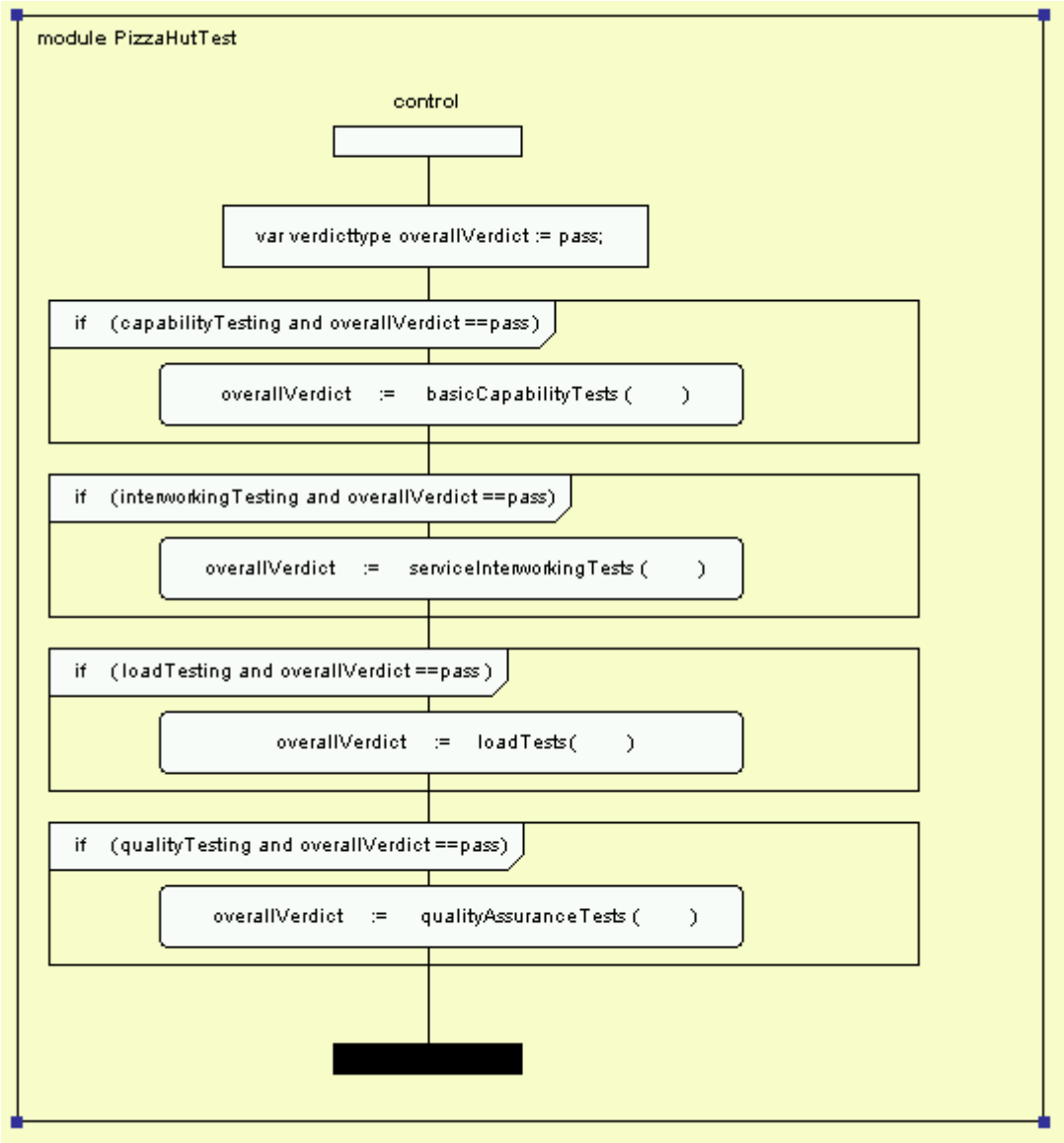
Now we call functions with a return value. These functions are invoked inside an if inline expression. Select an inline expression symbol  from the tools palette and attach it to the control instance below the action symbol. To change the type, left-click on the inline expression label and choose **if**. Left-click on the expression field (between the parenthesis) and enter: "capabilityTesting and overallVerdict := pass".

It's possible to add a comment for each symbol. Select the **if** inline expression and enter the following comment for this block: "Basic Capability Tests".

We now create an invocation of a function, which is represented by a reference symbol. Select a reference symbol  from the tools palette and attach it to the control instance inside the if inline expression

symbol. Change the function name (default is "untitled") by left-clicking on this text field. Select **Edit Name** and enter "basicCapabilityTests". To assign the return value, right-click on the reference symbol, select Show / Hide from the popup menu and select the **Variable** checkbox. In the new text field for the variable (left-most), enter "overallVerdict". Enlarge the inline expression symbol and center the reference. In the same way create the remaining inline expressions and references like shown below.

Figure 6.73. Complete Control Diagram



Note

If a symbol is selected, it can be enlarged by dragging on the four handles (here useful for the frame of the control diagram, the control instance and the inline expressions).

Invoking Functions

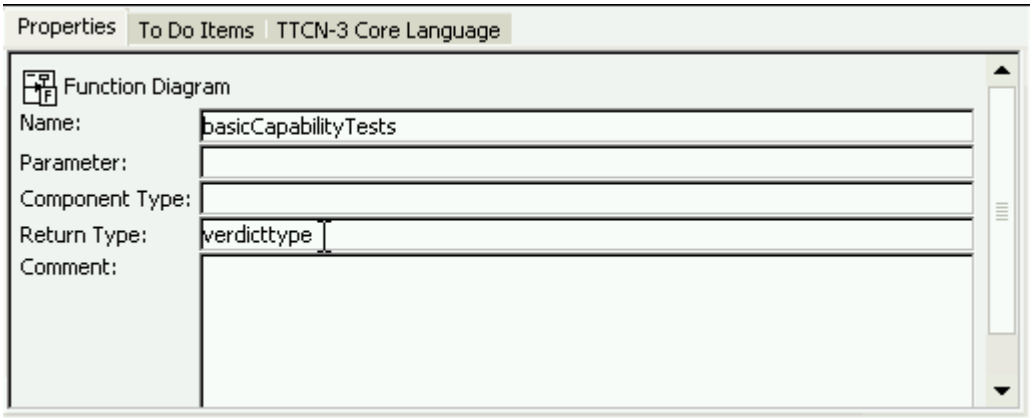
In the control diagram four invoking functions can be found: basicCapabilityTests(), serviceInterworkingTests(), loadTests() and qualityAssuranceTests(). To create one of these functions, choose the Add GFT

button  in the button bar, Add GFT menu, or just hit **Alt**+**N**. Choose **function** from the diagram dialog box, enter the function name: "basicCapabilityTests" and press Ok. Maximize the internal frame. In the properties below the editor panel, a return type for the function can be specified: "**verdicttype**".

Note

The function has to be selected to show the respective properties; left-click on the frame of the function.

Figure 6.74. Property Panel of a Function



To create the test component, click on the Instance button  in the tools palette and place the instance in the function diagram.

Note

The name associated with the test component instance shall be **self**. It is placed on the top of the instance head symbol automatically and cannot be changed.

Now we declare a variable localVerdict of type **verdicttype** with initialization, that represents the returned verdict. Therefore choose an action symbol  from the tools palette, place it over the control instance and enter the declarations:

var verdicttype localVerdict := **pass**;

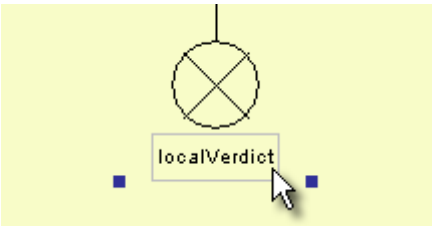
var integer nrP := 0, nrF := 0, nrI := 0;

Note

A sequence of declarations, statements and operations can be specified in a single action symbol. It is not necessary to use a separate action symbol for each declaration, statement or operation.

Now right-click on the instance to change the instance type and select **return**. At the bottom of the instance symbol, the return statement can be entered: "localVerdict".

Figure 6.75. Return Statement of an Instance

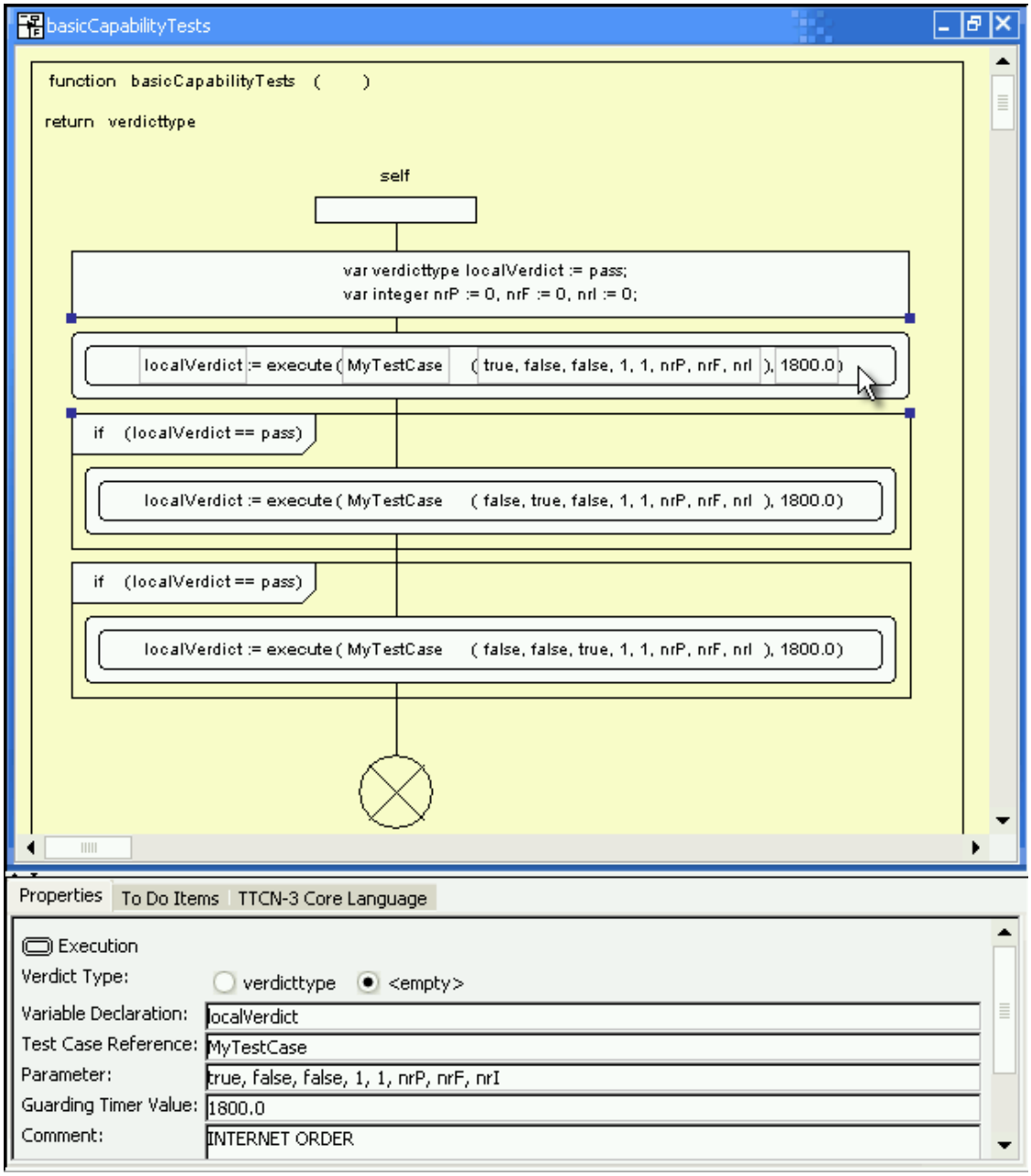


The execution of test cases is represented by use of the execute test case symbol . Select an execution in the tools palette and attach it to the component instance below the action symbol. The name of the test case to execute is "MyTestCase", so replace the initial name "untitled" for the test case reference by "MyTestCase": left-click on the respective text field of the execution symbol or enter the test case reference in the property panel on the bottom. To specify optional parameters, right-click on the execution symbol, select Show / Hide from the popup menu and select the **Parameter** checkbox. Enter the parameter: "true, false, false, 1, 1, nrP, nrF, nrI" in the originated text field (between the parenthesis). It's possible to specify the parameter as well as all other attributes at the property tab on the bottom too. To specify a time supervision, edit the text field Guarding Timer Value in the properties tab; enter "1800.0". To assign the returned verdict to our **verdicttype** variable, enter "localVerdict" in the Variable Declaration of the property panel. Finally we want to enter a comment to point up the content. Enter "INTERNET ORDER" in the comment field of the respective property tab.

If the executed test case has passed so far, we want to execute another test case. An **if** statement is represented by an inline expression symbol labeled with the **if** keyword and a Boolean expression. Select an inline expression from the tools palette and attach it to the component instance below the execution symbol. Change the label to **if** by left-clicking on the name and selecting **if** from the context menu. Now left-click in the expression text field (between the parenthesis) and enter the Boolean expression "localVerdict == pass". Again we want to enter a comment here. Enter "PHONE ORDER" in the comment field of the property tab of the inline expression. Enlarge the inline expression by using the four handles. Now it's useful to copy the adapted expression symbol from above: right-click the expression symbol and select **Copy** or just hit **Ctrl**+**C**, if the expression symbol is selected. To paste the symbol, hit **Ctrl**+**V**. Attach the expression symbol to the component instance inside the inline expression symbol. Change the parameter to "false, true, false, 1, 1, nrP, nrF, nrI" by left-clicking on the parameter field.

Continue with the third execution similar as described above.

Figure 6.76. Function basicCapabilityTests



Go ahead with the other three invoking functions loadTests(), serviceInterworkingTests() and qualityAssuranceTests(). They are very similar to the described invoking function. All invoking functions can be seen in the provided example.

Main Test Case

A GFT test case diagram provides a graphical presentation of a TTCN-3 test case. For creating a test case  , press the Add GFT button  in the button bar, Add GFT menu, or just hit **Alt**+**N**. The diagram type **testcase** is already chosen, so we only have to specify the name: "MyTestCase" and press Ok. Maximize

the internal frame. Left-click on the parameter field of the test case (between the parenthesis) in the editor pane. Specify the parameter: "in boolean internetService, ..., inout integer nrInc". The component type can be entered in the left-most text field (after **runs on**); enter "MtcType". The last text field indicates the system type; enter "TestSystemType".

A GFT test case diagram shall include one test component instance describing the behavior of the **mtc** (also called mtc instance) and one port instance for each port owned my the **mtc**. For creating a test component instance, left-click on the Instance button  in the tools palette and place the instance in the test case diagram.

Note

The name associated with the mtc instance shall be **mtc**. It is placed on the top of the instance head symbol automatically and cannot be changed.

The optional test component type can be entered within the instance head symbol: "MtcType". It is already specified, if the component type was given in the test case diagram.

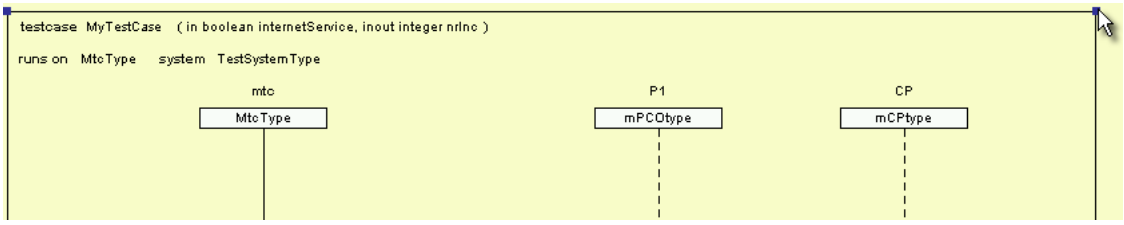
Note

Anytime there is consistency between the test component type after the **runs on** keyword in the heading of the GFT diagram and the component type of the test component instance.

Test component instances describes the flow of control for the test component that executes a test case, function or altstep. Port instances represents the ports used by the different test components. Here we have two ports. Left-click on the Port Instance button  in the tools palette and place a port instance in the test case diagram to the right of the component instance. A port instance has a dashed instance line. Enter the port name "P1" on top of the port instance head symbol. Enter the optional port type "mPCOtype" within the port instance head symbol.

In the same way create another port instance to the right of the other instances with the port name "CP" and the port type "mCPtype". Enlarge the test case diagram if necessary by selecting it and dragging on the four handles.

Figure 6.77. Test Component Instance and Port Instances

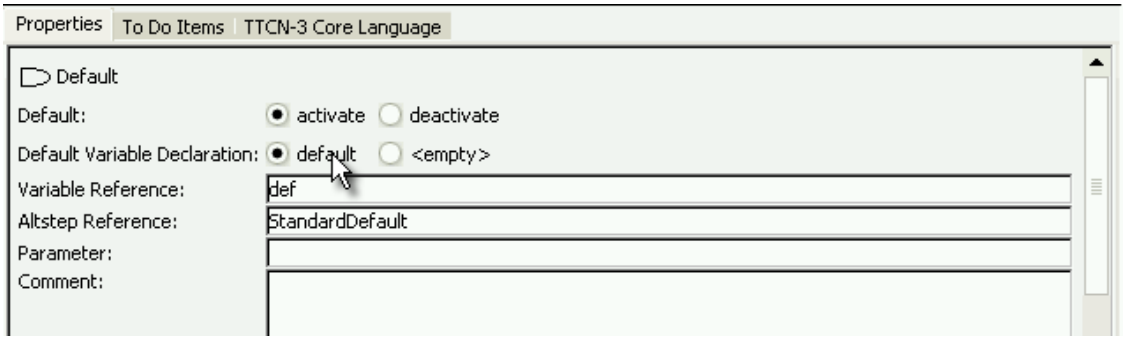


Now we declare a variable without initialization. Select an action symbol  from the tools palette and attach it to the component instance; enter "var ReportType report;".

Variables of type **default** can either be declared within an action symbol or within a default symbol as part of an activate statement. For the activation of a default, left-click on the Default symbol  in the tools palette and attach it to the component instance. Enter the altstep reference "StandardDefault" and

the variable reference "def" in the property panel of the activation below the editor panel. Here also the default variable declaration can be set to **default** by selecting the **default** toggle button.

Figure 6.78. Property Panel of an Activation



Note

The ports of a test component can be connected to other components or to the ports of the test system interface. In the case of connections between two test components the **connect** operation shall be used. When connecting a test component to a test system interface the **map** operation shall be used. The **connect** and **map** operations shall be represented within an action box symbol.

Left-click on the Action button  in the tools palette, attach it to the component instance and enter the map statement "map(self:P1, system:mPCO)".

An **if** statement is represented by an inline expression symbol  labeled with the **if** keyword and a boolean expression. Select an inline expression  and attach it to the component instance beneath the action symbol. Enlarge the inline expression to attach it also to all port instances. Change the name by left-clicking on the name field of the symbol placed in the top left hand corner and selecting if from the popup menu. Enter the boolean expression "internetService" for the expression; internetService is a boolean variable.

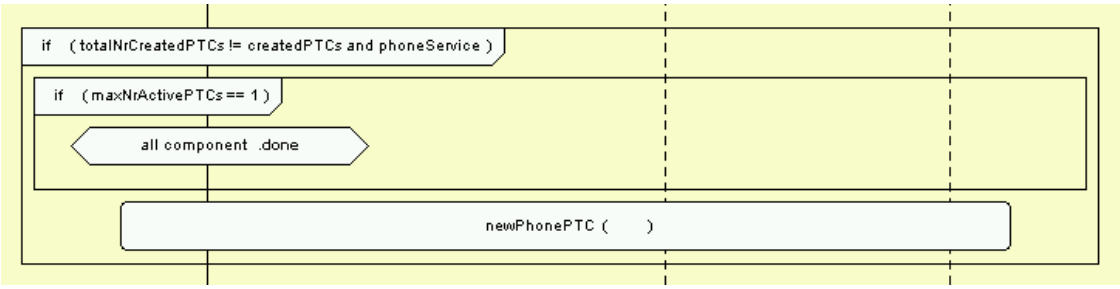
Inside the inline expression  we reference to another function; select a reference  , place it inside the inline expression and enter the function name "newInternetPTC".

It is possible to create nested inline expressions. Add an **if** inline expression  with the boolean expression "totalNrCreatedPTCs != createdPTCs and phoneService" as described above. Place a second **if** inline expression  with the expression "maxNrActivePTCs == 1" inside the first one.

The **done** operation shall be represented within a condition symbol  , which is attached to the test component instance, which performs the **done** operation. The **any** and **all** keywords can be used for the **running** and **done** operations but from the MTC instance only. Left-click on the Condition button  in the tools palette and place the condition inside the inline expressions. Right-click the condition symbol, select Modify from the popup menu and select done statement. Now enter the done statement "all compo-

nent". Inside the first but below the second inline expression a further reference will be added with the function/altstep name "newPhonePTC".

Figure 6.79. Nested Inline Expressions

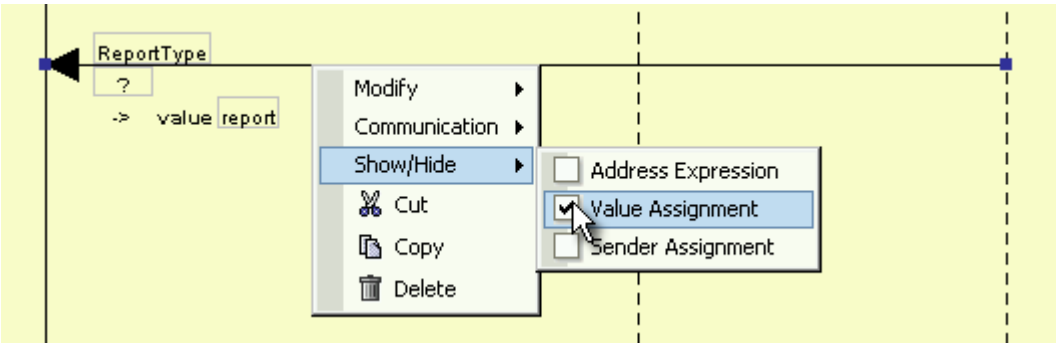


We now use a while statement. A **while** loop is executed as long as the loop condition holds. Choose again an inline expression and attach it to the component instance and all port instances beneath the **if** inline expressions. Left-click on the name field placed in the top left hand corner and select while from the popup menu. Enter the boolean expression "totalNrCreatedPTCs != createdPTCs".

To represent alternative behavior, select a new inline expression and place it inside the **while** inline expression; **alt** is the default selection.

To receive a value from an incoming message port queue, a receive message is needed. Left-click on the message icon in the tools palette and place it in the testcase diagram. Right-click on the message symbol, select Modify from the popup menu and select Change Direction. Attach the receive side of the message to the component instance inside the **alt** inline expression by dragging the message. Now attach the send side to the port instance "CP" by enlarging the message via the handles. To specify the matching type, enter "ReportType" in the type field above. The template is "?". The value of the received message shall be assigned to a variable, so right-click on the message, select Show/Hide from the popup menu, select Value Assignment and enter "report" in the new value assignment field or just enter "report" in the value assignment field of the property panel.

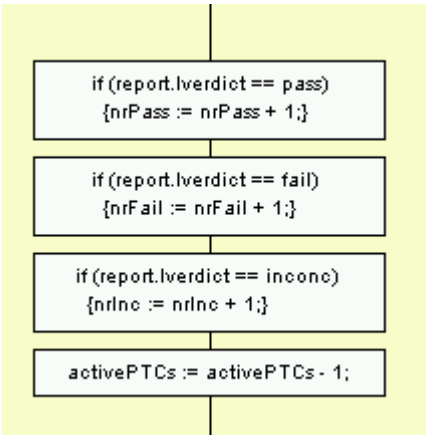
Figure 6.80. Message Value Assignment



The verdict set operation **setverdict** is represented in GFT with a condition symbol . Select a condition from the tools palette and attach it to the component instance beneath the message. The verdict setting mode is the default. Left-click in the condition symbol and edit the name: "report.lverdict".

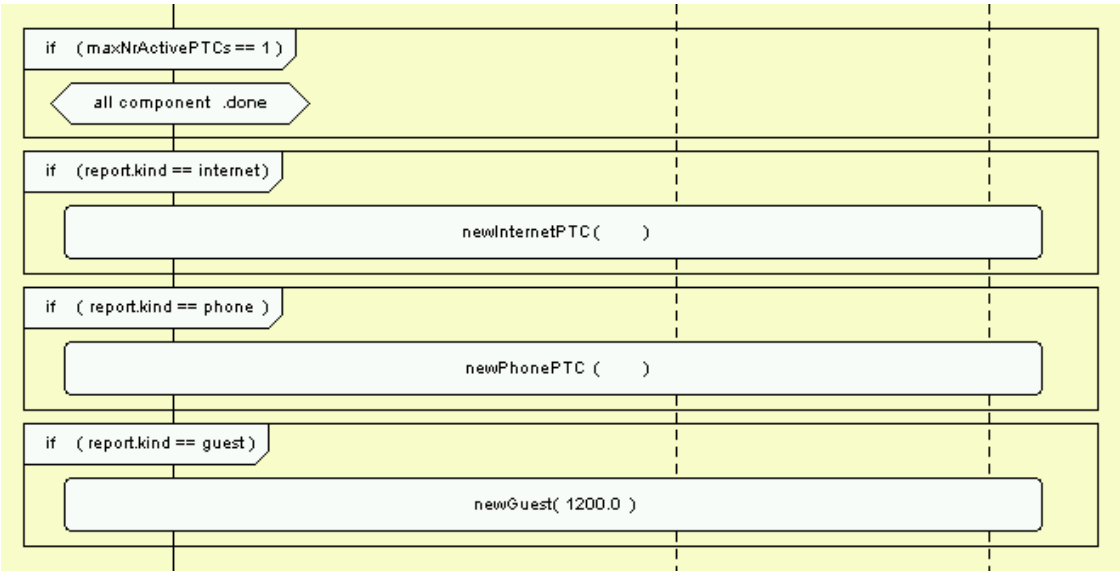
Sequences of statements in TTCN-3 core language can be entered in action symbols  .

Figure 6.81. Actions



The following **if** inline expressions, conditions and references can be created as described above.

Figure 6.82. If Inline Expressions



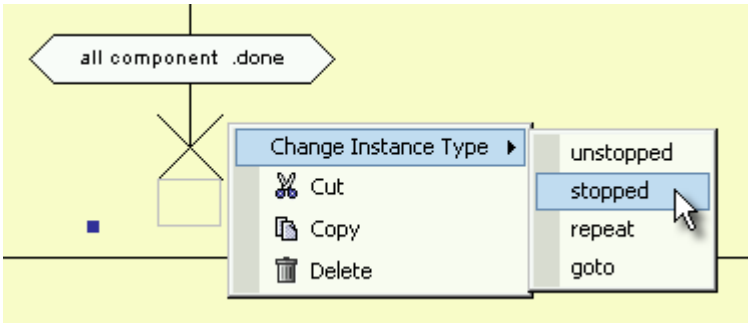
To realize the **else** branch, left-click the condition symbol  from the tools palette, place it upon the test component instance axis below the dashed separator line, right-click on the condition symbol, select Modify from the popup menu, select guarding boolean expression, left-click on the condition symbol and select the else keyword from the popup menu.

Note

An **else** branch has to be the last branch in an **alt** statement.

Beneath the inline expressions, we create again a **done** operation within a condition symbol. A **stop** execution operation is represented by a stop symbol. Right-click on the component instance symbol, select Change Instance Type and select stopped.

Figure 6.83. Add a Stop Symbol to a Component Instance



Functions and Altsteps

Functions are used to express test behavior, to organize test execution or to structure computation in a module. To create the function newInternetPTC(), add a new function diagram and enter the component type "MtcType". Create a component instance (identifier and type will be given automatically) and again the port instances "P1" and "CP" . Right-click on the component instance and change the type to return.

The MTC is the only test component which is automatically created when a test case starts. All other test components (the PTCs) shall be created explicitly during test execution by create operations. Add a creation , enter the variable "newPTC" and the component type "InternetType". Right-click on the creation symbol, select Show/Hide from the popup menu and show the Fixed Variable Declaration; enter "InternetType".

How to create actions can be seen above.

Once a PTC has been created and connected behavior has to be bound to this PTC and the execution of its behavior has to be started. This is done by using the start operation (PTC creation does not start execution of the component behavior). Create a start symbol , enter the component identifier "newPTC" and the function "internetUser".

Add the function "aGuest" with the parameter "in float eatingDur" and the component type "GuestType". Create a component instance and the port instances "P1" with type "gPCOtype" and "CP" with type "pCPtype". Set the component instance type to stopped. Declare a "timer T1" in an action . Now we add an activation symbol with a default variable declaration, a variable reference "def" and an altstep reference "GuestDefault".

To start a component timer, select a timer part from the tools palette and place it upon the test component instance axis. To change the direction, right-click on the timer symbol, select Modify from the popup menu and select Change Direction. Enter the timer name "Tvisit". Start another timer "T1" with the timer value "waitPizzaDur".

To send a message, left-click on a message symbol and place it between the component instance and the port instance "P1". Enter the template "standardPizzaOrder". To receive a message, place another message beneath. Change the direction and enter the type "PizzaType" and the template "?".

Now we want to stop the timer "T1". Right-click on the timer "T1", select Add new part and select stop. Change the direction and attach it to the component instance.

Note

When changing the timer name, the names of all parts of it (start, stop, timeout) changes too.

Create a condition symbol with the verdict setting "pass". The other symbols can be created accordingly.

To create the first message in the function "newGuest", select the message symbol from the tools palette, place it between the component instance and the port instance "P1" and change the direction. Enter the type "SeatAssignmentType", the template "?" and the value assignment "aSeat".

To create a verdict setting "inconc", select a condition symbol from the tools palette and attach it to the component instance. Now left-click on the text field of the condition symbol and select the verdict setting inconc. How to create the other symbols is described above.

Procedure-based communication can be found in the function "internetUser". Now the blocking call operation will be described. Select an inline expression symbol from the tools palette and attach it to the component instance and all port instances "P1", CP" and "P2". Change the name from alt to call by left-clicking on the text field of the inline expression. Enlarge the inline expression and right-click on it, to change the number of alternatives; enter "4". Now attach suspension region symbols on each separator line (see figure below). To send a **call** message, select a message symbol from the tools palette and attach it to the top right corner of the topmost suspension region symbol and to the port instance "P2". Select Procedure-based communication in the property panel. The port operation switches to **call**. Enter the template "StandardINetOrder".

The **call** operation has a timeout, represented by an implicit timer. Create a timer part symbol and change the direction. Right-click on the timer, select Change part from the popup menu and select start implicit. Attach it to the top left corner of the topmost suspension region symbol and enter timer value "maxConnectTime".

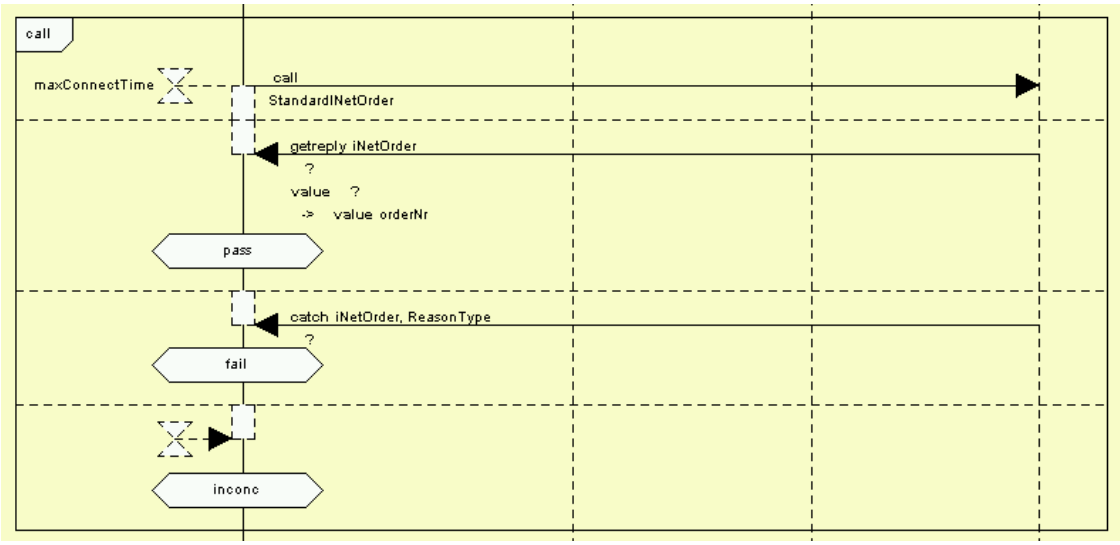
To receive a **getreply** message, create a message symbol , switch to Procedure-based communication, change the direction and attach it to the down right corner of the suspension region and the port instance "P2". Select the port operation **getreply** and enter the type "iNetOrder". Enter the template "?", the matching value "?" and the value assignment "ornerNr" in the property panel.

To create a verdict setting, select a condition symbol and change the verdict setting to **pass**, **fail**, **inconc** or **none**.

To create a **catch** message, choose a new message symbol , switch to Procedure-based communication, change the direction and attach it to the down right corner of the suspension region and the port instance "P2". Select the port operation **catch** and enter the type "iNetOrder, ReasonType" and the template "?".

To add the timeout timer, right-click on the implicit start timer, select Add new part from the popup menu and select timeout implicit. Left-click on the diagram, change the direction and attach it to the down left corner of the undermost suspension region.

Figure 6.84. Procedure-based Communication



The generation of the other symbols in this diagram is described before.

To create an **altstep** , add a new diagram, select altstep and enter the name "GuestDefault". Enter the component type "GuestType". Create a component instance, a port instance "P1" with the type "gPCOtype" and another port instance "CP" with the type "pCPtype". Create an **alt** inline expression . Enlarge the inline expression symbol and right-click on it to change the number of alternatives to "3".

Select a message , change the direction and attach it to the component instance and the port instance "P1". Enter the type "charstring" and the template "?". Now select again a message, attach it to both instances and enter the template "standardConversation". A repeat statement causes the re-evaluation of an alt statement. Right-click on the component instance axis short above the first dashed separator line, select Change Operation in Alternative from the popup menu and select repeat.

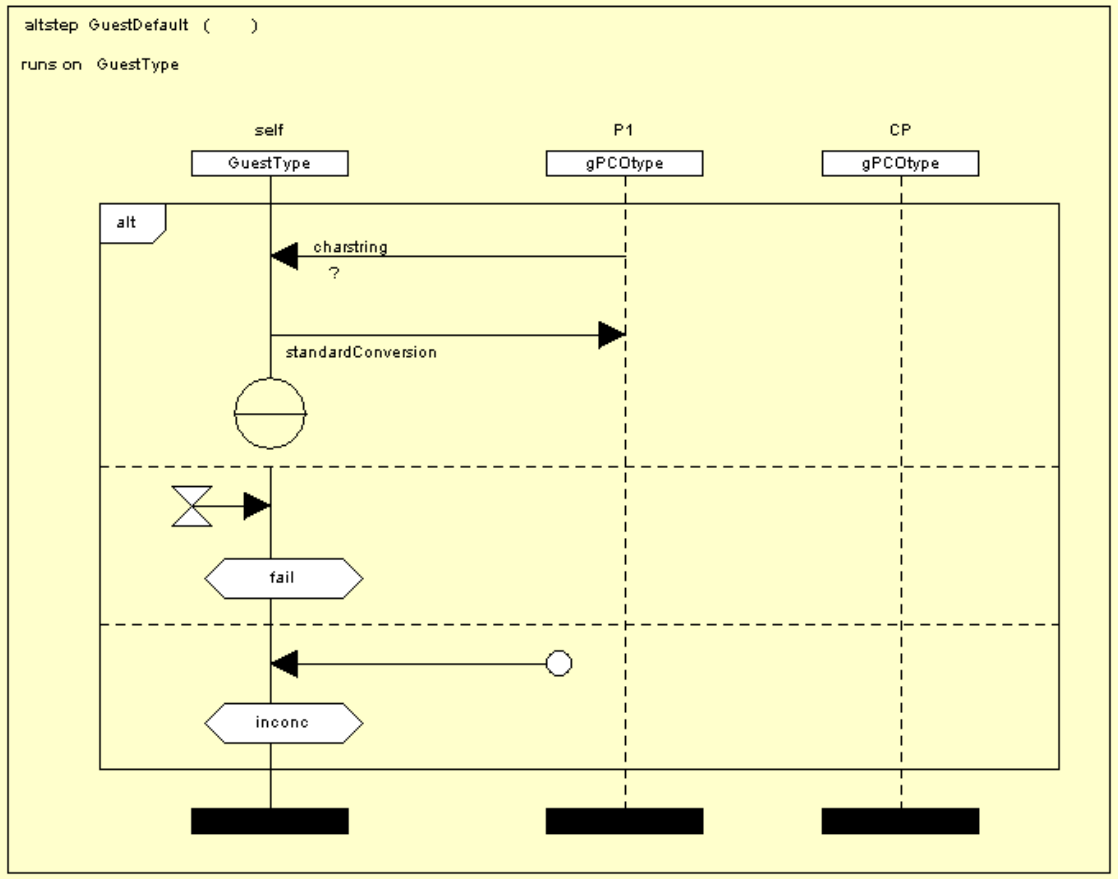
To create a timeout, select a timer part , change the direction and attach it to the component instance.

Note

If no timer name is declared, it will be checked for the timeout of any previously started timer. The generated TTCN-3 core language is "anytimer.timeout".

To create a verdict setting **fail** or **inconc**, select a new condition symbol and change the verdict setting by left-clicking in the verdict setting text field. To receive a message on any port, create a new message symbol, change the direction, right-click on the message, select Modify from the popup menu and select Found Message. Attach the message to the component instance.

Figure 6.85. Altstep



The **altstep** diagrams "StandardDefault" and "InternetDefault" can be edited in a similar manner.

Save TTCN-3 and GIF

To export the whole generated TTCN-3 core language, select Export TTCN-3 Core Language... from the File menu or just hit **Alt**+**T**.

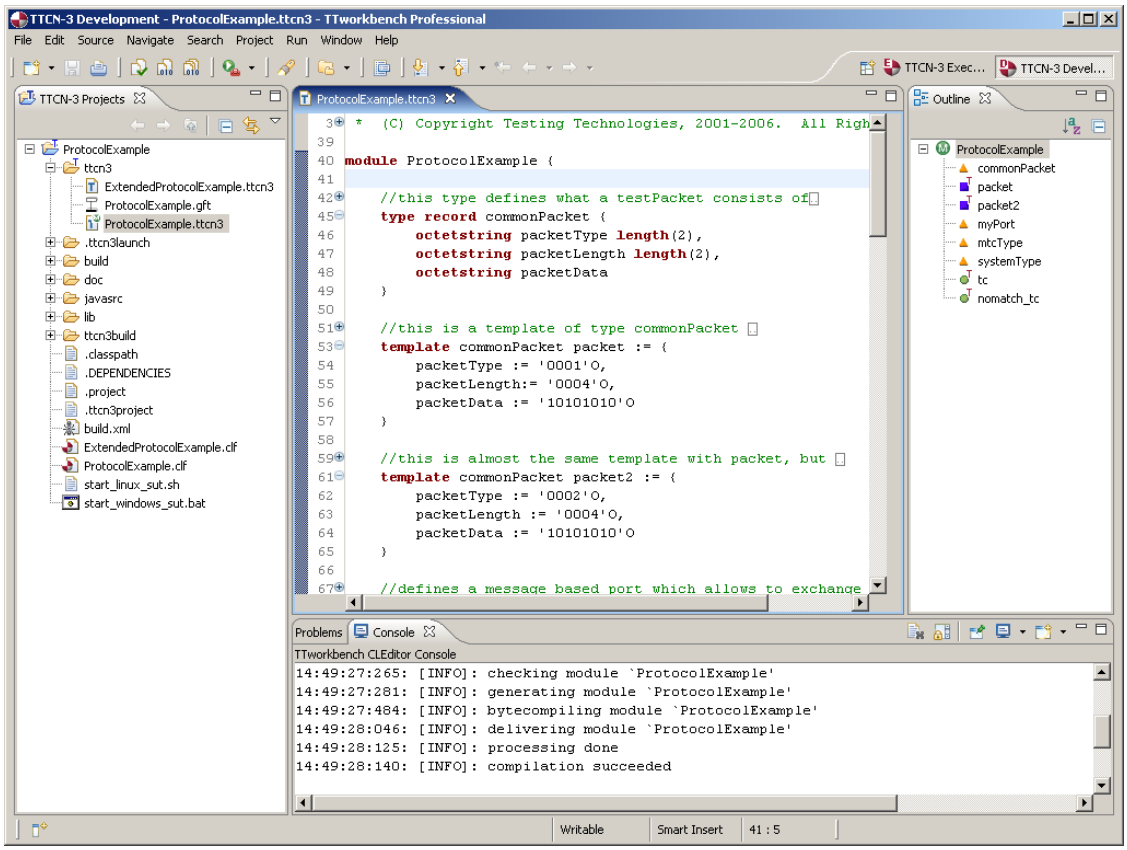
To export a diagram as a GIF file (e.g. for documentation purposes), activate the window with the desired diagram and select Export GIF... from the File menu or just hit **Ctrl**+**G**.

Chapter 7. Using TTworkbench TTthree

TTthree provides the generation of Java sources from test suite specifications based on the TTCN-3 Code Language, as well as the compilation of Java sources into byte code class files and their packaging into a single JAR archive file.

As shown in Figure 7.1, “TTworkbench TTthree”, TTthree is integrated in the editing environment of CL Editor. It makes use of the Problems view to report errors and warnings found during the compilation. The progress information is presented as messages of different verbosity levels to the Console view. Two actions, namely Compile and Rebuild, are added to the menus and tool bar of the editing environment. TTthree provides also a preference category for the parameterization of the compilation process, such as the verbosity level, the Java compiler, or the generation of a default campaign.

Figure 7.1. TTworkbench TTthree



Properties

TTthree supports all language features of TTCN-3. It is full TTCN-3 3.1.1 compliant, as defined by ETSI in ES 201 873-1, except of the features listed below. The TTthree runtime environment is compliant to the TTCN-3 Runtime Interface specification as defined by ETSI in ES 201 873-5 and TTCN-3 Control Interface specification as defined by ETSI in ES 201 873-6.

TTthree provides the following functions:

- Full support for TTCN-3 ETSI standard 3.1.1
- Execution from editing environment
- Redirection of error reports
- Preference page

Known limitations:

- **Template variables** Template variables are supported with restrictions. The possibilities of referencing templates or template fields are different from those explained in section 14.3.1. The assignment or referencing of parts of templates which are located inside of actual matching mechanisms will in most cases cause an error at runtime, even if explicitly allowed by the standard. But, it is possible to assign templates and values to (parts of) template variables, as long as the place of assignment is not contained inside a matching mechanism.
- **Permutation inside values** It is not allowed to use the permutation matching mechanism like AnyValueOrOmit (*) inside of values, as explained in section 14.3.0.
- **Meta characters in character patterns** The following character pattern matching mechanisms described in section B.5.1.0 are not implemented:
  - \N{reference} - the same can be achieved by using {reference} with the appropriate one-character template.
  - \s - the same can be achieved by using or referencing the character set containing the white space characters.
  - \b
  - + - the same can be achieved with #(1,)

  Set expression No meta-characters lose their special meaning inside character set expressions as explained in section B.5.1.1. Whenever special characters are to be used inside such a character set expression, they need to be quoted with a backslash.

  Set meta-characters outside set expression The character set meta-characters - and ^ are also considered meta-characters outside of character set expressions. Whenever these characters are to be used outside a character set expression, they still need to be quoted with a backslash.

  Pattern reference expression This is implemented differently from the behavior described in B.5.1.2. Whenever a charstring or charstring template is referenced via '{reference}', it is treated as a pattern only if it was defined using the keyword 'pattern' or as a character dictionary. Otherwise, the referenced string is treated literally, i.e. all meta-characters lose their special meaning.
- **The regexp function** Differently from the behavior described in C.17, the regexp function takes as its second argument a charstring template, not necessarily a charstring value.
  - If a charstring value is passed as second argument, all charstring pattern special characters are not interpreted as such, but as if they would appear escaped with a backslash in a charstring pattern.
  - If a charstring pattern template is passed, then the pattern is used for regexp-matching.
  - If another kind of charstring template is passed, then regexp returns the whole input string (the first argument to regexp) if the template matches and the group number 0 is passed as third argument to regexp, otherwise, the empty string is returned.

- **TLI log events** Following TLI log events are currently not supported: `tliPrCatchTimeout()`, `tliPrCatchTimeoutDetected()`, `tliTcExecute()`. These events will not be generated and therefore will not be present inside the log file.

Checked non-standard language extensions:

The following language extensions can be treated as an error or warning via the --strict-standard-compliance option. This option can be set directly on the command line when the command line version is used or on the general preference page for the GUI.

- **Identifiers starting with _** Identifiers are allowed to start with the underscore character _.

- **Inline Templates** The inline template construct is allowed to be used anywhere where a value expression is allowed. Since the compiler checks for every inline template, if its predicate part is definitely a value expression, this does not change the semantics, as T:V is seen to be equal to V.

- **Local Declarations after Statements** In statement blocks, it is allowed to introduce new declarations after non-declaration statements. It is not necessary to put all declarations at the beginning of the statement block.

- **Allow Templates as Type Constraint** In a subtype specification list, it is allowed also to use matching mechanisms to describe the subtype, instead of only constant expressions. This way, it is possible to describe infinite subtypes of structured types.

- **Escape Syntax for Special Characters in Strings** In charstring literals, it is possible to use the backslash escape syntax for the special control characters newline (\n), carriage return (\r), and tabulator (\t). It is also possible to use the sequence \" for escaping the quoting symbol. The special character \ must also be quoted by writing \\.

- **Multiple Free Text Lines** Free text can be split into several lines which are written as consecutive free text literals.

- **Non-Constant Expressions** In the declaration of const values and also in list, subset, superset, complement and length constraints, allow also non-constant value expressions (which means expressions that yield a value but which are not necessarily known at compile time), i.e. function calls or references to module parameters.

- **Fully Qualified Names in anytype** It is possible to use an anytype value with a fully qualified type name. This is useful for resolving nameclashes between types of the same name which are imported from different modules.

- **Concatenation of record of/set of Values** It is possible to use the concatenation operator & also for values of record of or set of type. The result is a new value which contains at the beginning the elements of the left operand which are succeeded by the elements of the right operand. The length of the resulting value is the sum of the lengths of the operand values.

- **Passing Timer References to PTCs** It is allowed to pass references of timers to a ptc in the start statement.

- **Passing inout Parameters to PTCs** It is allowed to pass variables to PTCS as inout or out parameters. The value of the out parameters should only be accessed by the starting component after the PTC has finished, as it is unspecified when the values will be updated.

- **External Behavior Functions** It is allowed to add a runs on clause to an external function. Such a function will be treated like a normal behavior function and can be started via a PTC.

- **External Functions with Template Result** It is allowed for external functions to also return results of template kind.

Other non-standard language extensions:

- **Implicit omit** For record or set values written with a list of field assignments, it is permitted to set fields declared as optional to 'omit' by omitting assignments to those fields.

- **Arbitrary field order of record values** If the fields of a record value are written with a list of field assignments, the assigned fields need not be in the same order as declared in the record type, but can be used in arbitrary order. The name of the field in each field assignment, not its place of occurrance, determines, which field of the record type is set.

- **Cyclic imports** As for Edition 2 of the Standard, it is still allowed for Edition 3 to use cyclic imports of modules, i.e. modules that import (parts of) each other (directly or indirectly).

- **Parameterized types** Parametric type declarations can also have formal type parameters. These are written simply with the parameter name (without a type in front) with an optional 'in' keyword in front.

- **Parameterized sub-types** When defining a sub-type of another type, it is also possible to parameterize it with a formal parameter list. However, the first formal parameter of such a formal parameter list must include the 'in' keyword (to syntactically distinguish the parameter list from a type constraint list).

- **Empty Formal Parameter List for Templates** Templates can also be parameterized via an empty formal parameter list. If such a template is later referenced, it must be instantiated also with an empty actual parameter list.

Preferences

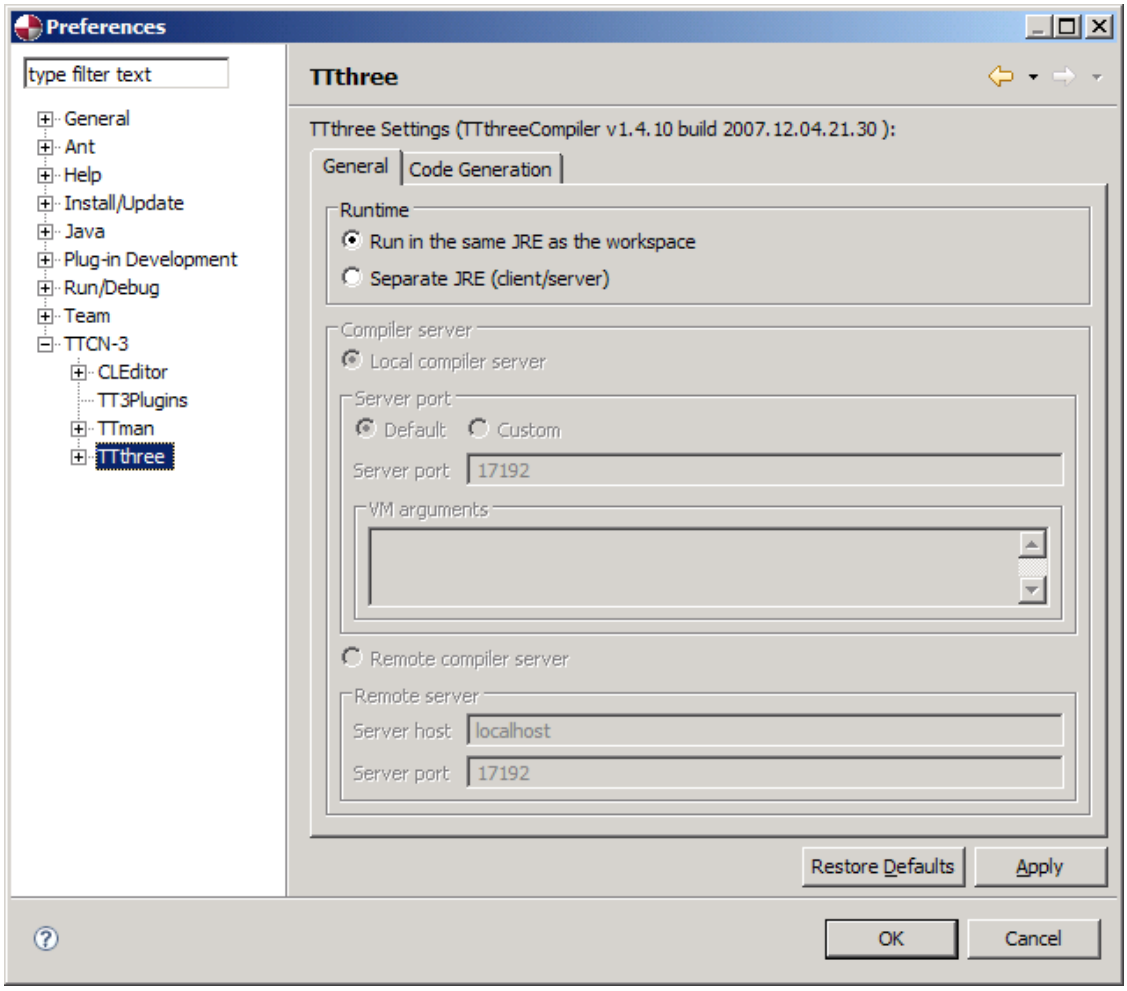
The TTthree preferences can be found under Window > Preferences > TTCN-3 > TTthree. Global preferences that also apply to TTthree are defined in preference pages directly included in the TTCN-3 preference category. Please refer to Chapter 3, *Global TTCN-3 Preferences* for further details. The TTthree preferences are structured into several preference pages. The compiler version information is printed at the top of every preference page.

Preference pages regarding options for logging, code generation, Java compiler, default campaign generation and TT3 plugins are introduced in the following. TTthree behavior can also be controlled on a per project basis via Project > Properties > TTCN-3 Settings. the section called “Compiler settings” explains which options are available.

General Settings

This preference page (Figure 7.2, “General preference page”) defines the running mode of the compiler. The compiler can be run in two ways, either in the same JRE as the workbench or in a separate virtual machine (client/server mode). Running the compiler in the same JRE is the default running mode.

Figure 7.2. General preference page



If the client/server mode is chosen (**Separate JRE (client/server)**), the compiler server can be configured. Either a local or a remote compiler server can be used. In case the **Local compiler server** is selected the server is started automatically when a compilation is initiated. The port the server will listen on can be configured choosing the **Custom** port. If the **Default** server port is selected, a predefined port will be used. The compiler server is started in a separate JVM using a default configuration. If the default configuration does not fit your requirements, additional parameters for the virtual machine can be added, e.g. for more heap space **-Xmx768m** can be used.

Choosing the **Remote compiler server** an already running compiler server will be used. The host where the remote server is running and the port the server is listening on can be defined editing the fields **Server host** and **Server port**.

Code Generation

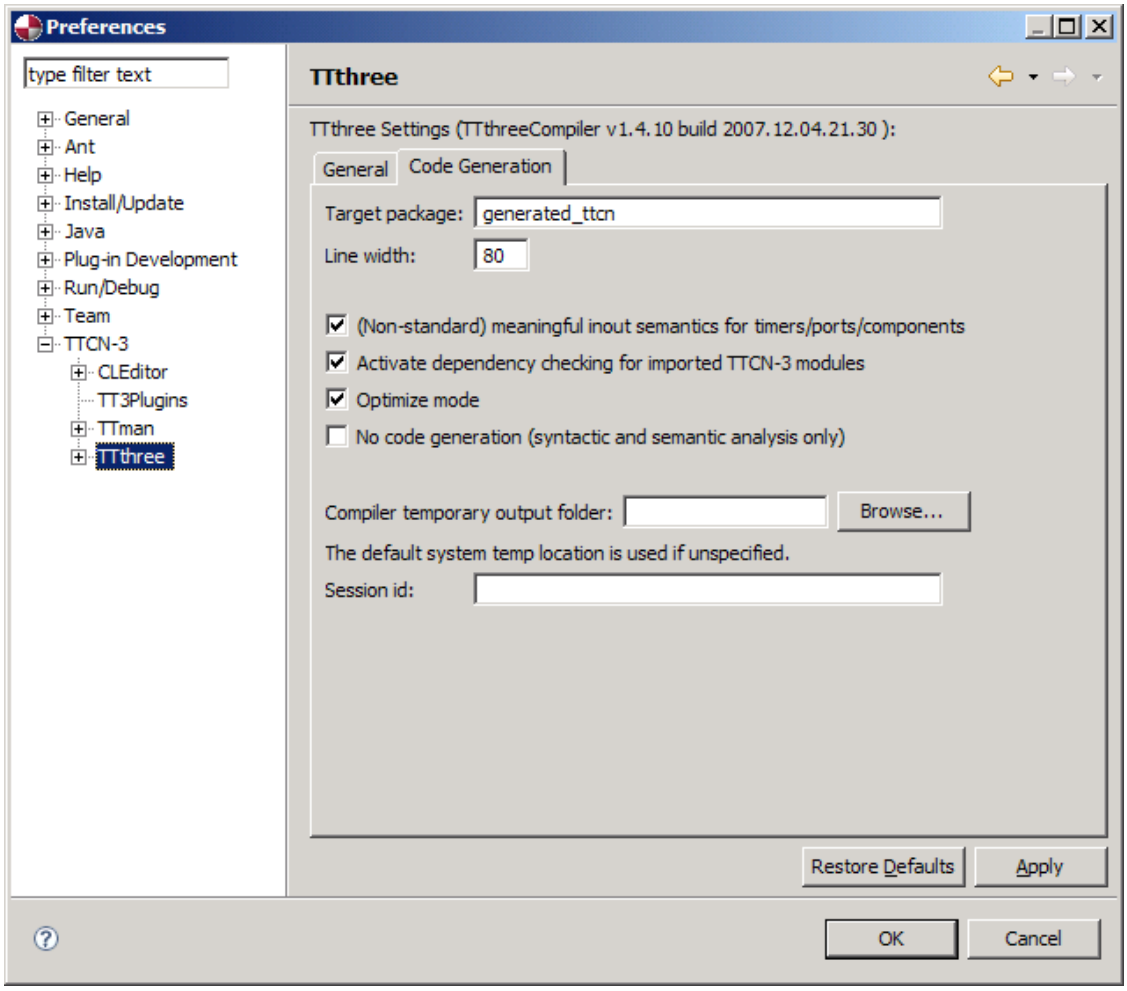
This page (Figure 7.3, “Code generation preference page”) is concerned with the generation of Java code in the given package. Default Java package is `generated_ttcn`.

In addition, line width of the generated Java code can be specified. The default line width is 80 columns.

This page provides also the following selections:

- **(Non-standard) meaningful inout semantics for timer, ports and components**: if selected (default), timers, ports and components that are passed as parameters the keywords "in" and "out" are also allowed besides "inout", as specified in standard. This option allows compilation of legacy test suites without modifying the sources.
- **Activate dependency checking for imported TTCN-3 modules**: if selected (default), only changed sources are compiled. To process all sources, the rebuild action should be used.
- **Optimize mode**: if selected (default), optimization of the abstract syntax tree is supported. Code optimization is not supported.
- **No code generation**: if selected, syntactic and semantic analysis is performed, without generation of Java source code.
- **Compiler temporary output folder**: the TTCN-3 compiler generates Java related temporary files during operation. It may be useful to direct generation of those files to a different folder or volume. A folder on a local file system is recommended.
- **Session Id**: this id must be used to distinguish temporary data folders from different sessions of the same user on a computer. By default the operating system's user name is used. The resulting temporary output folder name is constructed by appending the id to the original folder name.

Figure 7.3. Code generation preference page



Perform the Compilation

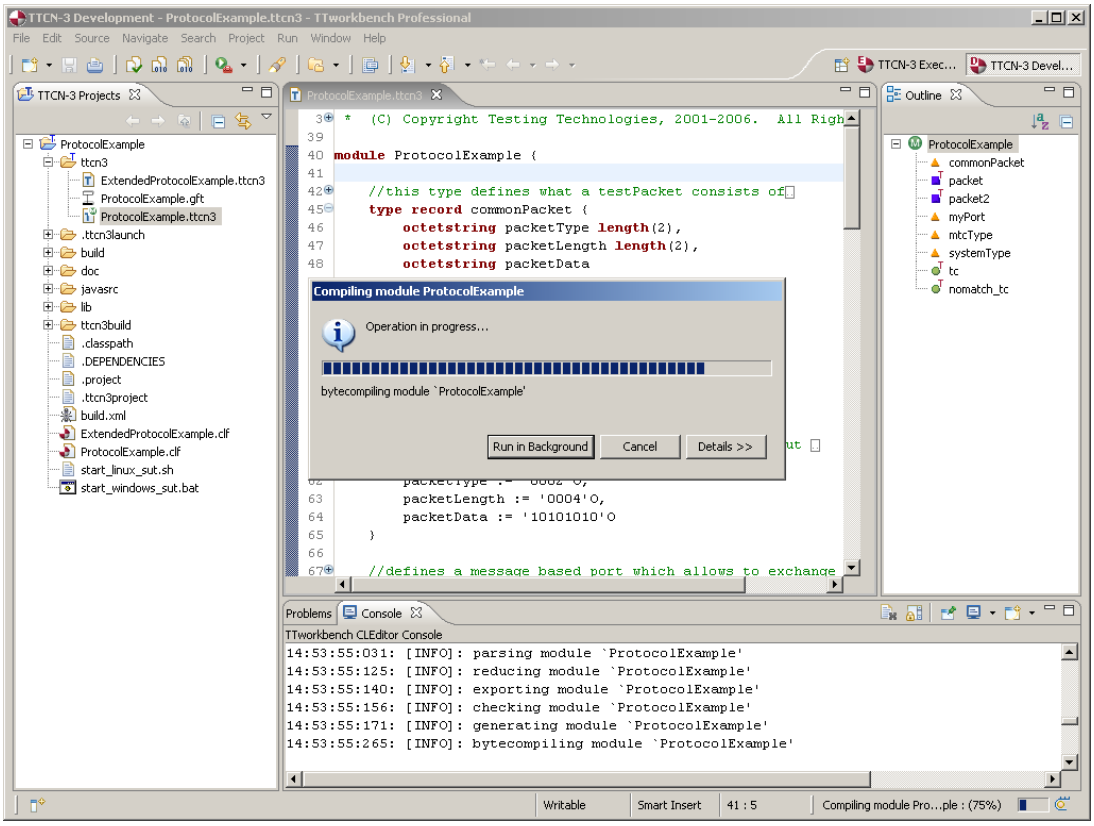
The compilation of TTCN-3 sources from the editing environment is performed in the following steps:

1. Open the file that contains the TTCN-3 module to compile.

2. Check related CL Editor and TTthree preferences, in particular project path, Java compiler and default campaign options.

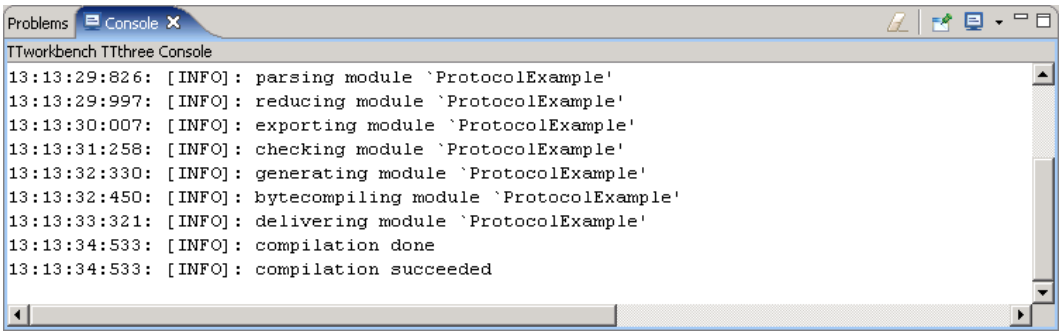
3. Press the **build** button. In case the Code Generation preference "Activate dependency checking for imported TTCN-3 module" is selected (default), **build** uses optimized make mode and processes only those sources that have been changed. To process all the sources, use the **rebuild**. command instead. Both actions are available over the menu bar **Run** and the context menu, as well as the short-keys **Ctrl+Shift+R** (rebuild) and **Ctrl+Shift+B** (build). The compilation progress is illustrated by a progress dialog and on the progress bar, as shown in Figure 7.4, “Compilation progress”.

Figure 7.4. Compilation progress



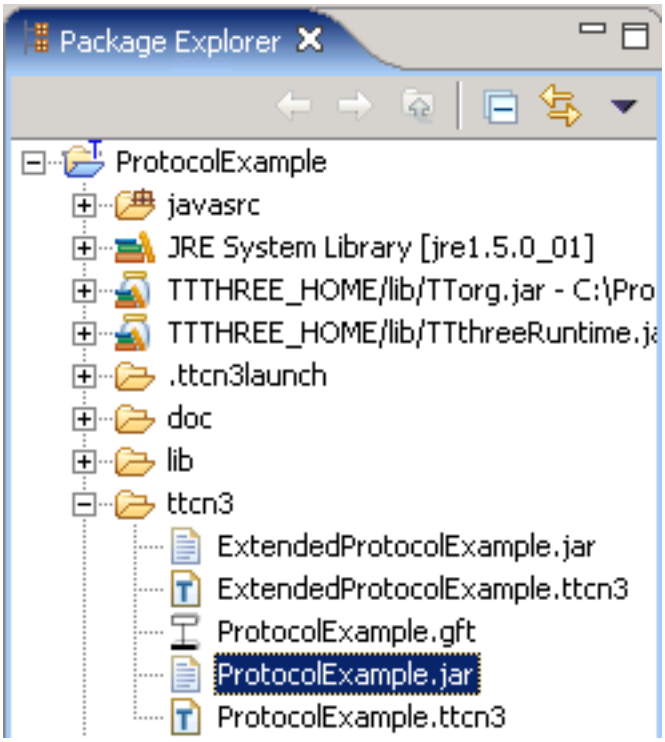
4. a. Observe the compilation output at the standard console according to the verbosity level specified in the logging preference page (the section called “Reporting”).

Figure 7.5. Compilation successful



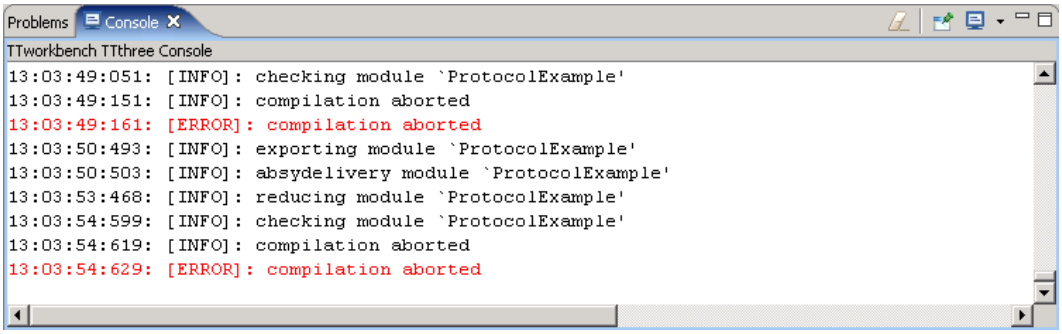
b. In case of successful compilation, the generated Java archive file will be placed by default in the same directory as the TTCN-3 source file, or in the **directory for compiled modules** as specified in the code generation preference page (see the section called “Code Generation”).

Figure 7.6. Generated jar file



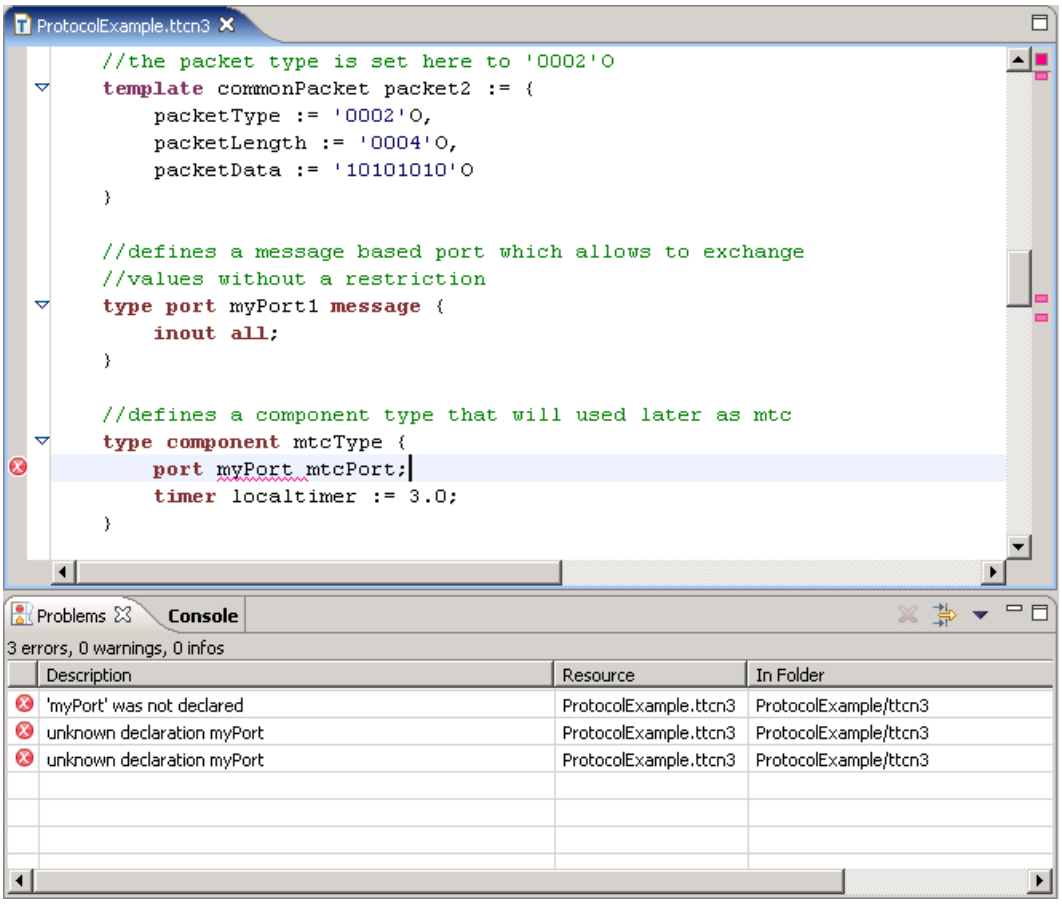
5. a. In case of errors in the TTCN-3 module, the errors will be logged during compilation process.

Figure 7.7. Compilation failed



b. Errors and warnings that are detected during compilation are listed in the Problems view. Clicking on one of the errors/warnings points to navigate to the erroneous location in the respective TTCN-3 source file.

Figure 7.8. Problems found



c. Repeat step 3 after correction.

Command-line Mode

Batch Compiler

TTthree can also be executed via command-line mode using the scripts located in TTworkbench installation directory:

- For Linux: `TTthree.sh`
- For Windows: `TTthree.bat`

Command-line synopsis:

`TTthree` [*`options`*] *`moduleId`* ...

`options`	Command-line options. TTthree Options may be in any order. For most options a short form (with one dash), and a long form (with double dash) exists.
`moduleId`	One or more modules to be compiled (such as MyModule).

The following TTthree options are available:

, *`--clean`*	Clean up jar-files of modules to be compiled to force their recompilation (see also --rebuild option).
, *`--clf-generate-default`*	generate default campaign during compilation
, *`--clf-name <default-campaign-name>`*	name of the default campaign
, *`--clf-testadapter <testadapter-name> <testadapter-file>`*	name and filename of the testadapter of the default campaign
, *`--clf-testadapter-file <testadapter-file>`*	filename of the testadapter of the default campaign
, *`--clf-testadapter-name <testadapter-name>`*	name of the testadapter of the default campaign
, *`--clf-only-visible-testcases`*	generate only the testcases that are visible in the root module
`-k`, *`--continue`*	Force the continuation of the compilation process even after a module could not be delivered.
`-D`, *`--debug`*	

Turn debugging mode on. The debugging option should only be used in case of unclear compiler output, or any other strange behavior.

-d, *--destination-path <dir>*

Specify the path where to place the compiled TTCN-3 modules (`*.jar` file). If **-d** or **--destination-path** is not specified, TTthree places the compiled TTCN-3 modules at the same location where the respective TTCN-3 source file (`*.ttcn3` file) resides.

Note

This directory should also be added to the project path, if the modules import each other (see --project-path).

Note

TTCN-3 import statements will be mapped into manifest class-path entries in the respective compiled TTCN-3 module (`*.jar` file). The class-paths will be relative, if the compiled output will be placed in the same directory, i.e. either the TTCN-3 sources where already in the same directory, or the **-d** option is used, or both. The class-path will be absolute, if the imported module comes from another directory and no **-d** option is used, i.e. the compiled TTCN-3 modules (`*.jar` files) reside in different directories. Please be aware of this fact, when moving compiled TTCN-3 modules for execution.

, *--do-not-check-send-templates*

If set the check whether send-templates do not contain matching mechanisms is disabled.

-0, *--dry-run*

Perform the syntactic and semantic analysis according to the TTCN-3 standard, but generate no Java source code.

, *--export-metamodel*

Export the TTCN-3 meta model as XMI.

-g, *--gen-debug <info>*

Generate code for debugging of <info>. <info> can be 'record-initialization' or 'none'. If 'record-initialization' is set, generated java classes will contain code which can be used to monitor initialization of TTCN-3 record structures. By default no such code will be generated. If 'none' is used, no trace and debug information will be generated, so that no TTCN-3 debugging or tracing through the log will be possible.

-h, *--help*

Get help information on command line options and exit.

-C, *--javac-command <command>*

Use the Java compile command **<command>** for the compilation of the generated Java source code into Java class files. If **-C** or **--**

javac-command is not specified, TTthree uses **jikes +E** for the Java compilation.

Note

TTthree assumes that the Java compiler is installed on your system.

-j, *--keep-java*

Keep the generated Java source code in the delivered JAR archive file. If **-j** or **--keep-java** is not specified, the JAR archive file does only contain the compiled Java class files.

Note

Due to a bug in the Java compiler **javac** this option cannot be selected when **javac** is used

, *--license-file <file>*

The license file to be used.

-l, *--line-width <number-of-cols>*

Specify the line width of the generated Java code. If **-l** or **--line-width** is not specified, TTthree uses a line width of 78 columns to generate Java code.

, *--log*

Show log messages independently of verbosity level.

-x, *--map-suffix <language> <suffix>*

Specify the filename suffix for modules of one of the following languages: **TTCN-3** and **ASN.1:1997**. If **-x** or **--map-suffix** is not specified, TTthree uses `.ttcn3` for TTCN-3 modules and `.asn1` for ASN.1 modules.

Note

Non-TTCN-3 language modules require additional TTthree plugins.

, *--named-logs*

Prefix all log messages with the session name.

, *--nolock*

Set OSGI file locking mode to **none** to avoid certain errors with installations on read-only file systems. When this option is activated the parameter **-Dosgi.locking=none** is passed to the Eclipse base system.

Note

On Unix platforms only, will be ignored on Windows platforms.

-i, *--normal-inout*

Switch the (nonstandard) meaningful inout semantics for timers/ports/components on. If activated, timers, ports and components that are passed as parameters the keywords "in" and "out" are also

allowed besides "inout", as specified in standard. This option allows compilation of legacy test suites without modifying the sources.

-O, --optimize

Switch the optimize mode on. Currently only optimization of the abstract syntax tree, no code optimization is supported.

-P, --project-path <dir>

Specify the TTCN-3 core language path to search for module definitions. Path entries are separated by semicolons (**;**) on Windows and by colons (**:**) on Linux and have to be directories. If **-P** or **--project-path** is not specified, TTthree uses the current directory (**.**) as default project path. This option can be used multiple times

-r, --rebuild

Recompile and deliver all imported modules, even if up to date. To achieve a full recompilation (the same as the Rebuild button does), this option has to be used together with **--clean** (see also --clean option)

-R, --runtime-class-path <path>

Specify the TTCN-3 runtime class path to search for the TTCN-3 runtime environment. It can be found in the JAR file `TTthreeRuntime.jar`. It is located in the `lib` directory of the installation directory. If **-R** or **--runtime-class-path** is not specified, TTthree assumes `TTthreeRuntime.jar` to be in the TTthree Java Archive directory.

Windows platforms

If you did not change the destination folder during the installation process of TTworkbench this is

```
[ProgramFiles]\[TTworkbench]\plugins
\com.testingtech.ttworkbench.ttthree.core_x.x.x
\lib.
```

Unix platforms

This is

```
[InstallDir]/[TTwork-
bench]/plug-
ins/com.testingtech.ttworkbench.ttthree.core_x.x
lib.
```

-S, --session-id <id>

Specify your own session id for the compilation process. The session id is used by TTthree to identify data that is stored temporarily in the temp directory of your system. If **-S** or **--session-id** is not specified, TTthree uses a combination of your user name and a timestamp.

-s, --source <language>

Set the default language of source files if not explicitly specified in the source file. If **-s** or **--source** is not specified, the default language will be TTCN-3:2008.

Note

If the language attribute is set within the TTCN-3 module itself, then this language takes precedence over the language from the command line.

, *--t3doc-outdir <output-dir>* — output directory for the generated T3Doc HTML documentation

-n, *--target-package <package-name>*

Generate the Java code in the package given by < **package-name**>. If **-n** or **--target-package** is not specified, the generated Java classes will be located in the Java package `generated_ttcn` .

Note

If **-n** or **-target-package** is specified, the package attribute of the module loader file has been set to the same value.

, *--tmp-copy-jars* — Copy reloaded JAR files to `tmp` working dir.

, *--tmp-dir <tmp-dir>* — The working directory where internal compiler output is stored (normally, the `tmp` directory).

, *--use-bigint* — Use arbitrarily large integer values

Note

Activating this option allows using of integer values outside of the domain -2^31..2^31-1; the use of these values is restricted only to TTCN-3 source. They cannot be used as module parameters, in external functions or codec, as such values cannot be handled by the standardized TCI interfaces. Activation of this option may lead to loss of performance.

, *--strict-standard-compliance<msg-kind>* — Check for standard compliance of input and generate appropriate Warnings or Errors. Valid message kinds are **suggest** (default), **enforce** and **relax**.

, *--implicit-import <modules>* — Import the given modules implicitly into all modules which do not import them explicitly. This option can be present more than once. If more than one module name is given, separate them with ';' or ':', according to the used operating system. The given modules must either be built-in modules, TTCN-3 source modules or compiled modules present in the project path.

, *--validate-asn1* — validate X.682 ASN.1 constraints

-v, --verbosity <level>

Define the verbosity level for TTthree. Depending on the level TTthree will produce output during the compilation process. The following verbosity levels are known by TTthree: **debug** , **log**, **hint**, **warning** , **error**, and **failure** . If **-v** or **--verbosity** is not specified, TTthree uses **hint** as its default verbosity level.

-V, --version

Get version information of the currently installed TTthree version and exit.

Exit codes:

0	Success
1	Error
2	Abort
3	Invalid usage

Advanced Batch-Compiler for Linux

For Linux systems an optimized version of the TTCN-3 compiler script has been created which starts up faster than the regular one: **TTthree2.sh**. This added benefit comes with the requirement of setting certain environment variables beforehand.

Usage of TTthree2.sh

1. To prepare the environment call **source /path/to/TTworkbench/ttthreeenv.sh** (when using the **bash** or **sh** command interpreter). In a **tcsh**-like shell use **source /path/to/TTworkbench/ttthreeenv.csh** respectively.

2. These environment variables will now be available:

   - TTTHREE_HOME
   - TTWB_CORE
   - ECORE_HOME
   - COMMON_HOME
   - TTTHREE_PLUGINS_PATH

3. Within this environment, **TTthree2.sh** can be used like the regular **TTthree.sh** command. See the section called “Batch Compiler” for parameters and options.

PluginHomeResolver.sh script

The **PluginHomeResolver.sh** utility script in TTworkbench installation directory is internally used to set the environment variables needed by **TTthree2.sh**. The documentation is included here for cases where

the procedure described above is not sufficient. Depending on the option given it prints out the installation path to certain TTworkbench (Eclipse) plugins. Such a path is then used to find needed libraries.

Recommended PluginHomeResolver.sh options:

, *--env-for-csh*	determines all paths needed for **TTthree2.sh** and prints a command line for **tcsh** users.
, *--env-for-sh*	determines all paths needed for **TTthree2.sh** and prints a command line for **sh** or **bash** users.

Use the generated output and execute it as a command in your shell to set the needed variables. The following options print the same information but just one path at a time:

, *--common-home*	prints path to plugin "org.eclipse.emf.common". Set variable COMMON_HOME to this value.
, *--ecore-home*	prints path to plugin "org.eclipse.emf.ecore". Set variable ECORE_HOME to this value.
, *--ttthree-home*	prints path to plugin "com.testingtech.ttworkbench.ttthree.core". Set variable TTTHREE_HOME to this value.
, *--ttwb-core*	prints path to plugin "com.testingtech.ttworkbench.core". Set variable TTWB_CORE to this value.
, *--tt3-plugins-path*	prints a list of paths where TTthree-plugins can be found. Set variable TTTHREE_PLUGINS_PATH to this value.
-h, *--help*	get help information on command line options and exit

TTthree Server

Starting

The TTthree server can be started using the shell script **TTthreeServer.sh**. Currently the server can be started only on Linux machines.

The usage of the script **TTthreeServer.sh** is similar to **TTthree2.sh**:

To prepare the environment call source **/path/to/TTworkbench/ttthreeenv.sh** (when using the bash or sh command interpreter). In a tcsh-like shell use source **/path/to/TTworkbench/ttthreeenv.csh** respectively.

These environment variables will now be available:

TTTHREE_HOME

TTWB_CORE

ECORE_HOME

COMMON_HOME

TTTHREE_PLUGINS_PATH

Within this environment, **TTthreeServer.sh** can be used.

Command line options

The list of all available command line options can be obtained calling **TTthreeServer.sh --help**

There is only one option that is specific to the server and not common with the stand-alone compiler:

-p, --port <port-number> the port number to connect to

TTCN-3 Documentation Generation (T3Doc)

TTCN-3 modules can include documentation in their source code, in special documentation comments. Such comments can appear before each module, group, test case, function, or altstep declaration and before each type, template, modulpar, or constant declaration. Documentation comments inside declarations are ignored and therefore do not contribute to any generated output. A documentation comment is a the text enclosed by the ASCII characters `/**` and `*/` that can be processed by the documentation generator to prepare automatically generated HTML documentation or to present it in hovers.

Generate HTML Documentation

Start generation in TTworkbench

In order to generate an HTML documentation from TTCN-3 open the Export dialog. This can by done by right click in the TTCN-3 Project view and choosing the context menu item "Export" or by clicking on the menu item "Export" in the file menu. Within the Export dialog open the folder "TTCN-3", select the item T3Doc and click on the Next button. Choose in the next dialog your TTCN-3 file you want to generate an HTML documentation from. If no file is chosen, the main module will be taken. Enter the destination path where the HTML files shall be saved and click on the Finish button to generate the documentation files.

Start generation from command line

The HTML documentation can also be generated via command-line mode by using the scripts located in the TTworkbench installation directory:

- For Linux: `T3Doc.sh`
- For Windows: `T3Doc.bat`

Command-line synopsis:

`T3Doc` [*`options`*] *`moduleId`*

The following T3Doc options are available:

`options`	Command-line options. T3Doc options may be in any order.
`moduleId`	The module for which the documentation shall be generated.
`-D`	Debug (see TTthree debug option for details)
`--nolock`	OSGI file locking mode (see TTthree OSGI file locking option for details)

--t3doc-outdir <output-dir>	Sets the output directory for the generated T3Doc HTML documentation to <output-dir>

The Text of a Documentation Comment

Content

The text of a documentation comment consists of the characters between the `/**` that begins the comment and the `*/` that ends it. The text is divided into one or more lines. On each of these lines, leading `*` characters are ignored; for lines other than the first, blanks and tabs preceding the initial `*` characters are also discarded. So, for example, in the comment:

```
/** @desc XYZ
 ** Initialize to pre-trial defaults.
    123 */
```

the text of the comment has three lines. The first line consists of the text " `@desc XYZ` "; the second line consists of the text " `Initialize to pre-trial defaults.` " and the third line consists of the text " `123` ". Subsequent documentation comments are combined to one logical documentation comment that is related to the respective declaration. Optional non-documentation comments do not contribute to a standalone documentation or combined logical documentation comments.

Formatting

Text in a documentation comment may use HTML-like markers for formatting. The following markers are supported: paragraph `<P>` , forced line break `<BR>` , italic text style `<I>` , emphasis `<EM>` , computer code fragment `<CODE>` , pre-formatted text `<PRE>` , unordered list `<UL>` , ordered list `<OL>` , list item `<LI>` .

General Description

The first sentence of each documentation comment should be a summary sentence, containing a concise but complete description of the declared entity. This sentence ends at the first period that is followed by a blank, tab, or line terminator, or at the first tagline. Alternatively the `@desc` tag can be used for general description.

Tagged Paragraphs

A line of a documentation comment that begins with the character `@` followed by one of a few special keywords starts a tagged paragraph. The tagged paragraph also includes any following lines up to, but not including, either the first line of the next tagged paragraph or the end of the documentation comment. Tagged paragraphs identify certain information that has a routine structure, such as the description of a function, in a form that the documentation comment processor can easily marshal into standard typographical formats for purposes of presentation and cross-reference. Different kinds of tagged paragraphs are available for module, group, test case, function, and altstep declarations and for type, template, modulpar, and constant declarations. Unless otherwise noted every kind of tagged paragraph can be used for every type of declaration. Each tagged paragraph defines its own multiplicity. If more than one tag of the same kind is defined where only a single occurrence should be defined the first one will be taken for subsequent processing. The order of tagged paragraphs will be retained unchanged.

The following table describes where the tags can be used.

Table 7.1. T3Doc Tags

	Data Types	Component Types	Port Types	Modulepars	Constants	Templates	Signatures	Functions	Altsteps	Test Cases	Modules	Groups	Control Part
@author	X	X	X	X	X	X	X	X	X	X	X	X	X
@deprecated	X	X	X	X	X	X	X	X	X	X	X	X	X
@desc	X	X	X	X	X	X	X	X	X	X	X	X	X
@exception							X						
@img	X	X	X	X	X	X	X	X	X	X	X	X	X
@member	X	X	X										
@param	X	X	X			X	X	X	X	X			
@remark	X	X	X	X	X	X	X	X	X	X	X	X	X
@return							X	X					
@see	X	X	X	X	X	X	X	X	X	X	X	X	X
@shortdesc	X	X	X	X	X	X	X	X	X	X	X	X	X
@since	X	X	X	X	X	X	X	X	X	X	X	X	X
@url	X	X	X	X	X	X	X	X	X	X	X	X	X
@verdict								X	X	X			
@version	X	X	X	X	X	X	X	X	X	X	X	X	X

@author

This tag should be used to specify the names of the authors or an authoring organization which either has created or is maintaining a particular piece of TTCN-3 code. The following are examples of `@author` taglines:

```
@author Mary Wollstonecraft
@author Hildegard von Bingen
@author Dorothy Parker
```

The information in an `@author` paragraph has no special internal structure. A documentation comment may contain more than one `@author` tag. Alternatively, a single `@author` paragraph may mention several authors:

```
@author Jack Kent, Peggy Parish, Crockett
 Johnson,
        A.A. Milne, Marjorie Weinman
 Sharmat,
        Mary Shelley, and Madeleine L'Engle
```

Note

It is recommended to specify one author per `@author` paragraph, which allows the documentation processing tool to provide the correct punctuation in all circumstances.

@deprecated

It should be used to describe if a particular piece of TTCN-3 code is deprecated. The first sentence of deprecated-text should at least tell the user when the code was deprecated and what to use as a replacement. Subsequent sentences can also explain why it has been deprecated. The following are examples of `@deprecated` taglines:

```
@deprecated As of version 1.2, replaced by
 ExtensionHeaderList
```

The information in a `@deprecated` paragraph has no special internal structure. A documentation comment may contain at most one `@deprecated` tag.

@desc

It should be used to describe the purpose of a particular piece of TTCN-3 code. It should contain a concise but complete description of the declared entity. The following are examples of `@desc` taglines:

```
@desc This type defines what a test packet
 consists of.
@desc SUT port number.
@desc Maximum duration a message will remain
 in the network.
```

The information in a `@desc` paragraph has no special internal structure. A documentation comment may contain more than one `@desc` tag.

`@exception`

This tag should only be used with signatures. It is used to provide additional information on the exceptions thrown by the given function. The following are examples of `@exception` paragraphs, which may be used in documentation comments for declarations of signatures:

```
@exception IndexOutOfBoundsException the
 matrix is too large
@exception FileNotFoundException the file
 does not exist
```

The information in an `@exception` paragraph should consist of the name of an exception followed by a short description of the circumstances that cause the exception to be thrown. A documentation comment may contain more than one `@exception` tag.

`@img`

This tag may be used to associate images with a particular piece of TTCN-3 code. The following are examples of `@img` taglines:

```
@img /ttcn3/doc/images/small.gif
@img http://portal.etsi.org/ptcc/images/
ptcc.gif
```

The information in an `@img` paragraph will be used to link to a image. A documentation comment may contain more than one `@img` tag.

`@member`

This tag is used to document the members of records, sets, unions, ports and component types. The following are examples of `@member` taglines:

```
@member ExtensionHeaderList List of
 extension headers defined by RFC 2460.
@member Ipv6Port definition required for
 LibIpv6COmp type compatibility.
```

The information in a `@member` paragraph should consist of the name of the member followed by a short description. For nested definitions the dot notation should be used. A documentation com-

ment may contain more than one `@member` tag. The usual convention is that if any `@member` paragraphs are present in a documentation comment, then there should be one `@member` paragraph for each member of the respective TTCN-3 definition and the `@member` paragraphs should appear in the order in which the members are declared.

Note

Subsequent `@member` paragraphs for the same member will be ignored.

Note

`@member` tags that are not applicable will be ignored.

`@param`

This tag is used to document the parameters of parameterized TTCN-3 definitions. The following are examples of `@param` taglines:

```
@param loc_User The name of user in the
 specified realm.
@param loc_password A known shared secret,
 the password of user of the specified
 username.
```

The information in a `@param` paragraph should consist of the name of the parameter followed by a short description. A documentation comment may contain more than one `@param` tag. The usual convention is that if any `@param` paragraphs are present in a documentation comment, then there should be one `@param` paragraph for each parameter of the respective TTCN-3 definition and the `@param` paragraphs should appear in the order in which the parameters are declared.

Note

Subsequent `@param` paragraphs for the same parameter will be ignored.

`@remark`

This tag may be used to add additional information, such as highlighting a particular feature or aspect not covered in the description. The following are examples of `@remark` taglines:

```
@remark Authorization was not requested as
 expected.
@remark This function should _not_ be called
 if the MTC acts as a client.
```

The information in a `@remark` paragraph has no special internal structure. A documentation comment may contain more than one `@remark` tag.

`@return`

This tag should only be used with signatures and functions. It is used to provide additional information on the value returned by the given function. The following are examples of `@return` paragraphs, which may be used in documentation comments for declarations of functions whose result type is not void:

```
@return The number of bytes received.
@return True in case of success, false
 otherwise.
```

The information in a `@return` paragraph has no special internal structure. The usual convention is that it consists of a short description of the returned value. A documentation comment may contain at most one `@return` tag.

`@see`

This tag may be used to refer to other globally visible TTCN-3 definitions in the same or another module. The following are examples of `@see` paragraphs, which may be used in any documentation comment to indicate a cross-reference.

```
@see SIP_TypesAndConf.PX_T2
@see calculateCredentials
```

The information in a `@see` paragraph may be used to link to the referenced TTCN-3 documentation. A documentation comment may contain more than one `@see` tag.

`@shortdesc`

It should be used to give a short description the purpose of a particular piece of TTCN-3 code that can be used in generated overview documents. The following are examples of `@shortdesc` taglines:

```
@shortdesc Registration group
@shortdesc Tests generation of IPv6 packets.
```

The information in a `@shortdesc` paragraph has no special internal structure. A documentation comment may contain at most one `@shortdesc` tag.

@since

This tag indicates the version of the module that a particular piece of TTCN-3 code was added to that module. The following is an example of a `@since` tagline:

```
@since 493.0.1beta
```

The information in a `@since` paragraph has no special internal structure. A documentation comment may contain at most one `@since` tag.

@url

This tag should be used to associate references to external files or web pages with a particular piece of TTCN-3 code, e.g. a protocol specification or standard. The following are examples of `@url` taglines:

```
@url http://www.ietf.org/rfc/rfc3261.txt?
number=3261
@url file:///D:/docs/DTS-TIPHON-06021-2.pdf
```

The information in an `@url` paragraph may be used to link to the referenced file or web page. A documentation comment may contain more than one `@url` tag.

@verdict

This tag should only be used with test cases, functions, and altsteps. It is used to provide additional information on the verdict assigned by the given test case, function, or altstep. The following is an example of a `@verdict` paragraph:

```
@verdict fail MAC Address for test cleanup
 could not be configured.
@verdict pass Registration successful.
```

The information in a `@verdict` paragraph should consist of one of the verdict values pass, fail, or inconc followed by a short description. A documentation comment may contain more than one `@verdict` tag. The usual convention is that it consists of a short description of the reasons for verdict assignment.

@version

This tag is used to state the version of a particular piece of TTCN-3 code. The following is an example of a `@version` tagline:

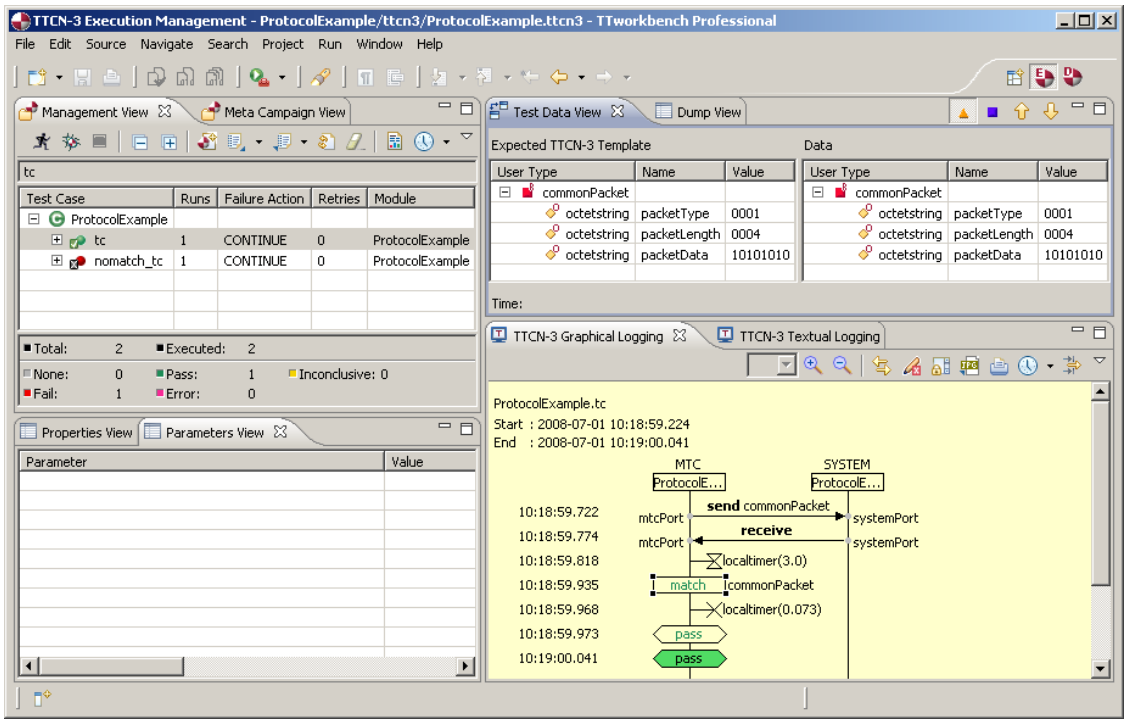
```
@version 493.0.1beta
```

The information in a `@version` paragraph has no special internal structure. A documentation comment may contain at most one `@version` tag.

Chapter 8. Using TTworkbench TTman

Overview

Figure 8.1. Overview of TTman



As depicted on figure Figure 8.1, “Overview of TTman” above, TTman consists of 7 views, which build the TTCN-3 Execution Management perspective. Each of those views deals with a specific aspect of the test execution:

- The management view is the central view of TTman. It provides an interface for the user to select a test suite and to start and stop the execution. All subsequent operations in TTman are relative to the selected test suite.
- The properties view displays the properties of a selected element in the test management view. For example, if a test case is selected in the test management view, the properties view will display its name, its description, its verdict as well as any other of its properties.
- The parameters view allows you to view and edit the module parameters.
- The TTCN-3 graphical logging view displays the traces from the test execution process in graphical form.
- The TTCN-3 textual logging view displays the same traces as the graphical logging view, but in textual form.

- The test data view is used to display the data transmitted or received during test execution.
- The dump view is used to display sent or received data as hex dump or as plain text.

There are three possibilities to open the TTman perspective:

- Select **Window > Open Perspective > TTCN-3 Execution Management** as depicted on figure Figure 8.2, “Open the TTCN-3 execution management perspective”.
- Double-click on an `*.clf` file.
- Right-click on an `*.clf` file, select **Open With** and then the TTworkbench TTman Figure 8.3, “Open the TTCN-3 execution management perspective with right click”.

Figure 8.2. Open the TTCN-3 execution management perspective

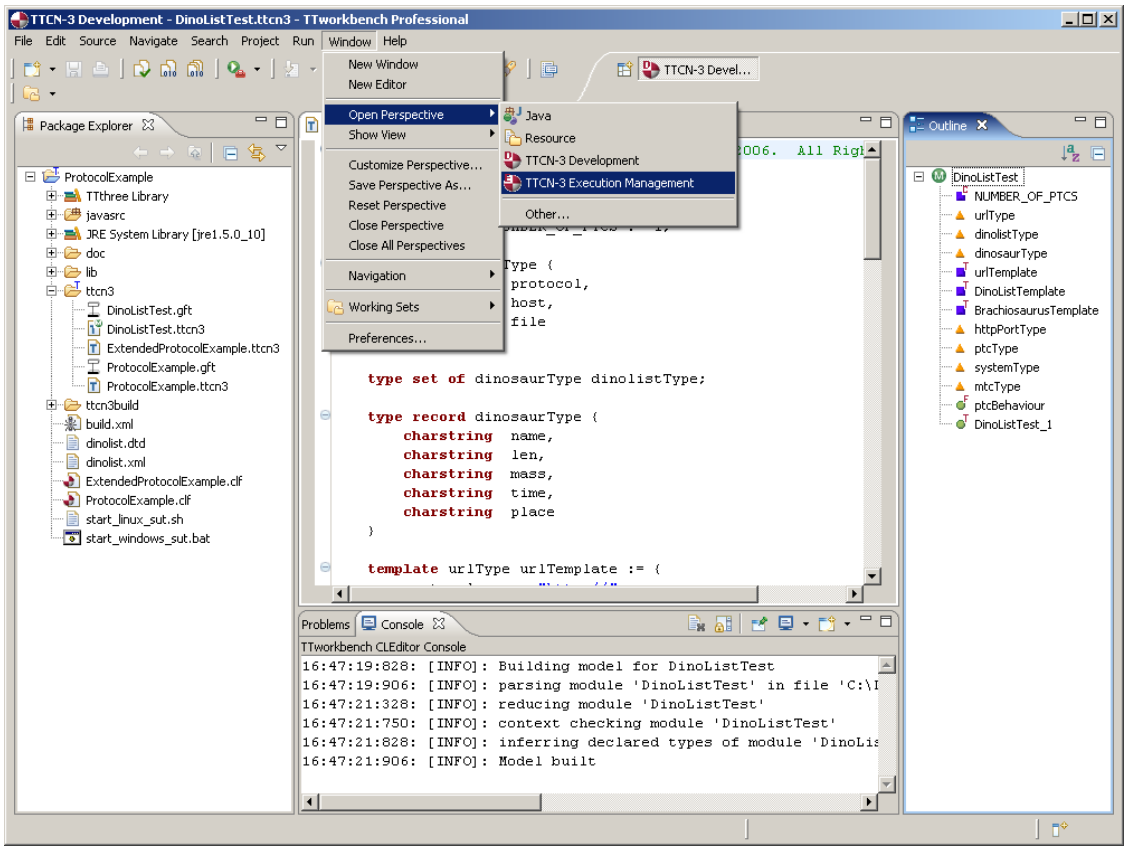
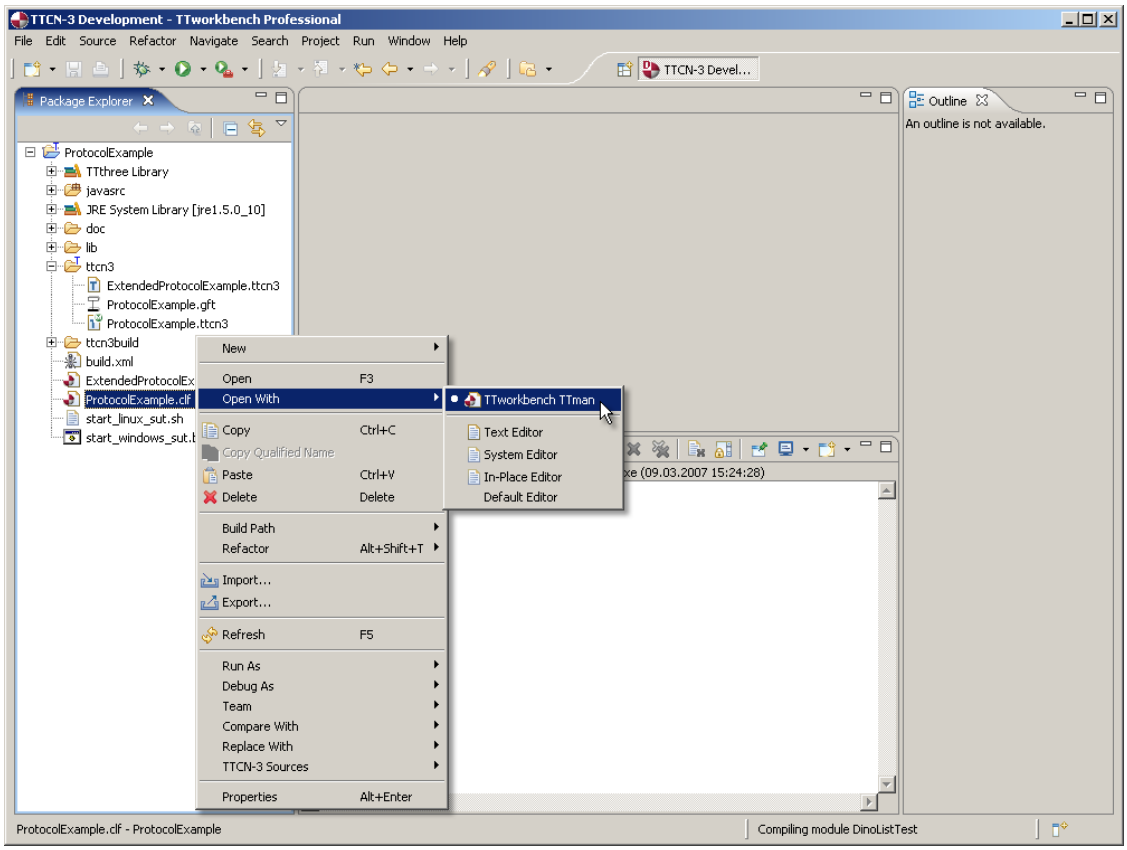


Figure 8.3. Open the TTCN-3 execution management perspective with right click



Using TTman

TTman provides a means for configuring, managing and executing a TTCN-3 test suite. Prior to loading a test configuration, the managing activities have to be done on the Management View. After loading but prior to executing the configuration the Parameters View allows you to edit the module parameters. During the test execution process, the Properties View, the test console view and the Dump View display status information and data as well as property values for selected elements, while the Graphical Logging View and the Textual Logging View show detailed information about the current test run.

Test Campaign

Test configurations get stored in test campaigns. A test campaign represents a collection of test cases, parameter settings and information about the test adapter to use. It contains neither test results nor test logs (see also the section called “Test Session”).

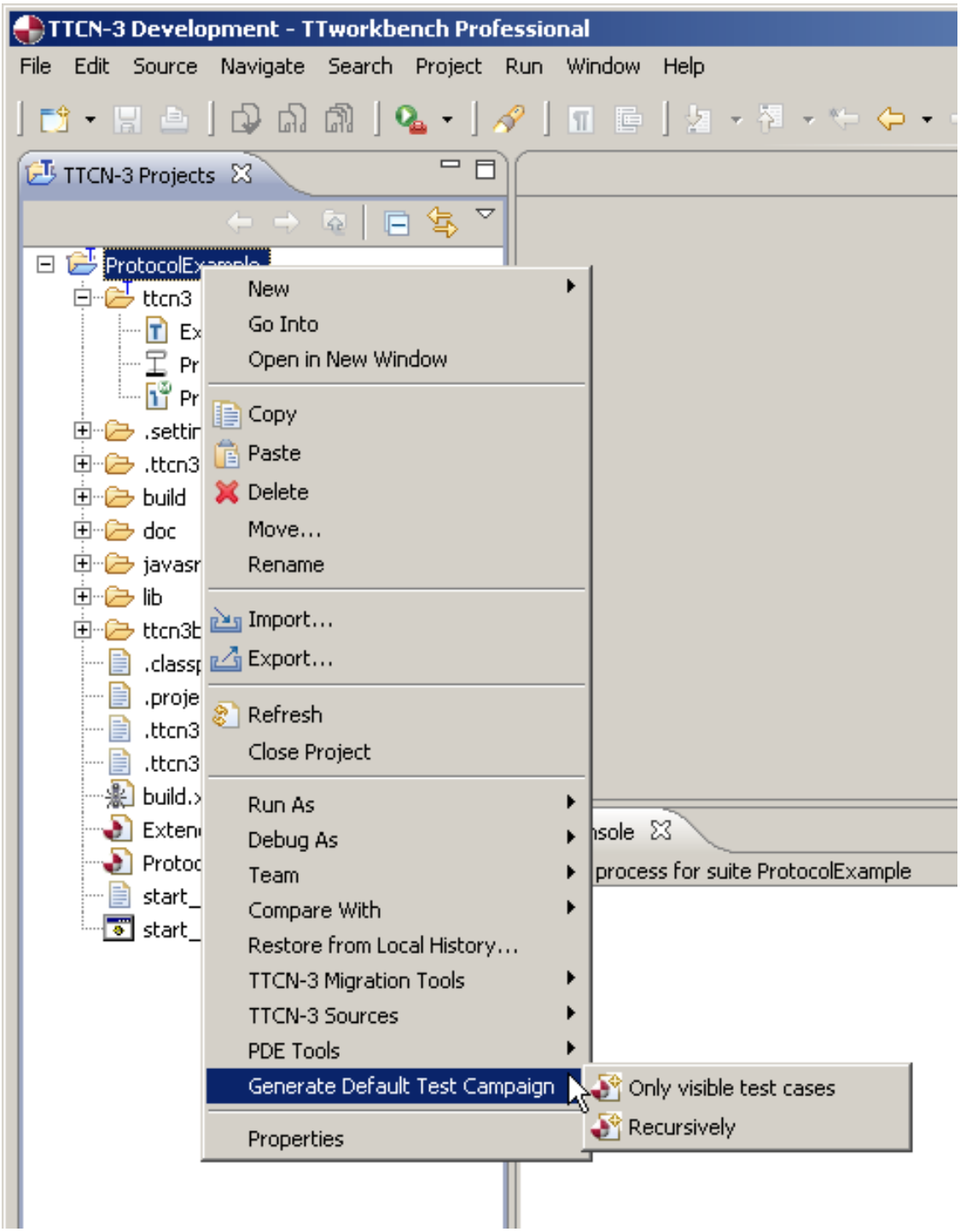
To create a test campaign, you have two options: You can either create a Default Campaign for your project or use the start the test campaign wizard to create a custom test campaign that fits your needs.

Default Campaign

A default campaign contains all test cases of the project's main module and, depending on which default campaign action was used for generation, imported test cases or all test cases of all imported modules. All test case properties (see the Management View section for details) are set to default values.

To allow the generation of a default campaign for a project, you must specify a main module and a test adaptor (see the section called “General”). To actually generate the default campaign, you have to right click on the project in development view, select the item "Generate Default Campaign" as shown in Figure 8.4, “Generating the Default Campain” and then choose one of the two available actions. Only visible test cases adds only test cases that are visible from the main module due to import statements, while Recursively adds all test cases of all imported modules and their imported modules and so on.

Figure 8.4. Generating the Default Campain



The default campaign's .clf file will be named <main module name>.clf and placed in the directory that contains the main module or in the output folder of the project, according to the project settings.

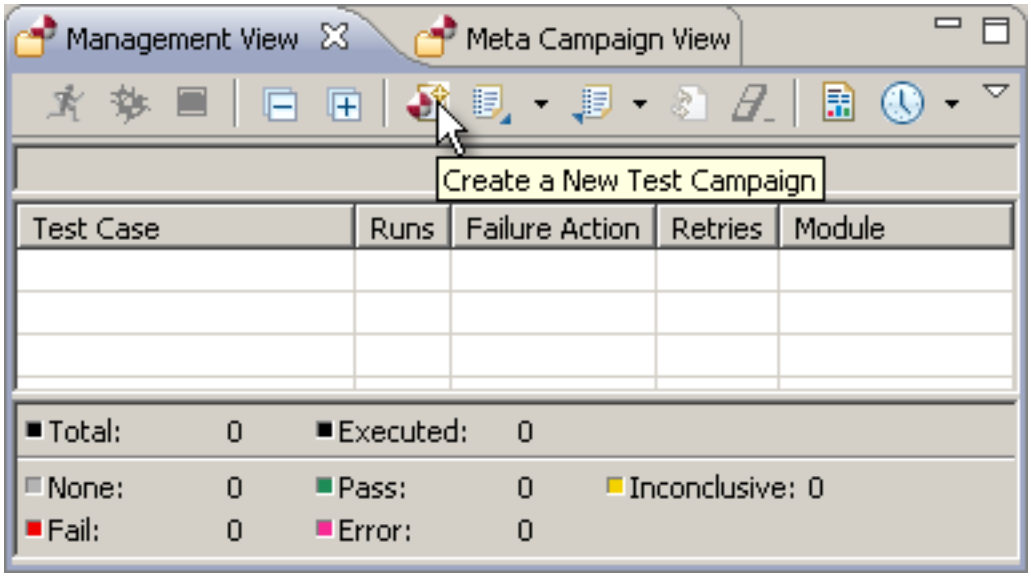
The "Generate Default Campaign" actions are not only available for the TTCN-3 Project but also for TTCN-3 files. In case one of the actions is applied to a TTCN-3 file, a default test campaign will be

generated in a similar way as for the project with the only difference, that the module from the specified file is used as root module.

Test Campaign Wizard

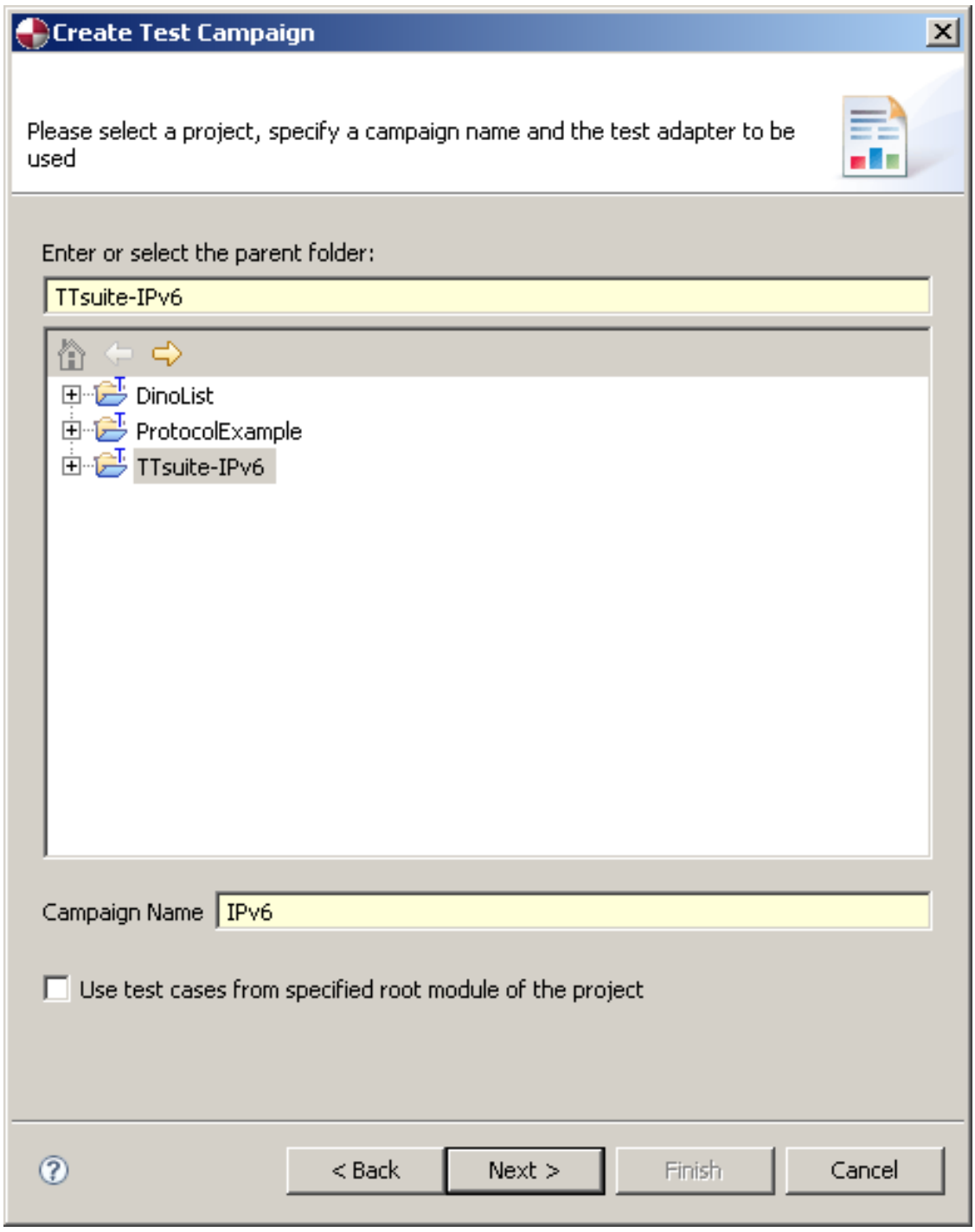
To create a custom test campaign, first start the test campaign wizard.

Figure 8.5. Starting the test campaign wizard



At the first page you have to choose the folder that contains the test suite. Usually this will be the folder of the TTCN-3 project. Then the new test campaign needs a name. If the property "Use test cases from specified main module of the project" is activated, only test cases from the project's main module and its imported modules will be shown in the next step, otherwise all test cases from all modules contained in the project will be included.

Figure 8.6. The test campaign wizard (first page)



The second page of the test campaign wizard allows you to select the test cases that will be executed during the test campaign execution. To put a test case into the area of test campaign on the right side:

- Double-click on the test case of choice, or
- Select the test case and push the > button in the middle.

If all available test cases should be included in the test campaign:

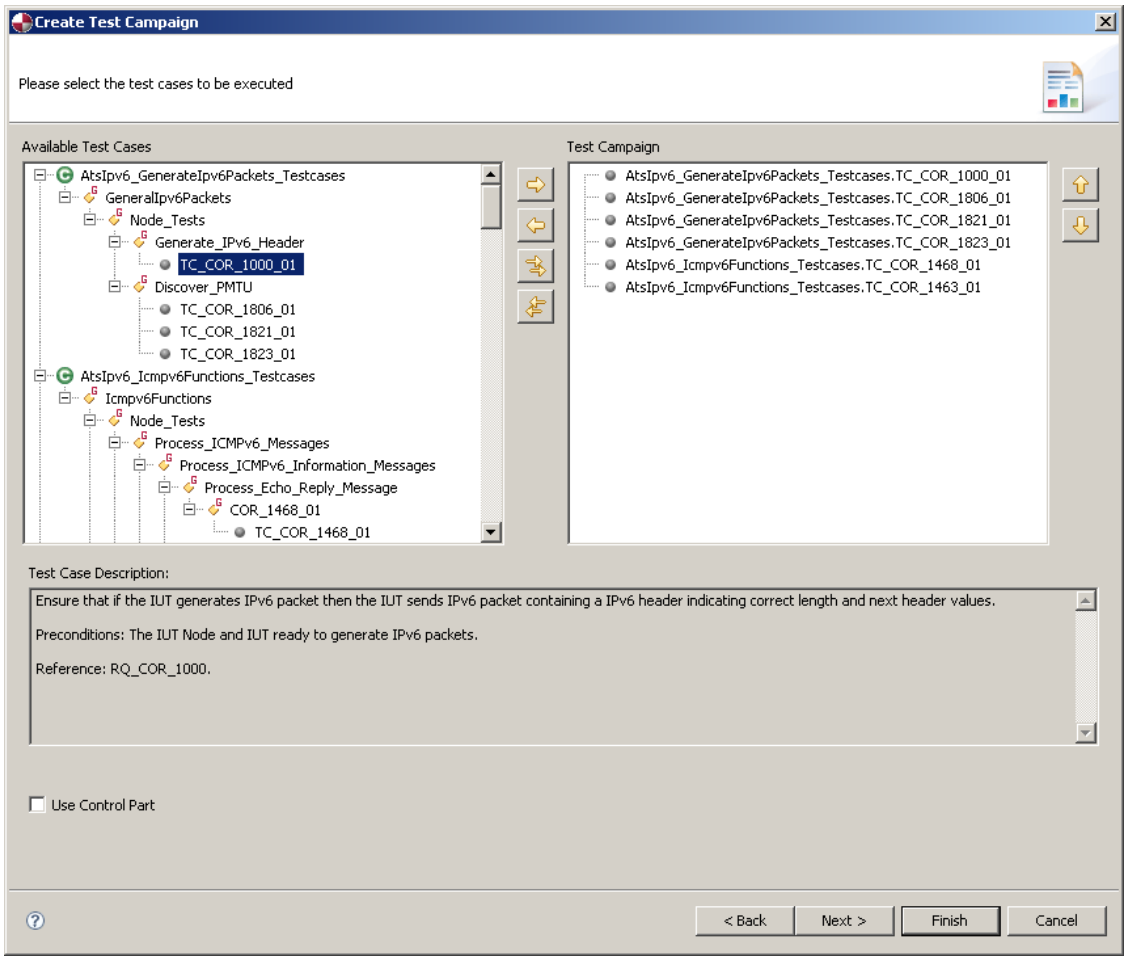
- Double-click on test suite entry, or
- Push the >> button in the middle.

It is also possible to select multiple test cases by additionally using the CTRL key while selecting. The selected test cases are put into the test campaign by simply using the > button.

If test cases have been selected for the test campaign, it is possible to change their order by selecting the specific test case and pushing the up or down arrow button on the right side.

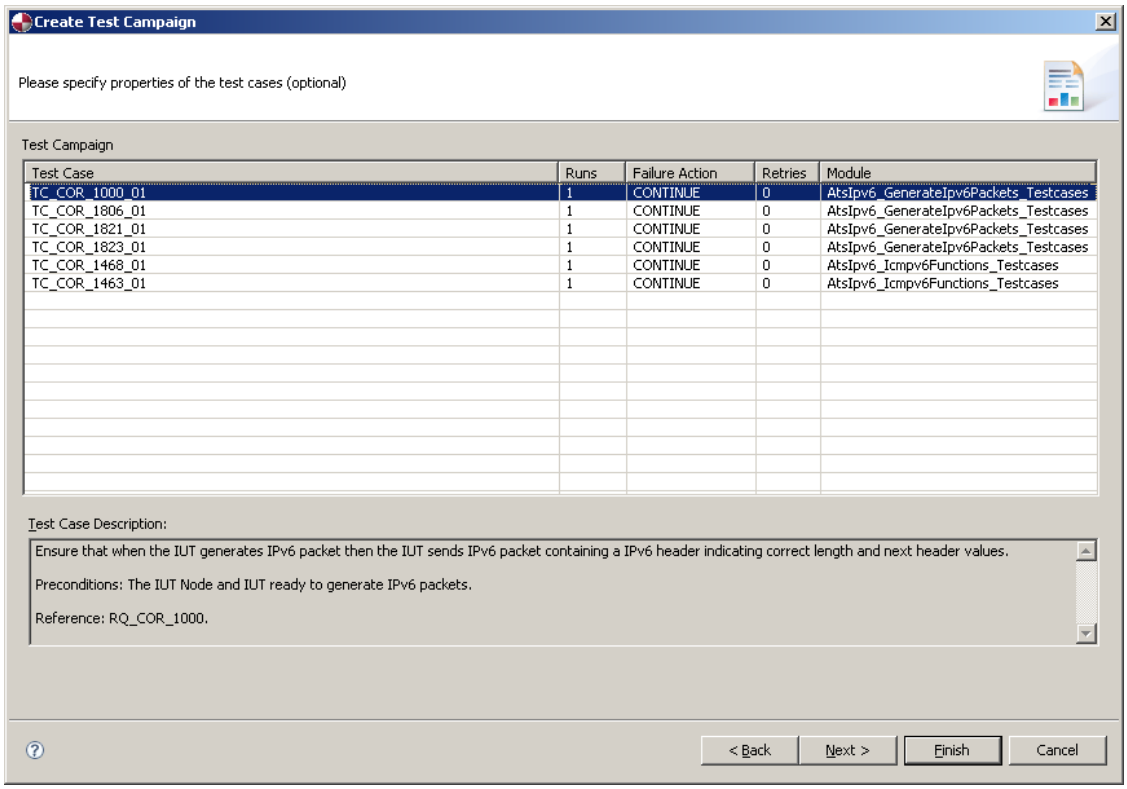
Selecting the checkbox labeled with "Use Control Part" in the lower left corner will advice TTman to load the test campaign dynamically during the test execution depending on its verdicts.

Figure 8.7. Selecting test cases in the test campaign wizard



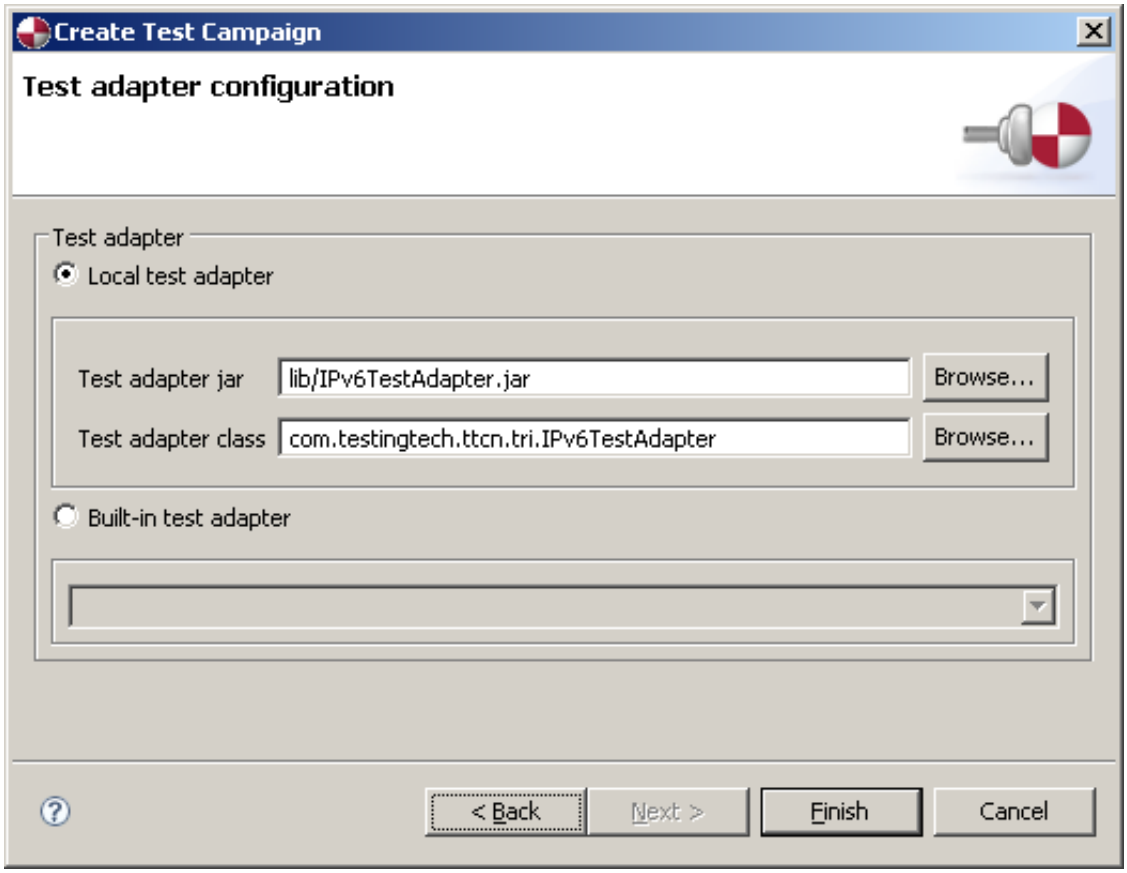
On the third page you can set test case properties. They also can be edited in the Management View; see there for a detailed description.

Figure 8.8. Setting test case properties in the test campaign wizard



The fourth page of the test campaign wizard allows you to select the adapter that will be used by this test campaign. For details see the Setting the Test Adapter section.

Figure 8.9. Setting the test adapter in the test campaign wizard



Meta Campaign

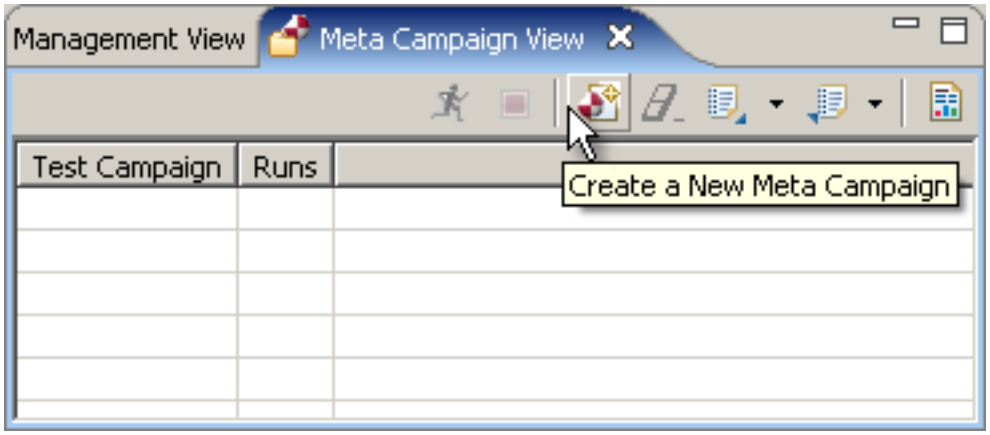
A meta campaign can be used to manage the automatic execution of different test campaigns. It basically does the same with single test campaigns that a test campaign does with single test cases: It allows the user to define an order in which the campaigns get executed and it allows to set parameters.

Meta campaigns get handled in the Meta Campaign View and created in the Meta Campaign Wizard.

Meta Campaign Wizard

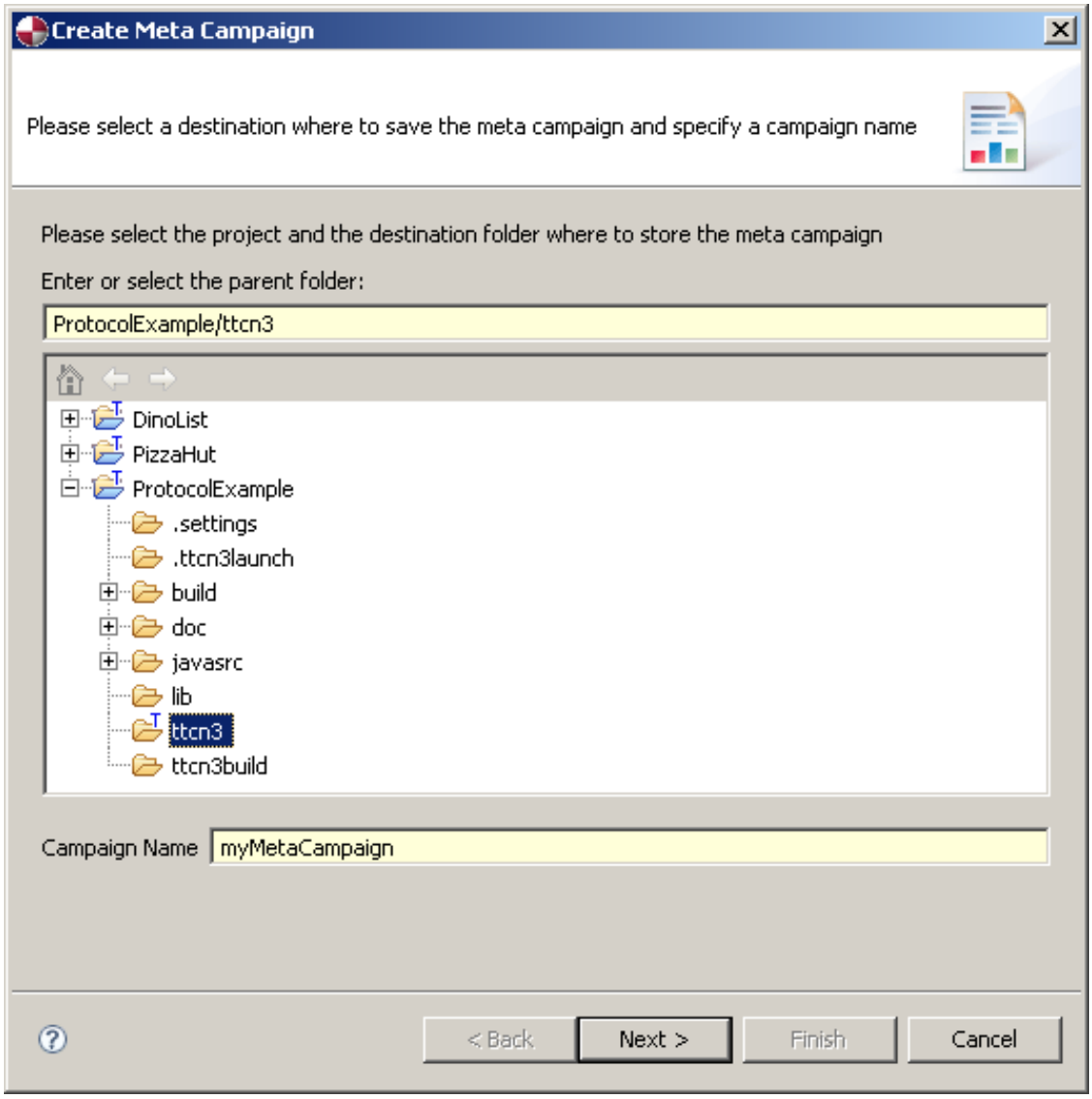
To create a meta campaign, first start the meta campaign wizard.

Figure 8.10. Starting the meta campaign wizard



At the first page you have to choose the location where the meta campaign shall be saved. To complete this wizard page, you also have to choose a name for the new meta campaign.

Figure 8.11. The meta campaign wizard (first page)



The second page of the meta campaign wizard allows you to select the test campaigns that will be executed during the meta campaign execution. You can use test campaigns from different projects for your meta campaign. To put a test campaign into the area of selected test campaigns on the right side:

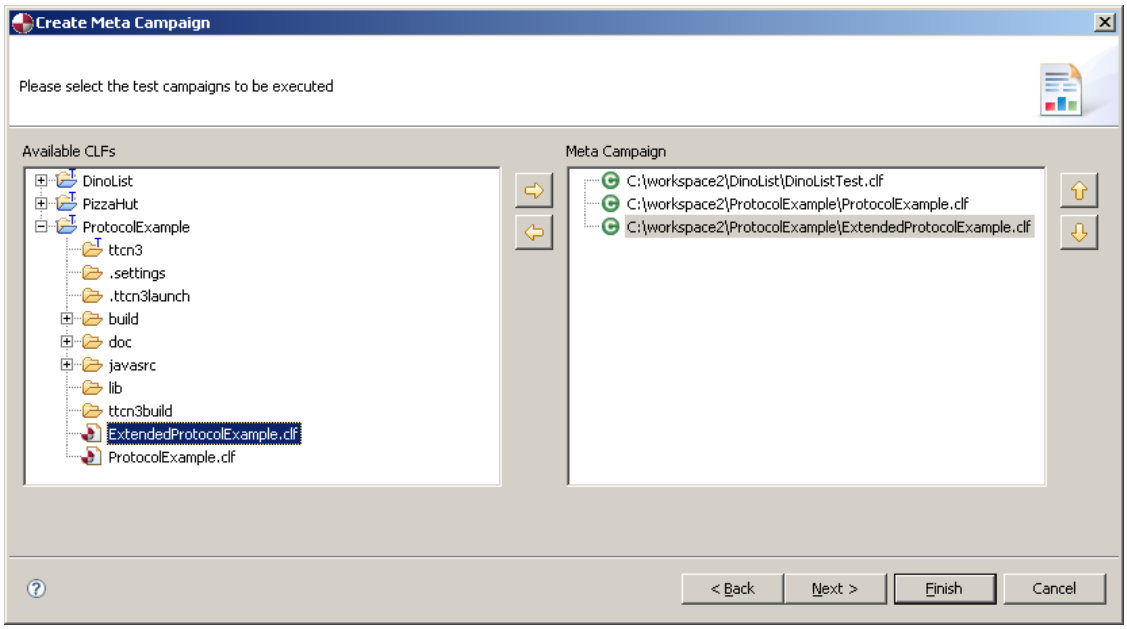
- Double-click on the test campaign of choice, or
- Select the test campaign and push the > button in the middle.

It is also possible to select multiple test campaigns by additionally using the CTRL key while selecting. The selected test campaigns are put into the meta campaign by simply using the > button.

If test campaigns have been selected for the meta campaign, it is possible to change their order by selecting the specific test campaign and pushing the up or down arrow button on the right side.

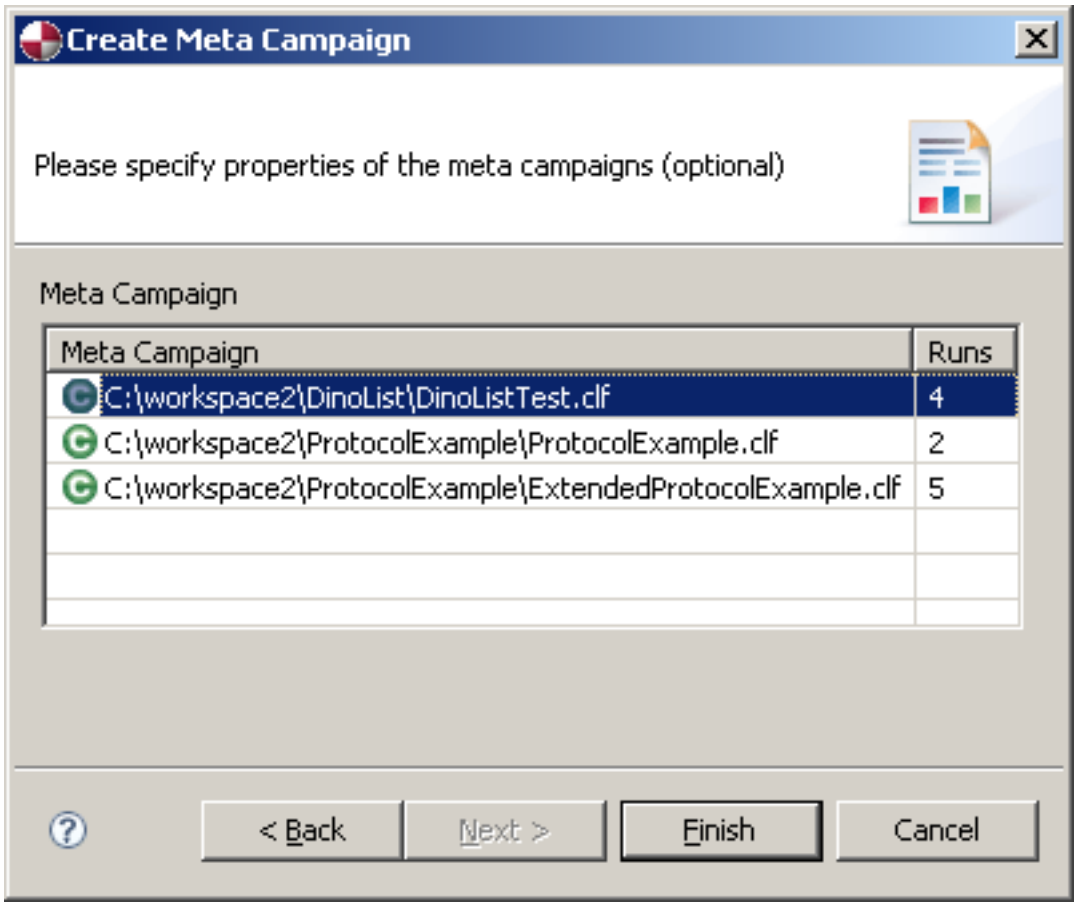
After choosing the test campaigns, you can either choose to select properties for the selected test campaigns in the next wizard page or push the Finish button and thus use default values for the properties. You can also change them later in the Meta Campaign View.

Figure 8.12. Selecting campaigns in the meta campaign wizard



On the third page you can set test campaign properties. Theses are the same that can be edited in the Meta Campaign View.

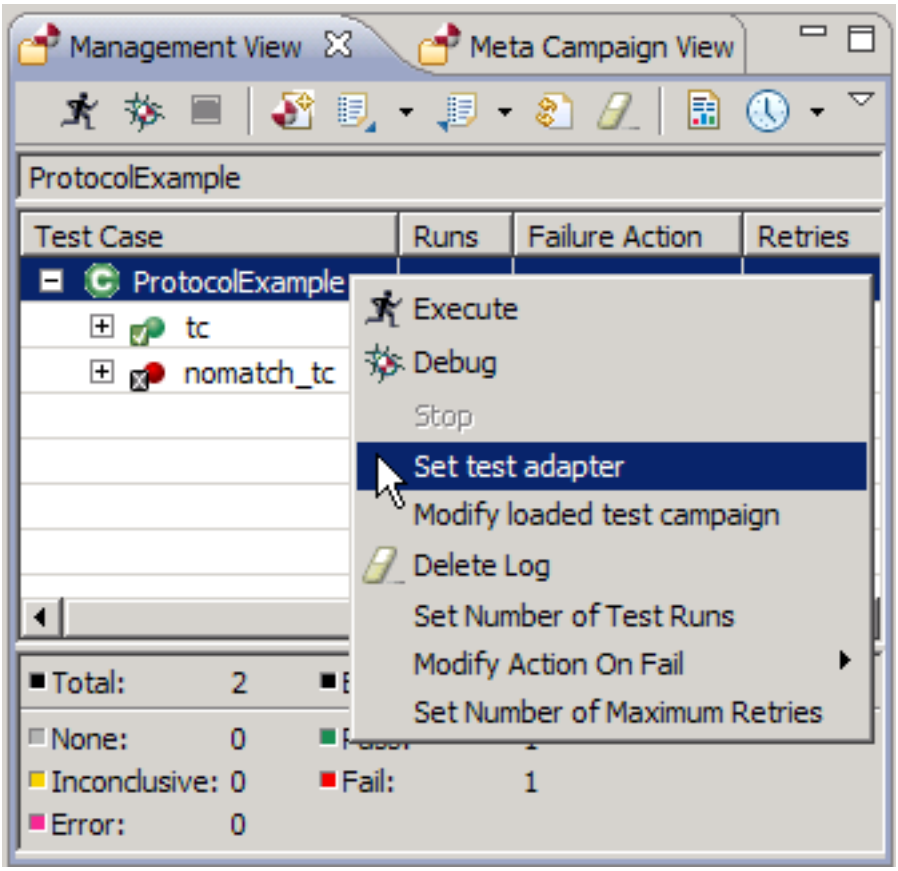
Figure 8.13. Setting test campaign properties in the meta campaign wizard



Setting the test adapter

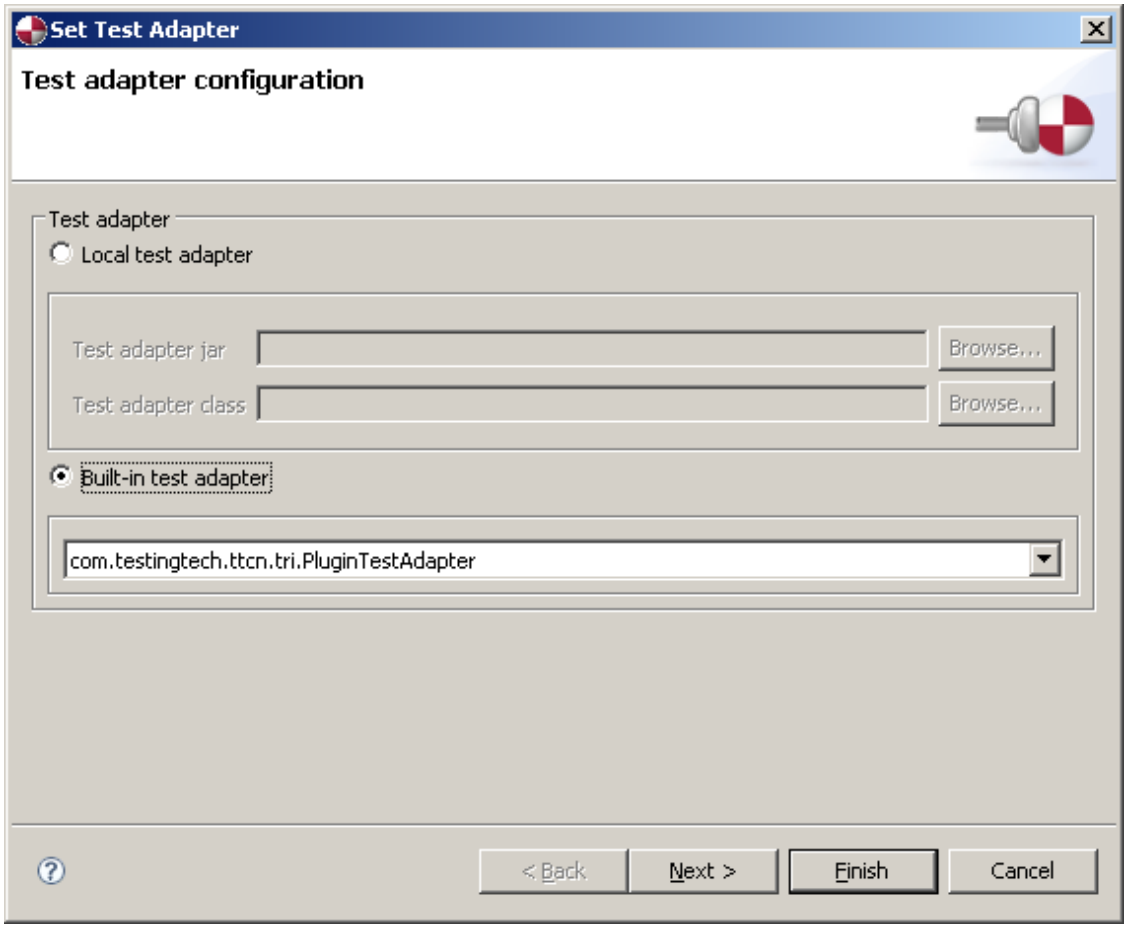
The test adapter that is used by a test campaign during execution can be set by choosing the Set test adapter after right-clicking on the test campaign in the Management View.

Figure 8.14. Setting the test adapter in the Management View



On the Test adapter configuration page, you can either use a test adapter that comes with the selected project or one of the adapters that are included in TTworkbench. For a project-specific test adapter, you first have to choose the .jar file that contains the adapter and then select the test adapter class. For a built-in test adapter, you have to choose one of the adapters in the drop-down list. If you select the built-in PluginTestAdapter, you can continue the dialog by specifying the plugin adapter's details.

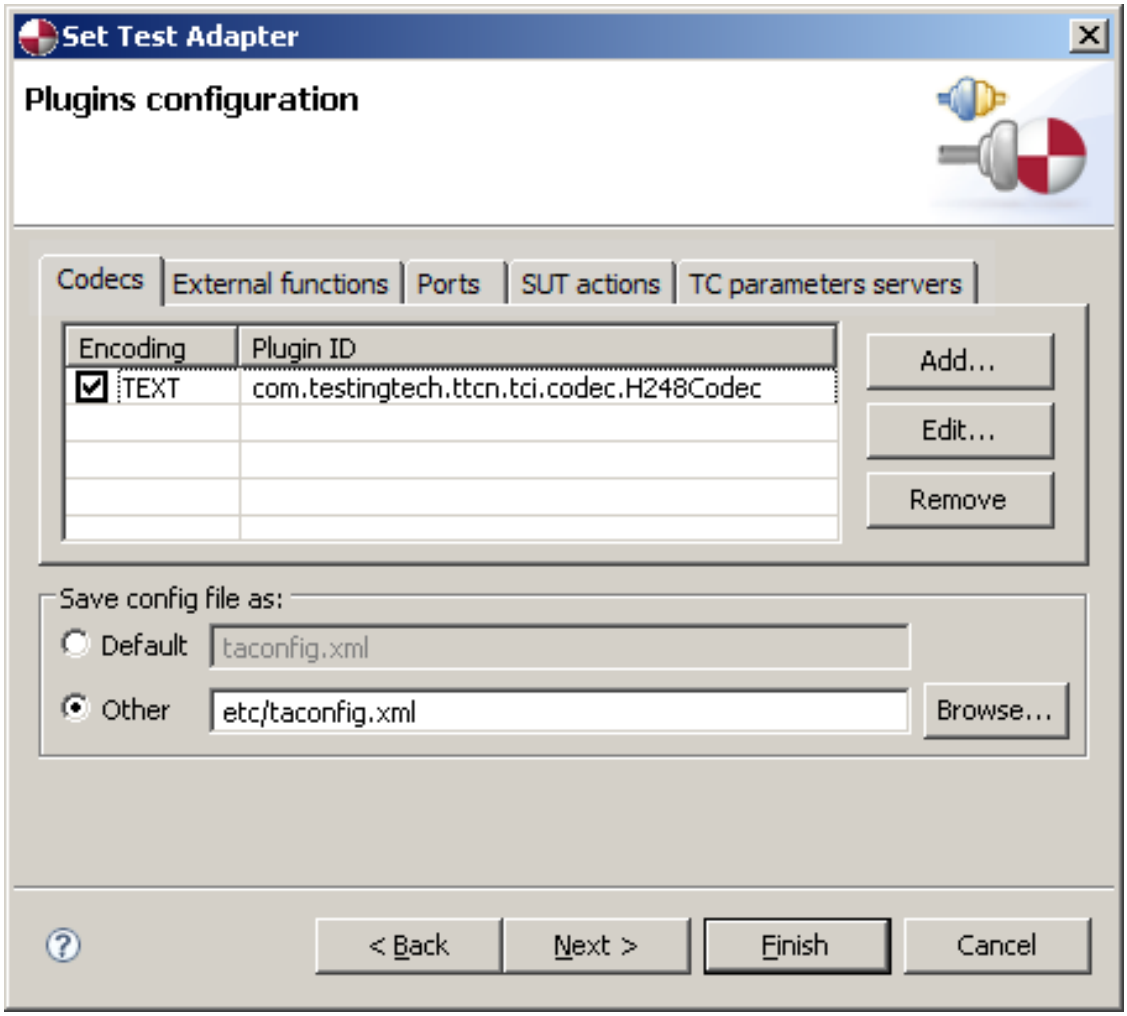
Figure 8.15. Selecting the test adapter



On the Plugins configuration page, you can configure which TT3 runtime plugins have to be used by the PluginTestAdapter. The edited configuration can be saved either in a file named taconfig.xml located in the project root directory by chosing Default on the bottom of the page or at another location by choosing a different file using the Browse... button.

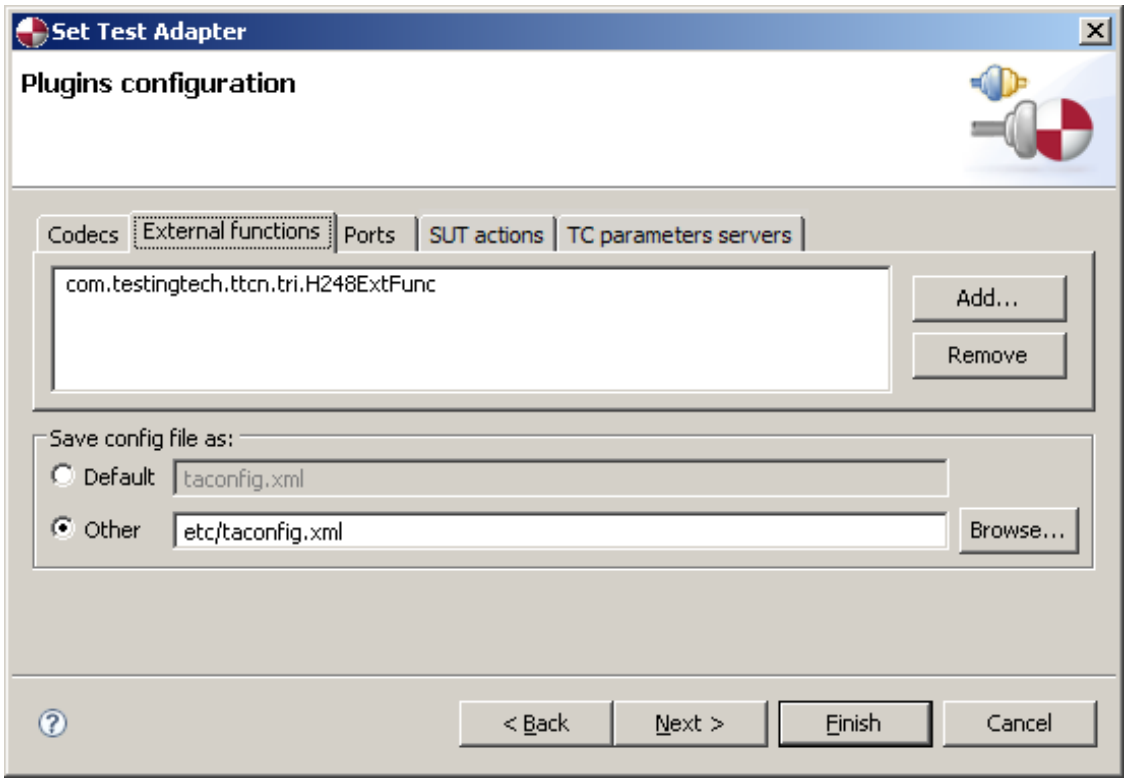
The codec plugins that have to be used can be configured on the initially displayed Codecs tab. Using the buttons located on the right side, different codec configurations can be added, edited or removed. Pressing the 'Add...' or 'Edit...' buttons, a new window is opened allowing you to choose the codec plugin and the encoding name the plugin is assigned to. The default codec can be configured using the checkbox in the codec configurations list.

Figure 8.16. Configuring the codec plugins



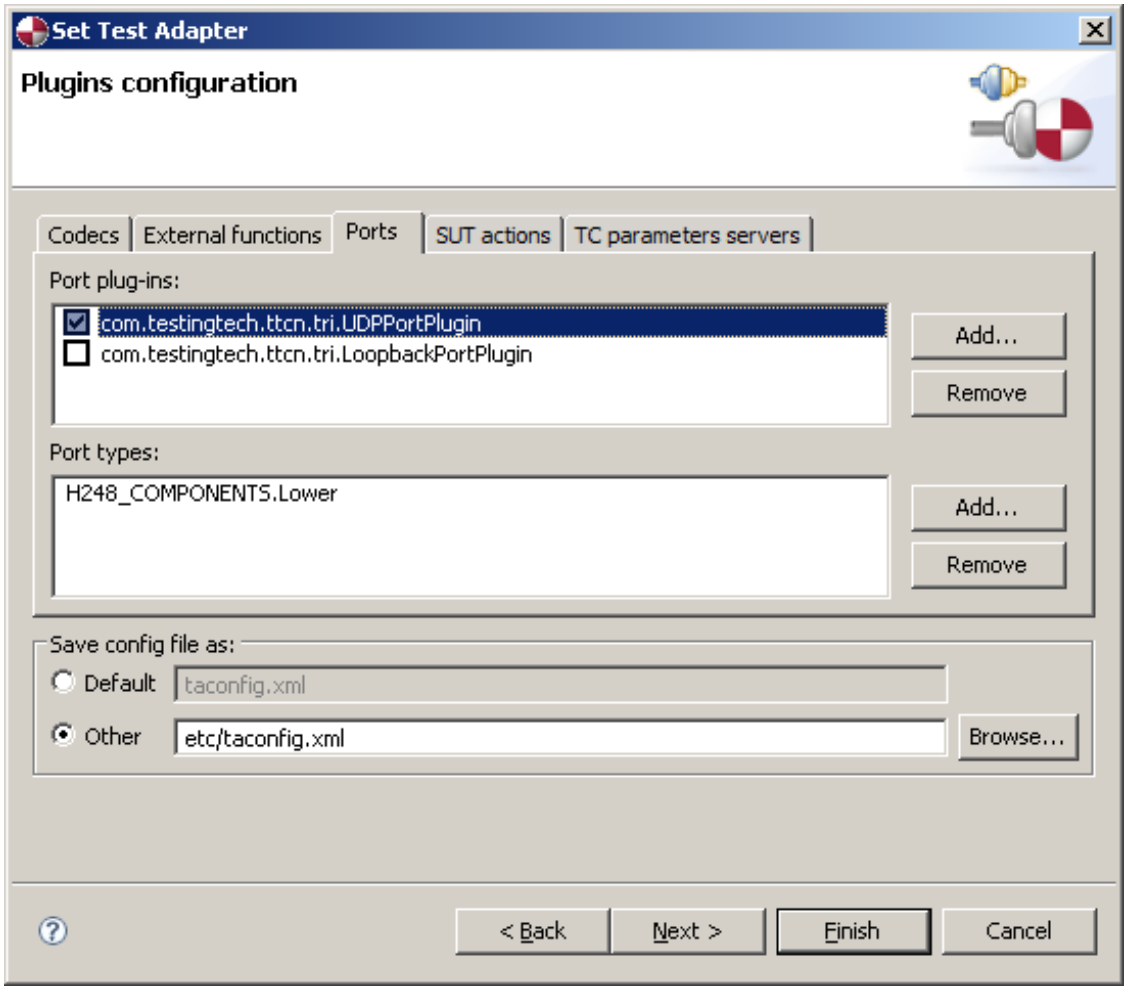
On the External functions tab page you can configure the external function plugins. External function plugins can be added and removed using the two buttons at right.

Figure 8.17. Configuring the external functions plugins



The port plugins are configured on the 'Ports' tab. On the top side of the page, port plugins can be added to or removed from the configuration. A new window is opened when the 'Add...' button is pressed. Within this window a port plugin can be chosen. It is possible to define a port plugin to be the default one, i.e. this plugin is used for all ports that are not covered explicitly by other plugins. The default port plugin can be selected in the same way like the default codec, by using the checkbox in the port plugins list. On the bottom side, you can configure the list of port types the configured port plugin is responsible for. By pressing the 'Add...' button, you can choose a system port type from the list given in the new window that is opened.

Figure 8.18. Configuring the port plugins



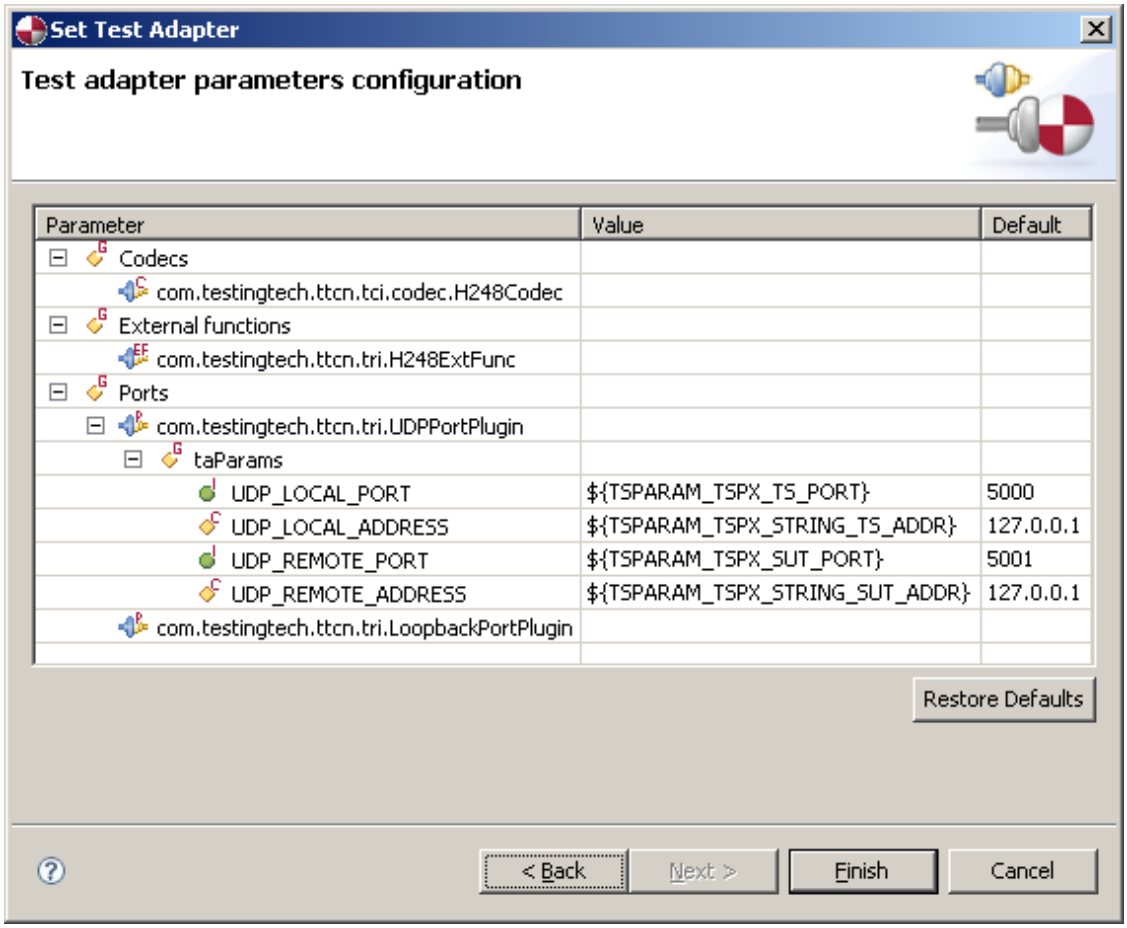
In addition to the built-in SUT action implementation, additional custom SUT action plugins can be configured on the 'SUT action' page.

In case test case having parameters have to be started directly not from the control part, a test case parameter server is needed. Such test case parameter server plugins can be added to and removed from the configuration on the 'TC parameters servers' page.

The parameters specific for each configured runtime plugin can be edited on the test adapter parameters configuration page. The parameters are grouped by the plugin category and within each category by the plugin identifier. The parameter values can be edited directly in the 'Value' table. The values for the parameters may be either literal values (e.g. 5001 as a port name) or module parameter references (e.g. ${PARAM_SUT_ADDRESS}). For module parameter references content assistance is provided and can be activated by pressing CTRL+Space during editing.

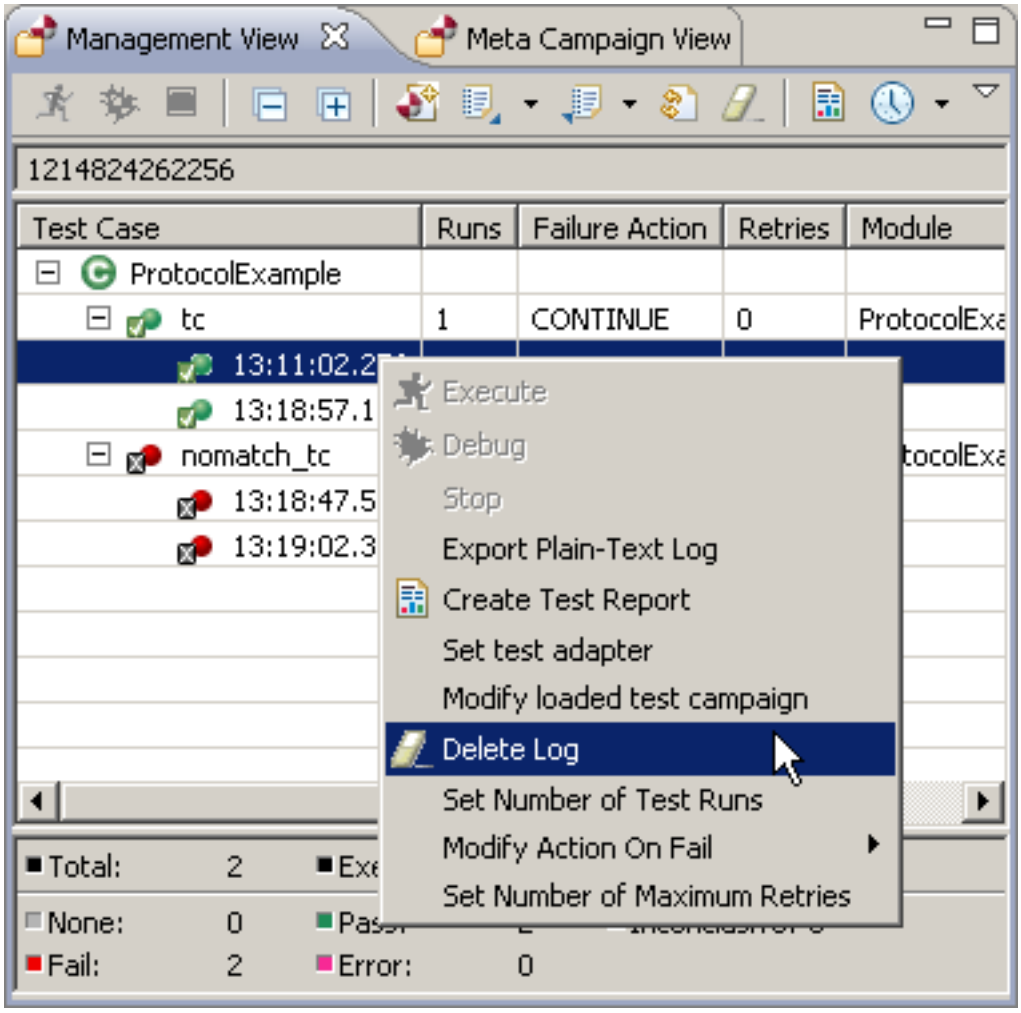
In addition to the global parameters, for port plugins it is possible to define separate parameter sets for different port instances. This can be done by right clicking on the port plugin identifier and choosing the menu item 'Add parameter set'. Selecting this action, a new window is opened where you can select the system port name (content assistance is provided by pressing CTRL+Space during editing). Such a specific configuration can be deleted by right clicking on the port name in the parameters table and choosing the according option. The values for all parameters can be restored to the default values by using the button Restore defaults on the bottom side of the page.

Figure 8.19. Configuring the test adapter parameters



Management View

Figure 8.20. Management view



The management view can be used to load and execute the test cases from a test campaign. It contains at most one test campaign at a time. To reload a loaded test campaign, click on the Reload Test Campaign button  . To unload it, click on the Unload Test Campaign button  .

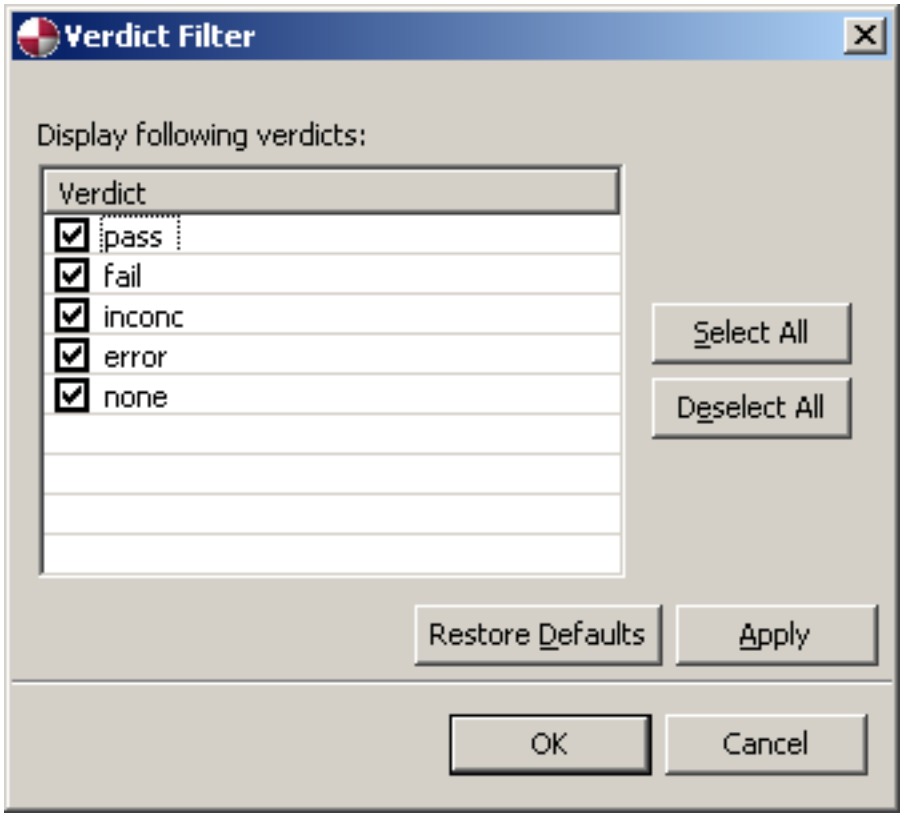
The test cases (shown as children of the module) have properties which can be viewed and edited in the 2nd to 4th column of the Figure 8.20, “Management view”. The "Runs" property regulates how many times in a row the test case will be executed. With "Failure Action" you can define what happens, if the test case finishes with the verdict "fail". The options are to "STOP" the campaign execution, to "CONTINUE" without special treatment or to re-execute the test case, if "RETRY" is chosen. In the latter case, the "Retry" property specifies how many times the test case gets re-executed, until the campaign execution gets continued.

As children of the test cases the finished test runs get displayed - named as the timestamp of their execution. You can change the timestamp format by clicking on the clock symbol in the menu bar of the management view.

In the bottom section of the management view, execution statistics are displayed. To disable this section, select the menu of the management view and choose Execution Statistics. You can reenable them the same way.

To filter the displaying of test cases based on their verdicts in the Management View, in the menu of the Management View select **Verdict Filter**. A dialog will pop up and the verdicts can be chosen.

Figure 8.21. Verdict Filter



Clicking on the campaign name (highlighted by the green C icon) will display the campaign properties together with status information in the properties view (see Properties View). Clicking on a test case will display information about the selected test case.

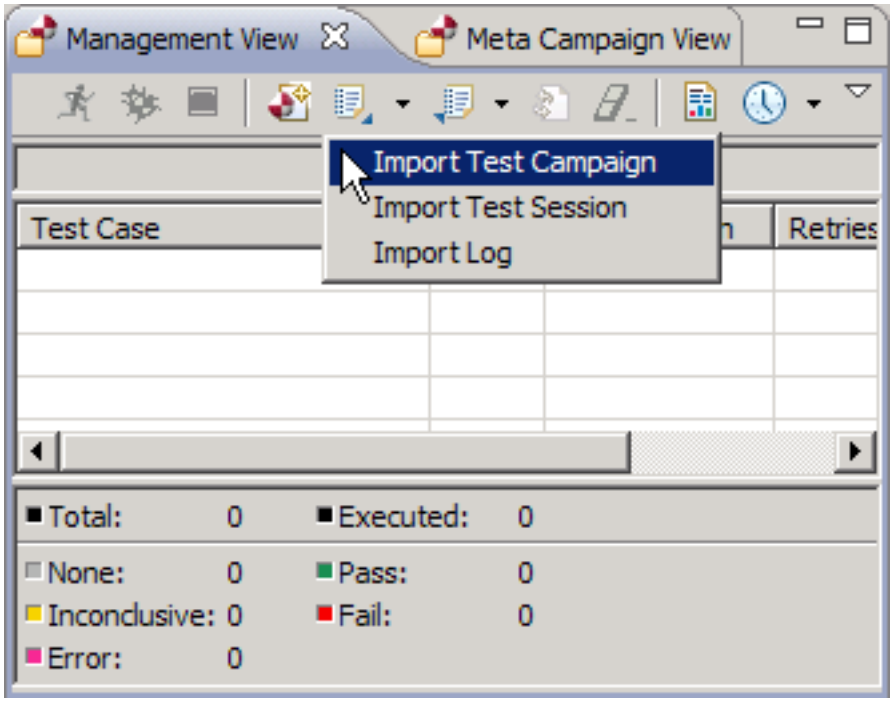
Test Case Execution

Note

To execute test cases, a test campaign must exist. If you didn't create one or specified a default test campaign, you should do this now (see the section called “Test Campaign”).

First you have to import the test campaign. The test suite will be loaded now.

Figure 8.22. Importing a test campaign



The management view also enables the user to export the loaded test campaign by choosing Export Test Campaign from the Export Files menu of the management view's menu bar. The actual test campaign is then exported to a campaign loader file (`*.clf`) which of course can be imported again later.

After successfully loading the test campaign, the user is able to execute the test cases in the test campaign. If the campaign is not configured to use a control part, a test case or a set of test cases has to be selected for the execution. To execute a test case:

- Double-click on the test case, or
- Click on the Execute button provided by the context menu, or
- Click on the Execute button in the management view's menu bar

If all test cases of the test campaign or a control part are to be executed:

- Double-click on test suite entry, or
- Select the test suite entry, click either on the Execute button from the management view's menu bar as shown in Figure 8.23, “Start a test case from the menu bar” or on the Execute button from its context menu as illustrated in Figure 8.24, “ Start a test case from the context menu ”.

Figure 8.23. Start a test case from the menu bar

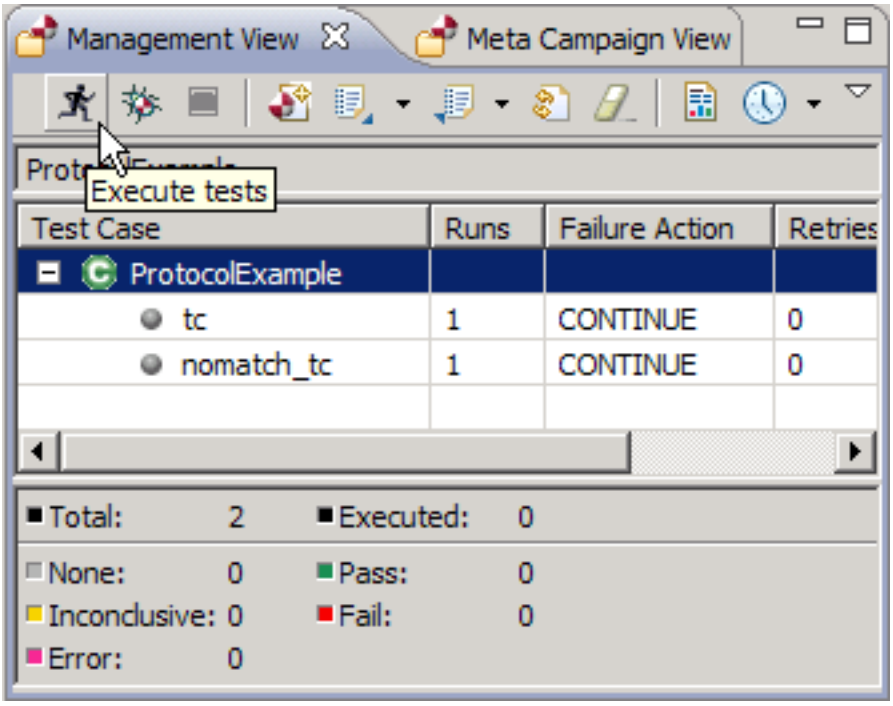
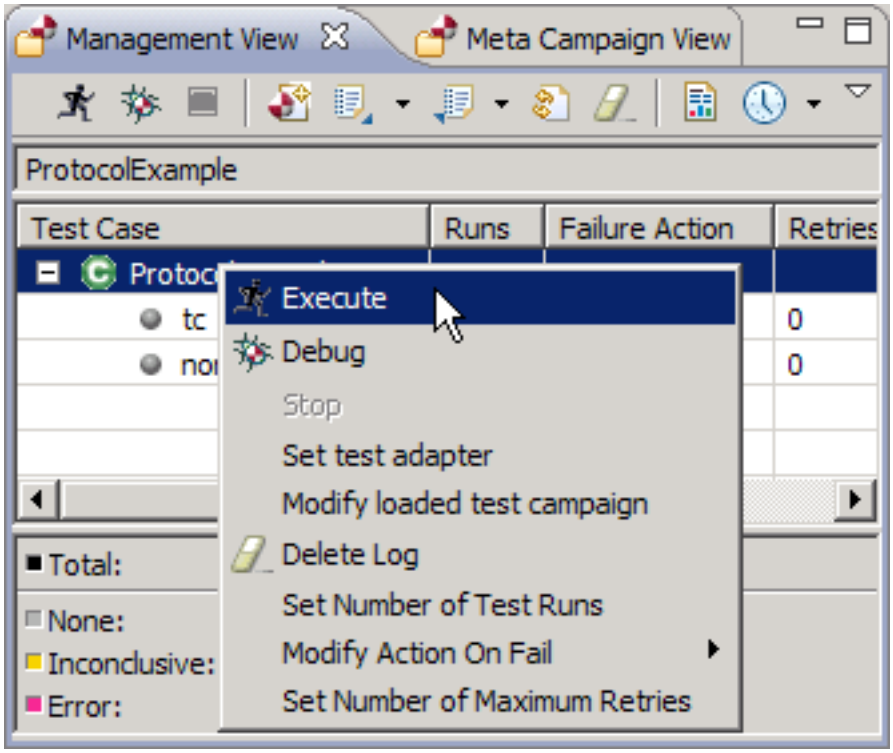


Figure 8.24. Start a test case from the context menu



Note

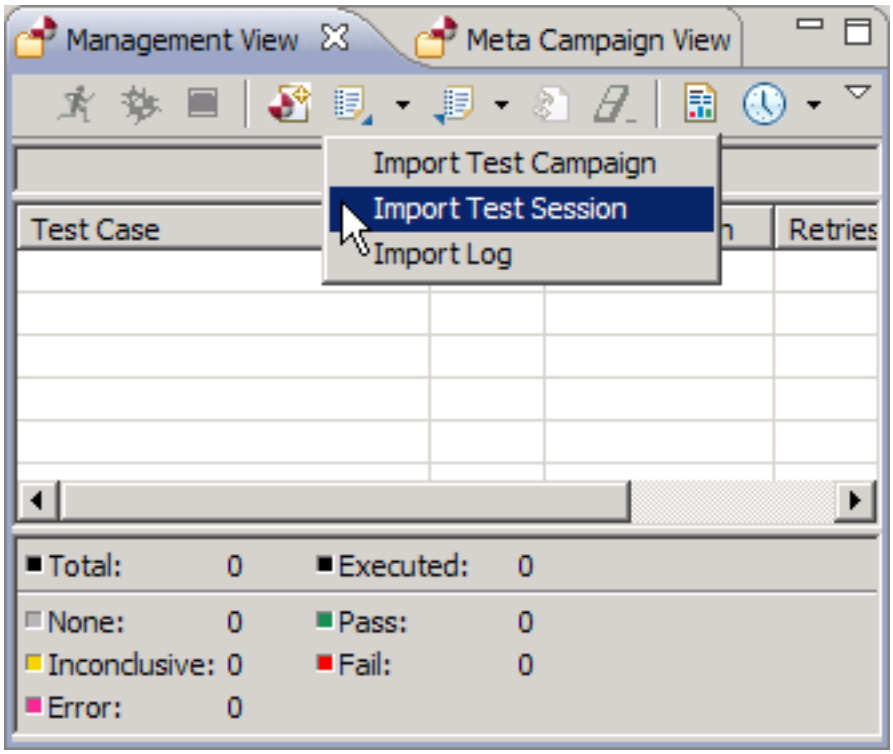
On most machines a personal firewall is running. Sometimes the security options are too high so TTworkbench is not able to communicate. Please make sure that your security level is not too high, or create exception rules for it. For further instructions please refer to the users guide of your security software.

Test Session

A test session represents a test campaign and all current test results and respective test logs. Test sessions can be saved and resumed via exporting and re-importing.

To export a test session choose Export Test Session from the Export Files menu of the management view's menu bar. To import a test session choose Import Test Session from the Import Files menu of the management view's menu bar.

Figure 8.25. Importing a Test Session



Test Log

For each test campaign executed with TTman, a test log gets created, containing all the events generated by the execution process. It is possible to import and export log files from and to the file system. The log file for a test log will be saved file as a zipped file (`*.tlz`) and stores all generated logging information together with the test campaign file that was used for the respective test run, including the current setting of the module parameters. The location of the file can be specified after clicking on the Export Files button on the Management View's menu bar and selecting Export Log

Note

The tlz-file can be unzipped into the pure log file and a test campaign file.

The user can import a tlz-file to review the test execution process by using the Import Files button on the Management View's menu bar and selecting Import Log. It is also possible to load old log files (`*.log`).

It is also possible to export the log in a human readable form by clicking on the Export Files button on the Management View's menu bar and selecting Export Plain-Text Log. After selection a location, a `.zip` archive gets stored that contains both the according `.clf` and a `.txt` file. The latter contains the current log formatted to be easily readable outside of TTworkbench. This functionality also is accessible by using the according entry in the log's context menu in the Management View.

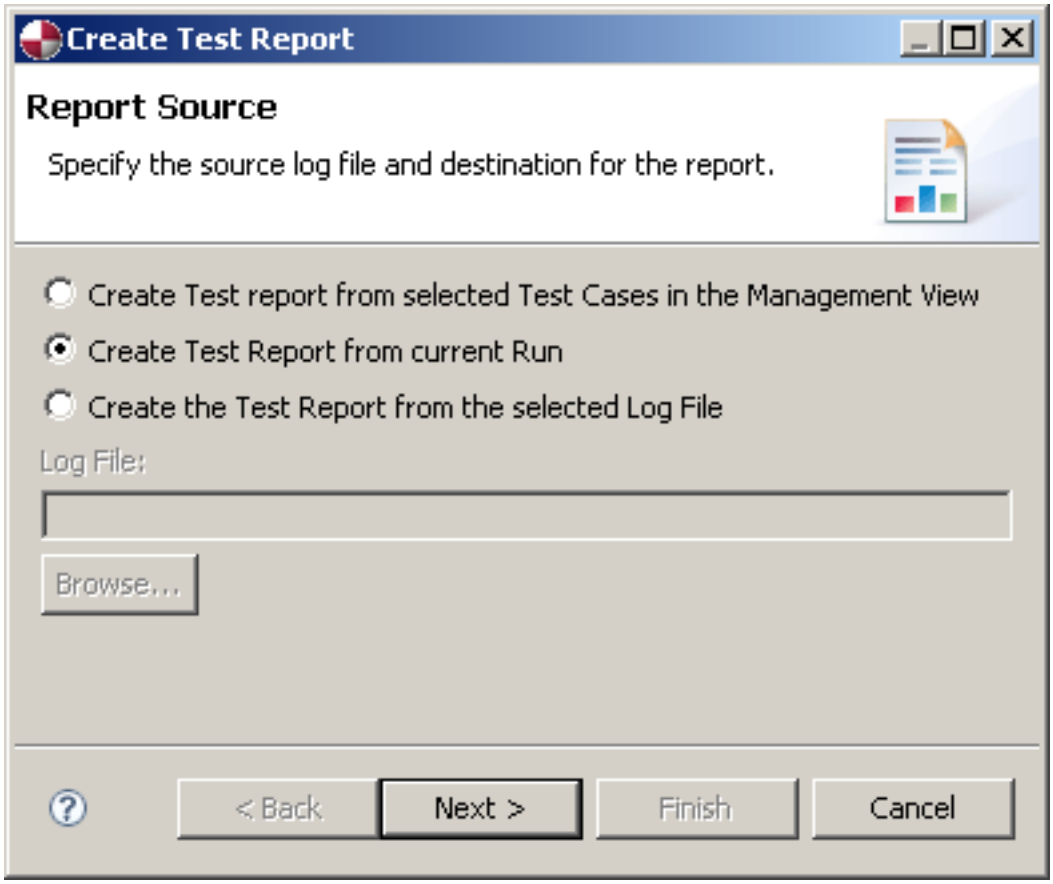
Test Report Generation

TTman supports the generation of test reports in HTML, PDF, Excel, or Word format. The Excel format is an XML file that can be opened with Microsoft Office Excel 2003 and later versions as well as with OpenOffice 2.0. The Word format is also an XML file that can be opened with Microsoft Office Word 2003 and later versions. To do so, start the test report generation wizard by clicking the Create Test Report button

.

On the Report Source page, you can choose between generating test reports only from the in the Management View selected test cases of the current test case run, from the complete current test case run, or from a previously exported log file.

Figure 8.26. Selecting the data source for the Test Report



The Report Destination page allows you to select a destination directory, a test report file name, and the type (HTML, PDF, Excel, or Word) of the generated report. The directory and the report name get saved and used as default values the next time you generate a test report for the same project.

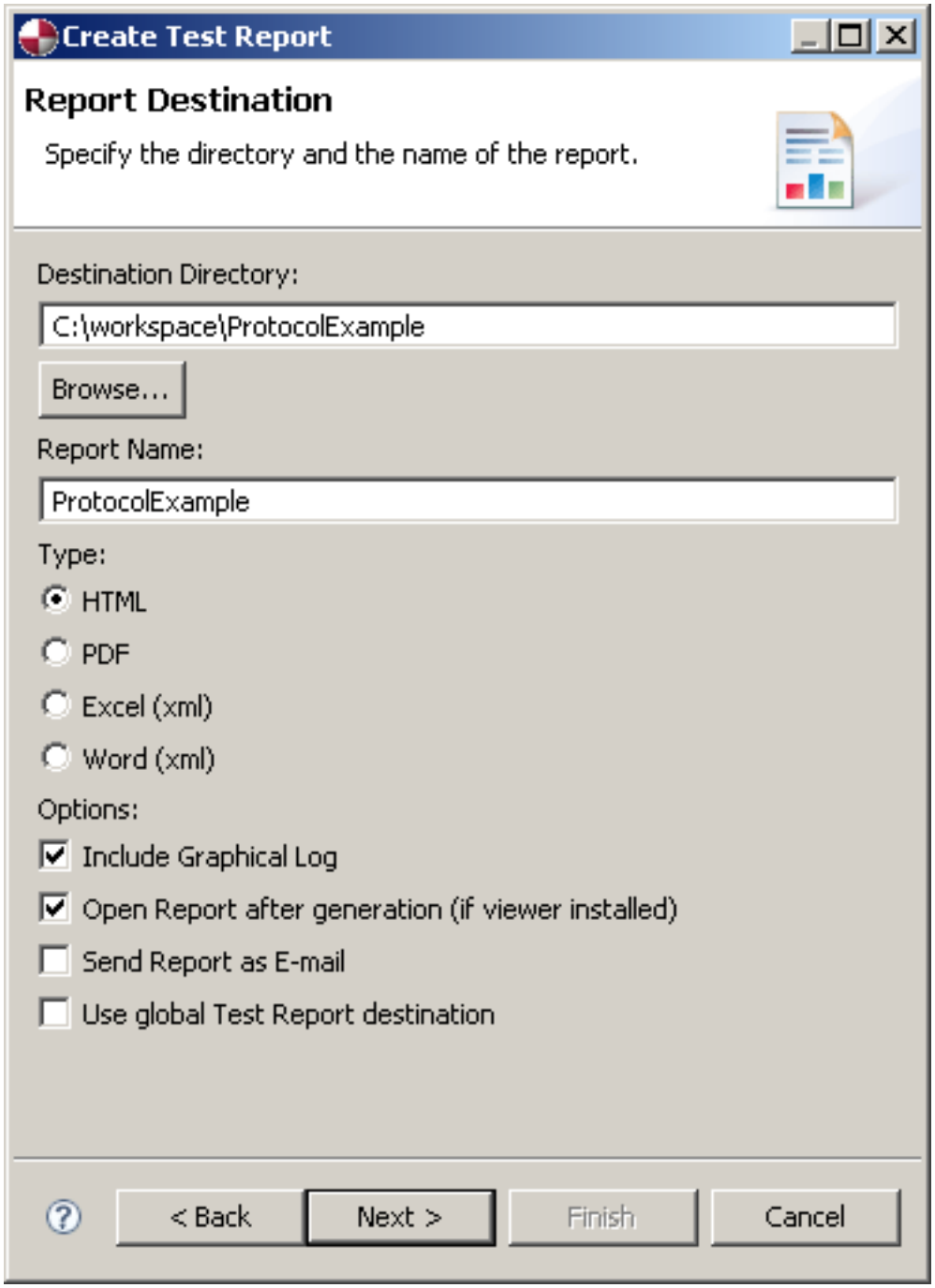
The option Include Graphical Log sets if an image will be generated and saved in the test report directory for each test case. In the test report, the test case name will be linked to the correspondent image file.

The option Open Report after generation (if viewer installed) can be used to set if the test report opens automatically in an appropriate viewer after its creation. For PDF, Excel, and Word reports, an appropriate viewer has to be installed.

With the option Send Report as E-mail, you can enforce the sending of the generated test report as e-mail. Before enabling this option, it has to be ensured, that the Emailing Preferences are set correctly.

If the option Use global Test Report destination is set, destination directory and report name get used as default values the next time you generate a test report, regardless from if it is generated for the same project or not.

Figure 8.27. Setting test report type and destination

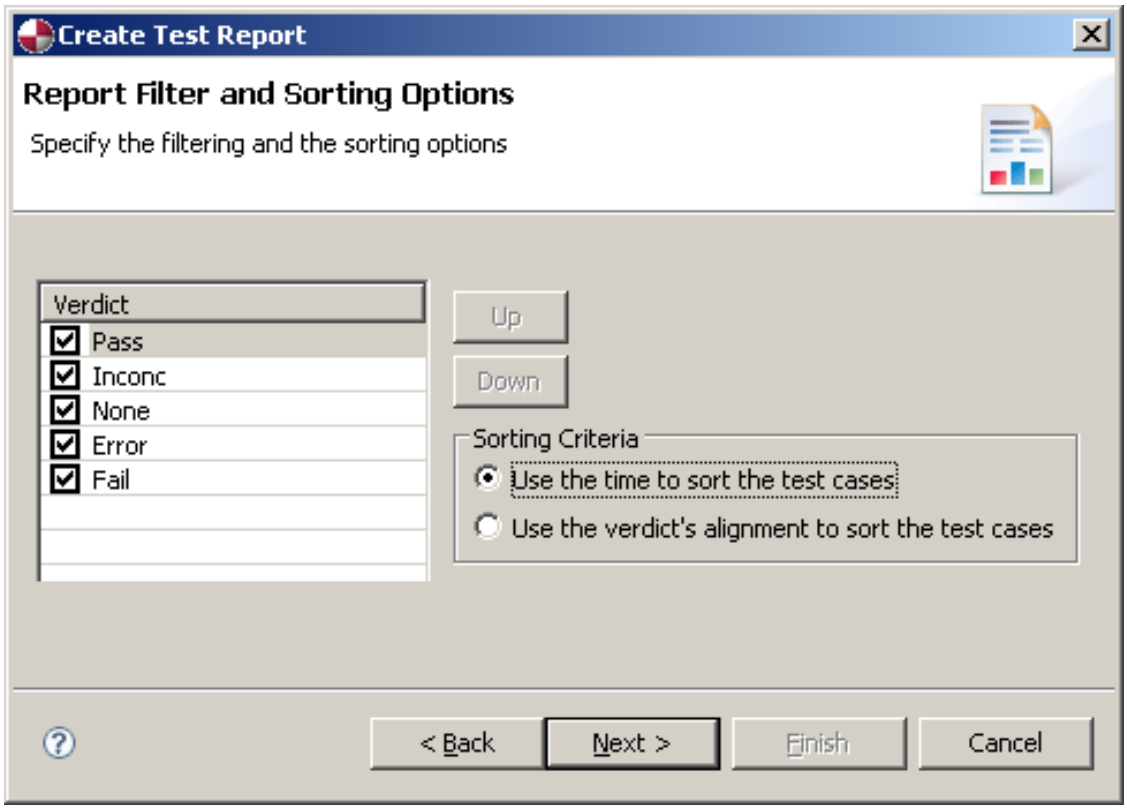


The Report Filter and Sorting Options page allows you to define which test cases shall be included in the report depending on their verdict by checking or unchecking the checkboxes at the left. With the Sorting Criteria box, you can specify how the test cases shall be sorted in the test report. If you set it to Use the verdict's alignment to sort the test cases, you can define the resulting order by rearranging the verdicts on the left by using the Up and Down buttons.

Note

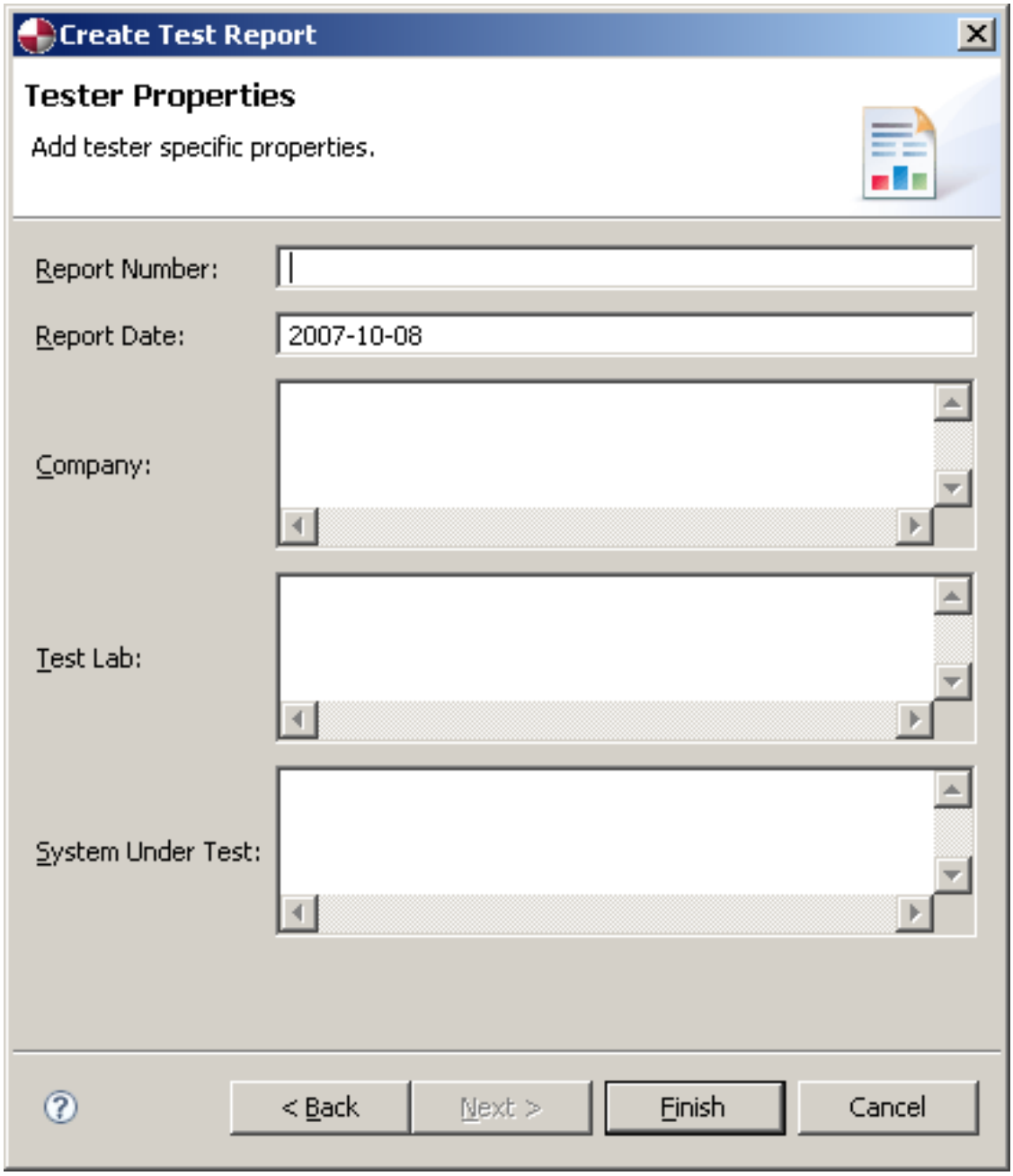
Neither filtering nor sorting are available for XML test reports yet.

Figure 8.28. Setting filtering and sorting options



On the Tester Properties page, you have the possibility to enter a report date, report number, company name, test lab, and system under test (SUT) that get included in the report.

Figure 8.29. Tester Properties



40%

Campaign Configuration

Campaign Name	SIP_RFC3261_Eval
Campaign File	SIP_RFC3261_Eval2.clf

Test Adapter

Class	com.testingtech.ttcn.tri.SipNist.SIP3261_TestAdapter
File Name	TTsuite-SIPEtsi.jar

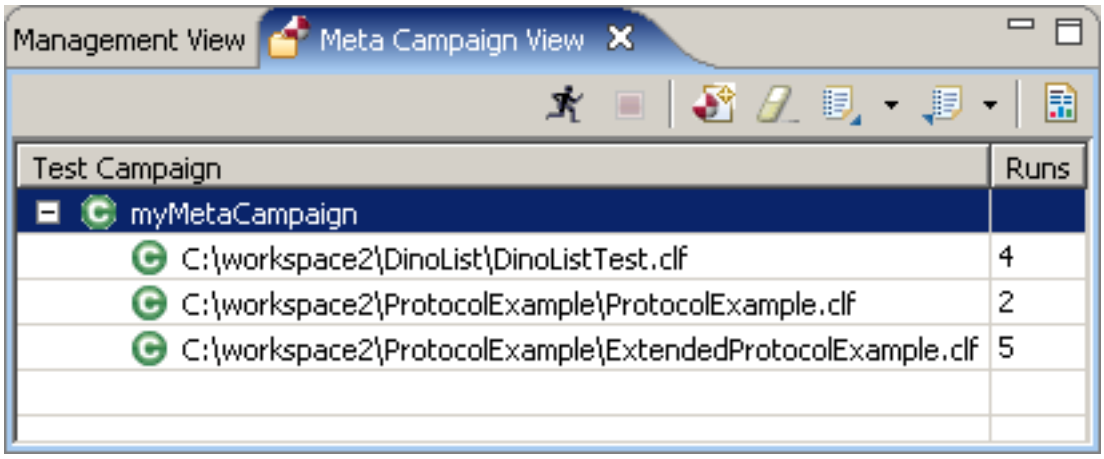
Modules

Name	SIP_CallControl
File Name	SIP_CallControl.jar
Package	
Name	SIP_Registration
File Name	SIP_Registration.jar
Package	

Timestamp	Test Case	Test Purpose	Verdict
2007-09-19 13:38:19.481 2007-09-19 13:38:35.738	SIP_CallControl.SIP_CC_TE_CE_V_001	Ensure that the IUT on receipt of an INVITE request, sends a Success (200 OK) or a provisional (101-199) response.Preconditions: IUT can behave as User Agent serverReference: 8.2 [1], 13.3.1.1 [1], 8[1]	pass
2007-09-19 13:38:36.222 2007-09-19 13:38:40.108	SIP_CallControl.SIP_CC_TE_CE_V_002	Ensure that the IUT on receipt of an INVITE request with a Request-URI set with a scheme that it does not support, sends a Unsupported URI scheme (416 Unsupported URI scheme) response.Preconditions: IUT can behave as User Agent serverReference: 8.2.2.1 [1]	pass
2007-09-19 13:38:40.331 2007-09-19 13:38:47.467	SIP_CallControl.SIP_CC_TE_CE_V_003	Ensure that the IUT on receipt of an INVITE request with a Request-URI set with an address that it does not accept sends a Not Found (404 Not Found) response.Preconditions: IUT can behave as User Agent serverReference: 8.2.2.1 [1]	fail
2007-09-19 13:38:47.745 2007-09-19 13:38:56.074	SIP_CallControl.SIP_CC_TE_CE_V_004	Ensure that the IUT on receipt of an INVITE request with a Timestamp header, when it answers with a provisional response Trying (100 Trying), set a Timestamp header with same timestamp value and optional an increased delay value in its response.Preconditions: IUT can behave as User Agent serverReference: 8.2.6.1 [1], 8[1]	fail
2007-09-19 13:38:56.403 2007-09-19 13:39:03.173	SIP_CallControl.SIP_CC_TE_CE_V_005	Ensure that the IUT on receipt of an INVITE request including an Expires header set to 0, sends a Request Terminated (487 Request Terminated) responsePreconditions: IUT can behave as User Agent serverReference: 13.3.1 [1]	fail
2007-09-19 13:39:03.405 2007-09-19 13:39:09.029	SIP_CallControl.SIP_CC_TE_CE_V_006	Ensure that the IUT on receipt of an INVITE request including no message body, includes in its first 2xx response an initial offer session description.Preconditions: IUT can behave as User Agent serverReference: 13.2.1 [1], 13.3.1 [1]	pass
2007-09-19 13:39:09.235 2007-09-19 13:39:15.337	SIP_CallControl.SIP_CC_TE_CE_V_007	Ensure that the IUT on receipt of an INVITE request including an initial offer session description in its message body, includes the answer in its first 2xx response a session description.Preconditions: IUT can behave as User Agent serverReference: 13.2.1 [1], 13.3.1 [1]	pass
2007-09-19 13:39:15.609 2007-09-19 13:39:22.206	SIP_CallControl.SIP_CC_TE_CE_V_008	Ensure that the IUT on receipt of an INVITE request including a message body with a Content-Disposition header not set to session value, includes in its first 2xx response an initial offer session description.Preconditions: IUT can behave as User Agent serverReference: 13.2.1 [1], 13.3.1 [1]	pass
2007-09-19 13:39:22.435 2007-09-19 13:39:28.141	SIP_CallControl.SIP_CC_TE_CE_V_009	Ensure that the IUT on receipt of an INVITE request including a Content-Language header value that it cannot understood and a Content-Disposition header including a handling set to"optional", includes in its first 2xx response an initial offer session description.Preconditions: IUT can behave as User Agent serverReference: 13.2.1 [1], 13.3.1 [1], 20.11 [1]	pass
2007-09-19 13:39:28.604 2007-09-19 13:39:45.122	SIP_CallControl.SIP_CC_TE_CE_V_010	Ensure that the IUT on receipt of an INVITE request including a Content-Language header value that it cannot understood, a Content-Disposition header including a handling empty, sends an Unsupported Media Type (415 Unsupported Media Type) response with an Accept header that lists the types of all bodies it understands.Preconditions: IUT can behave as User Agent serverReference: 13.2.1 [1], 13.3.1 [1], 20.11 [1]	fail

Meta Campaign View

Figure 8.31. Meta Campaign view



The meta campaign view is the place where meta campaigns get displayed and where you can set the properties for each included test campaign. Here you also can load and execute your meta campaigns (the view contains at most one test campaign at a time). The entries of the currently loaded meta campaign are displayed in the according to their structure. To unload the meta campaign, click on the Unload Meta Campaign button  .

The test campaigns (shown as children of the module) can be set to run multiple times. This property can be viewed and edited in the 2nd column of the Figure 8.31, “Meta Campaign view”.

As children of the test campaigns the finished campaign runs get displayed - named as the timestamp of their execution.

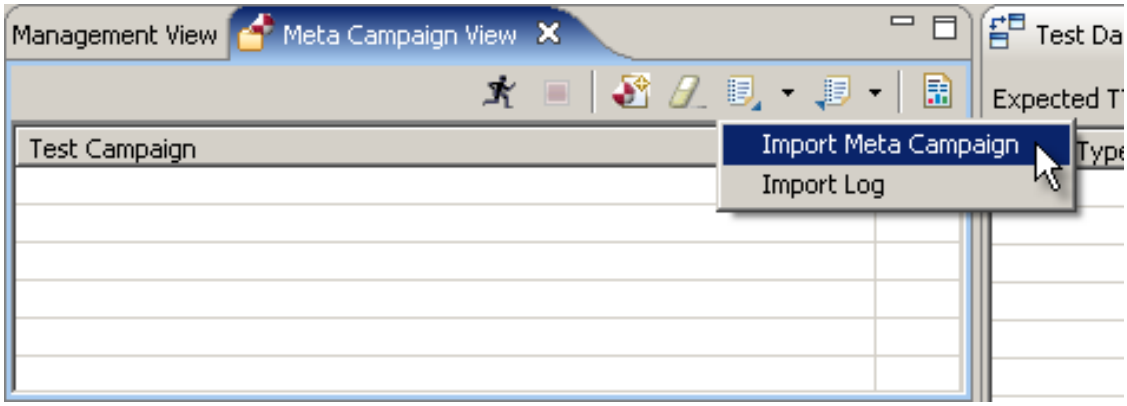
Meta Campaign Execution

Note

If you didn't create a meta campaign yet, please use the Meta Campaign Wizard to do so.

First you have to import the meta campaign.

Figure 8.32. Importing a meta campaign



The meta campaign view also enables the user to export the loaded meta campaign by choosing Export Meta Campaign from the Export Files menu of the meta campaign view's menu bar. The meta campaign then gets exported to a meta campaign file (`*.mcf`) which of course can be imported again later.

After successfully loading the meta campaign, the user is able to execute it by clicking on the Execute button in the meta campaign view's menu bar. After doing so, the campaigns get loaded and executed in the order that you defined while creating the meta campaign.

Meta Campaign Log

While executing a meta campaign, a meta campaign log gets created, containing all the single test logs that were generated while executing the single test logs. To view a log, you have to select the specific test run (test runs are diplayed as children of the respective test campaign) and choose Import Log from the context menu. Now the test campaign's log data gets displayed by the Management View and can be viewed, imported and exported. It is possible to import and export a meta campaign log file from and to the file system. The log file for a meta campaign log will be saved as a zipped file (`*.tlz`) and stores all the single log files for the executed test campaigns together with the meta campaign file that was used for the respective test run. The location of the file can be specified after clicking on the Export Files button on the meta campaign view's menu bar and selecting Export Log.

Note

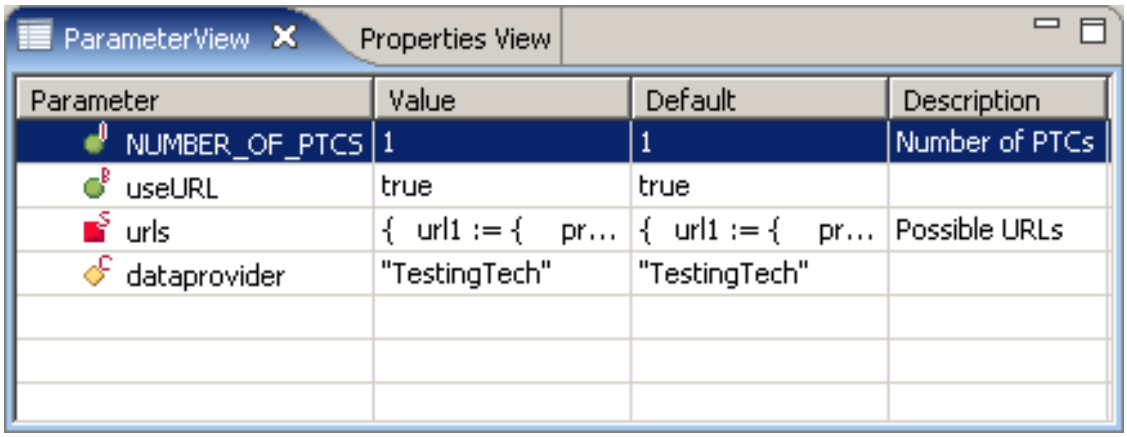
The meta campaign .tlz-file can be unzipped into the meta campaign file and the single log files of each contained test campaign.

The user can import a tlz-file to review the meta campaign execution process by using the Import Files button on the meta campaign view's menu bar and selecting Import Log.

Parameters View

The parameters view contains a table displaying the module parameters of the currently loaded campaign. It is possible to change the current value of a parameter here.

Figure 8.33. Parameters view



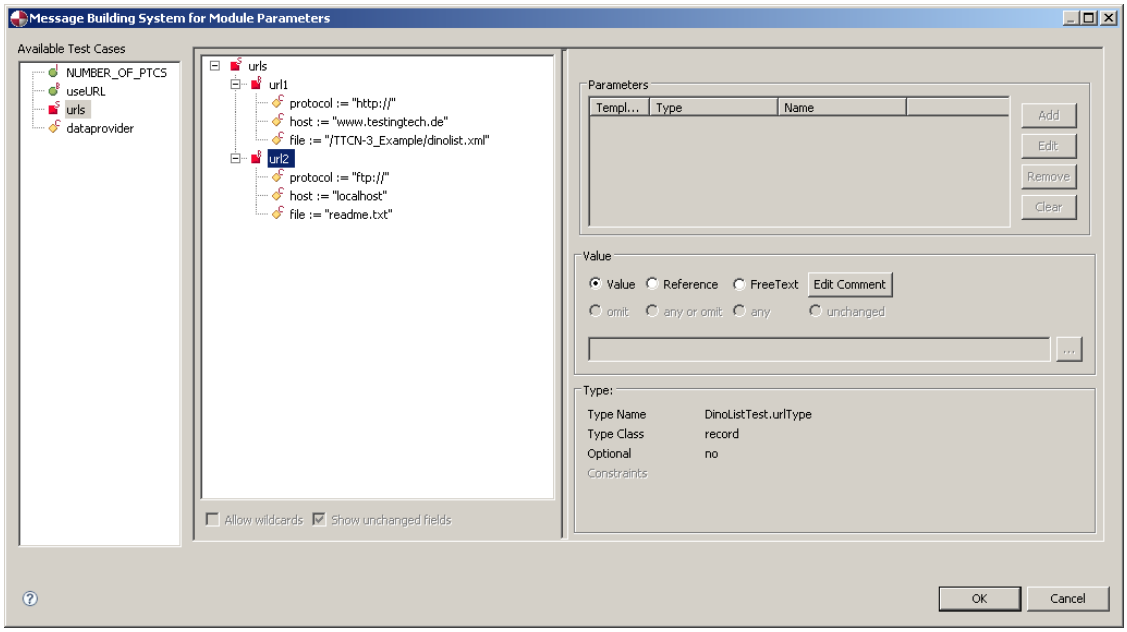
Parameters of a structured type can be edited by using the context menu entry **Open message building system**. This opens the module parameter editor, which is based on the Template Wizard, but adds a list of all module parameters at the left side. By selecting a parameter its current content gets displayed in the middle and can be edited.

Parameters of a basic type can be edited either inline by simply clicking on the value or like structured type parameters by using the module parameter editor.

Note

Currently the module parameter editor still contains the template parameters section of the Template Wizard though it is disabled. You can safely ignore that section.

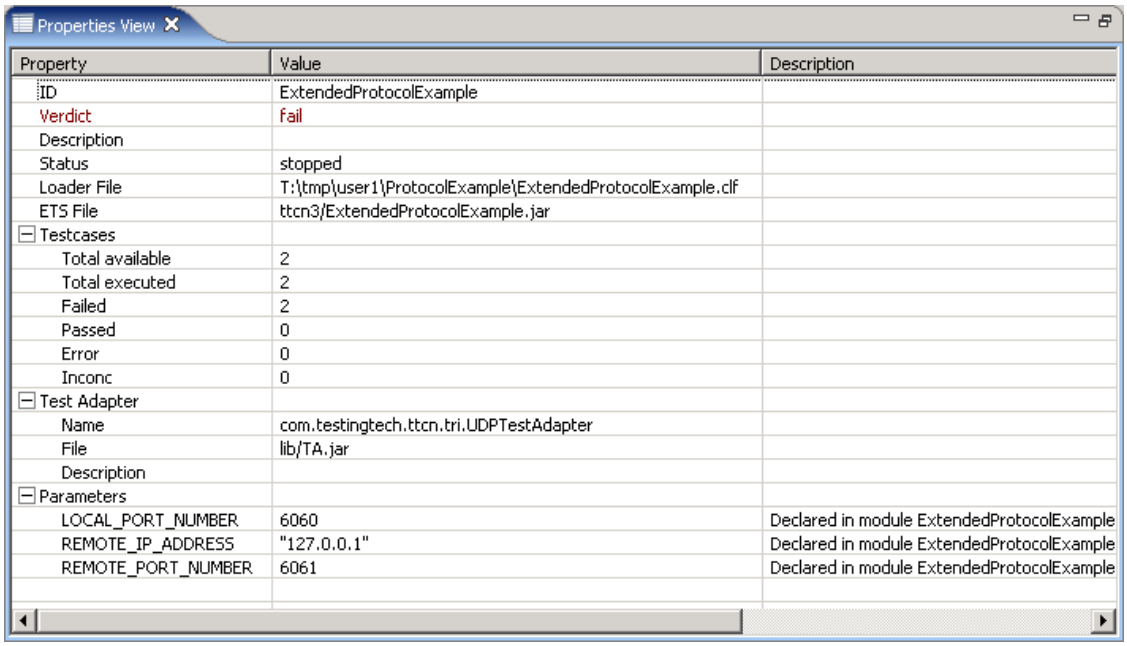
Figure 8.34. Module Parameter Editor



If values of module parameters have been changed, the user gets asked whether the campaign loader file should be saved before loading a new module or exiting TTman.

Properties View

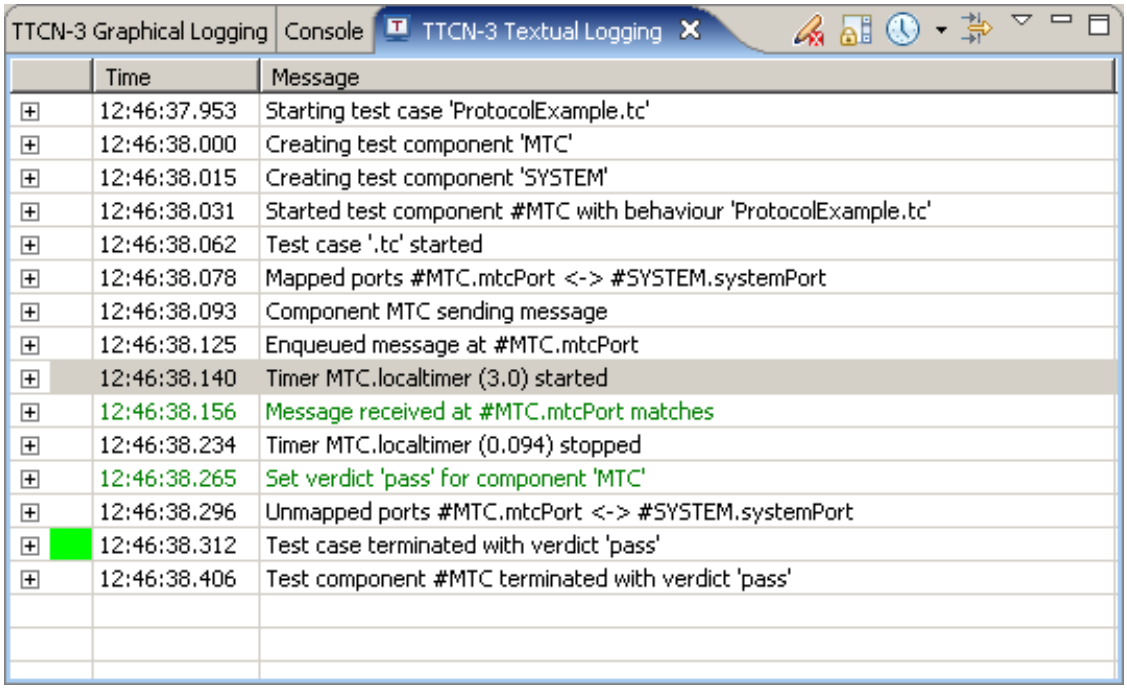
Figure 8.35. Properties view



The properties view (Figure 8.35, “Properties view”) contains a table displaying the properties of each selected entry in the management view. All properties are continually updated at all stages of the test configuration and during the execution process.

Textual Logging View

Figure 8.36. Logging View



As depicted in Figure 8.36, “Logging View”, the logging view displays all the logs and traces collected during the test execution. This can be performed on-line or off-line, depending on the user's preferences.

Events can be copied in a textual form to the system clipboard by using the event's context menu. The command Copy Event to Clipboard formats the event details in a human readable way, while Copy Event as XML to Clipboard provides the plain TLI XML data. It is also possible to export the whole textual log in human readable form (see the section called “Test Log”).

For certain types of log events, the user can view details of the data contained in those events by double-click on their entry in the logging view. If the log event corresponds to a matching or non-matching event upon receiving a message, both the expected template and the actually received data structure are displayed in the data view. The data view, if not yet opened, will be opened automatically.

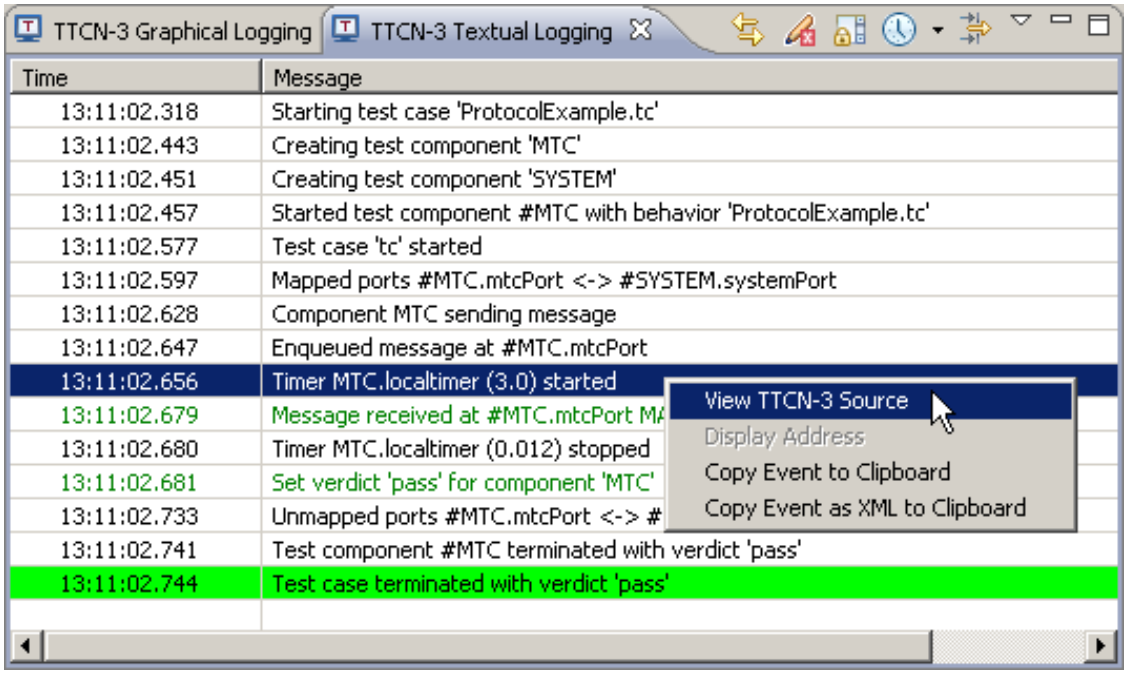
Note

Please mark that specific TLI log events are currently not supported by TTworkbench (see Language Features for details).

The textual logging view supports the View TTCN-3 Source feature. If there is an event with a corresponding statement in the TTCN-3 code it is possible to go directly from the logging view to its declaration. There are three ways to perform this action:

- hold SHIFT key pressed and left click on the logging event
- right click on the logging event and choose "View TTCN-3 Source"
- select the logging event and hit F3 on your keyboard

Figure 8.37. View TTCN-3 Source



Time Stamps

The logging view provides three options to display time stamps of received log events. The options are:

- Time of day (default)
- Seconds since previous event
- Seconds since beginning test case

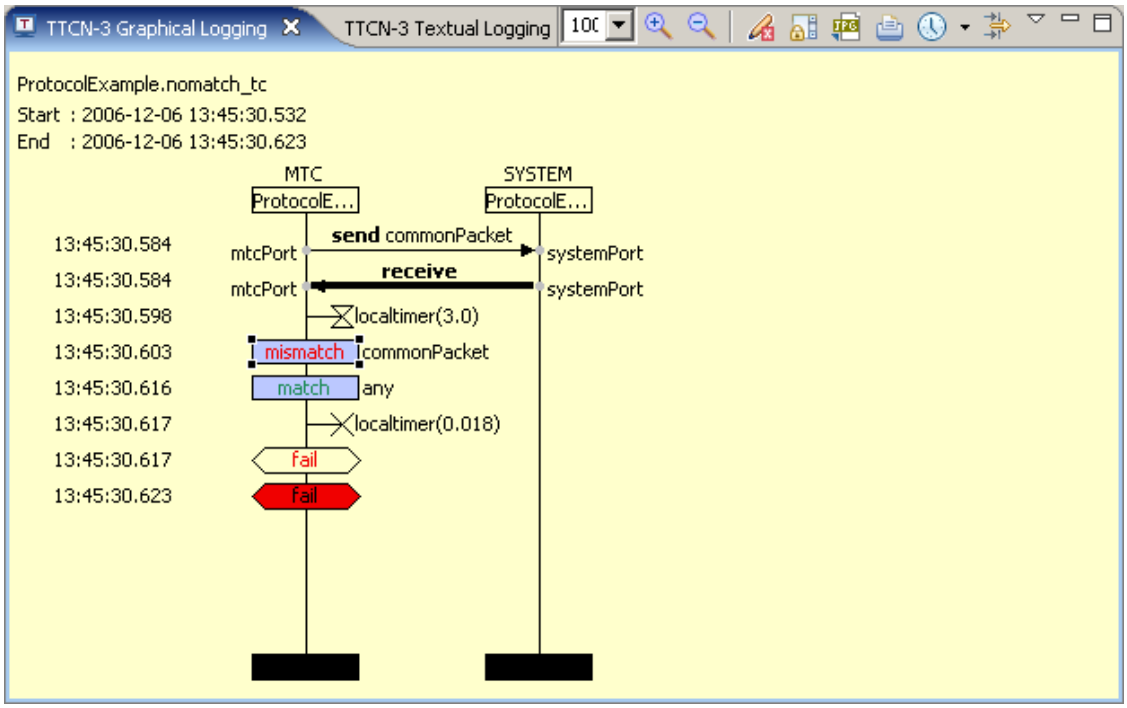
Using Filters

The logging view provides a set of mechanisms for filtering the logs, in order to limit the amount of log output. The following criteria can be used for the filters:

- Component name
- Log event type

Graphical Logging View

Figure 8.38. Graphical Logging View



In addition to the textual logging view (Textual Logging View), TTman supports a graphical logging view. It displays the relevant logs and traces collected during the test execution as symbols in a clearly arranged diagram. The test case name can be found in the top left corner of the graphical logging view.

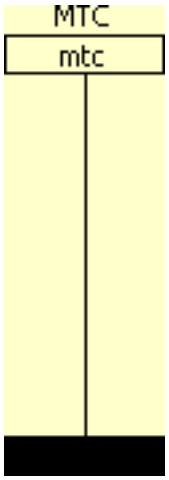
Messages and Match/Mismatch symbols will be selected together to know easily which elements are belong to each other.

Messages and Match/Mismatch symbols can be selected to get more details, listed in the Data View (the section called “Data View”).

Symbols

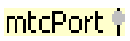
The graphical logging view provides the following symbols:

- Instance: the ID name is located above the instance, the component type beneath



- Message: the port operation and message type is located above the message

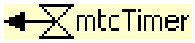
- Port name associated with the sending or receiving message



- Start Timer: starting timer with timer name and duration (in sec)

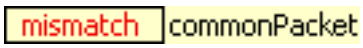
- Stop Timer: stopped timer with timer name and stop time (in sec)

- Timeout Timer: timeout of a timer with timer name



- Match: a previously received message matches; data type is given after it

- Mismatch: a previously received message mismatches; data type is given after it



- Log: a log comment

- Log Event: can be pass, fail, inconc, none or error

- Verdict is pass

- Verdict is fail

- Verdict is inconclusive

- Verdict is none

- Verdict is error; a log statement with the reason is given above the verdict error

- SUT action

Zooming

The graphical view can be zoomed in and out by using the zoom buttons 
and 
or alternatively by selecting a value between 50% and 400%.

Export Graphic as Image

It is possible to export the graphical logging view as a JPG image. Therefor left-click on the 
icon in the graphical logging view and enter a file name.

Print

It is also possible to print the graphical logging view. Therefor left-click on the print icon 
in the graphical logging view.

Note

Printing in Eclipse under Linux is a known problem (printing in Eclipse on GTK has not been implemented yet).

Time Stamps

The graphical logging view provides four options to display time stamps of received log events. The available formats are:

- None
- Time of day (default)
- Seconds since previous event
- Seconds since beginning test case

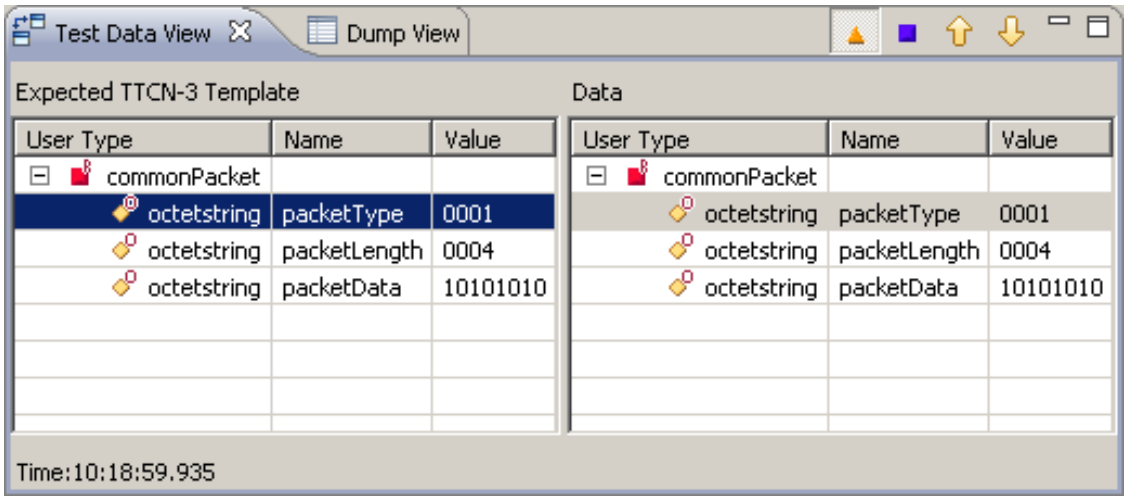
Using Filters

The graphical logging view provides the filtering of logs, in order to limit the displayed symbols.

Pressing the filter button 
will popup a selection menu, where to filter the logging events and to enable or disable the filter mechanism.

Data View

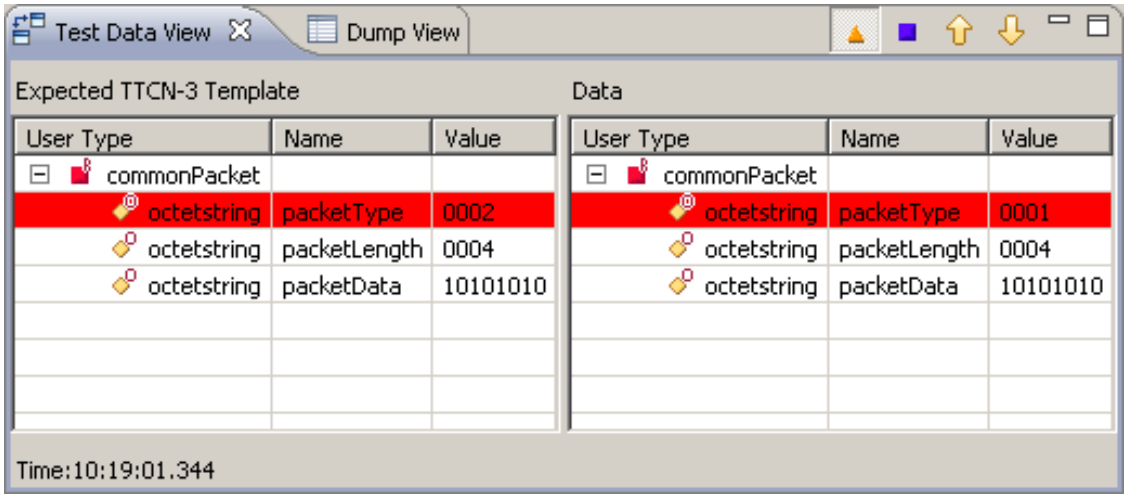
Figure 8.39. Data View



The Data View contains two table trees. The right side shows the received or transmitted value, structured as a tree according to the value's type. The left side contains the data template given in the corresponding receive operation or nothing, if it is a send operation. When one or more lines on the template table tree are selected, the corresponding data lines at the received data tree get marked, too, and vice versa. By using the button, you can switch the User Type columns on or off. The button allows you to display an additional column that shows integer values in hexadecimal format.

The data view can be used to compare the received data with the expected template. Data that does not match the expected template, is highlighted as shown in Figure 8.40, “ The received message does not match the given template ”.

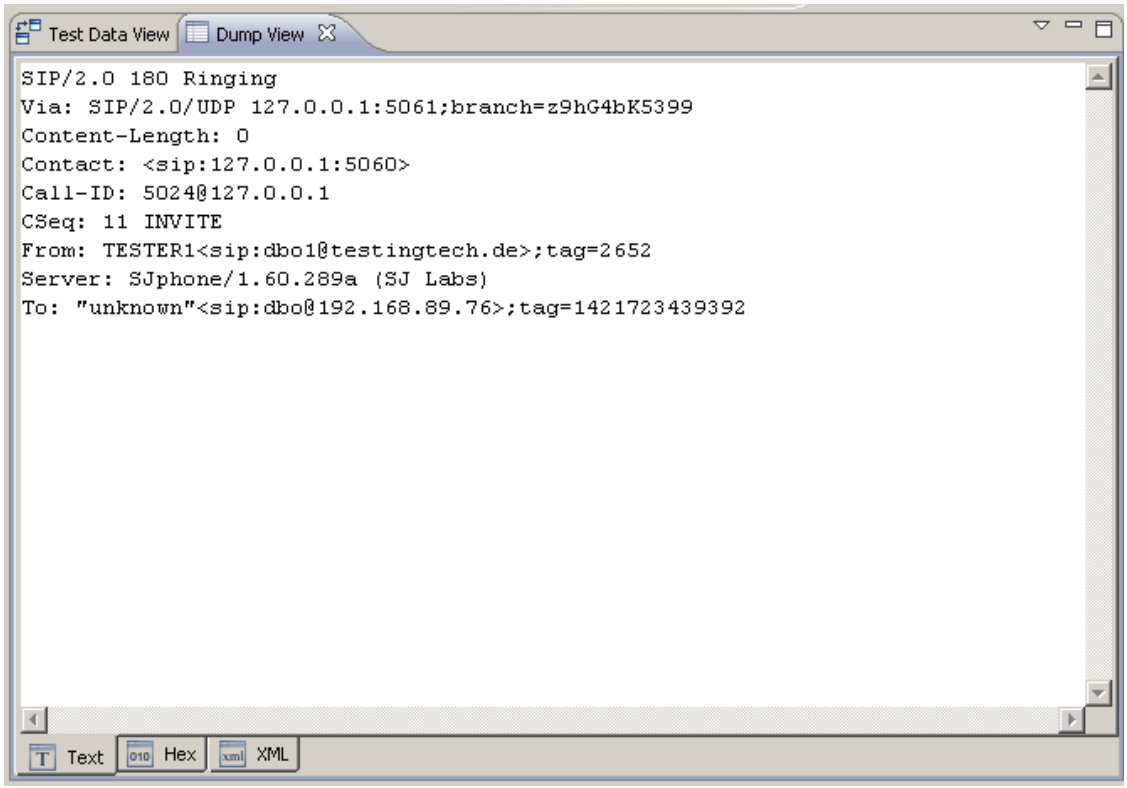
Figure 8.40. The received message does not match the given template



Dump View

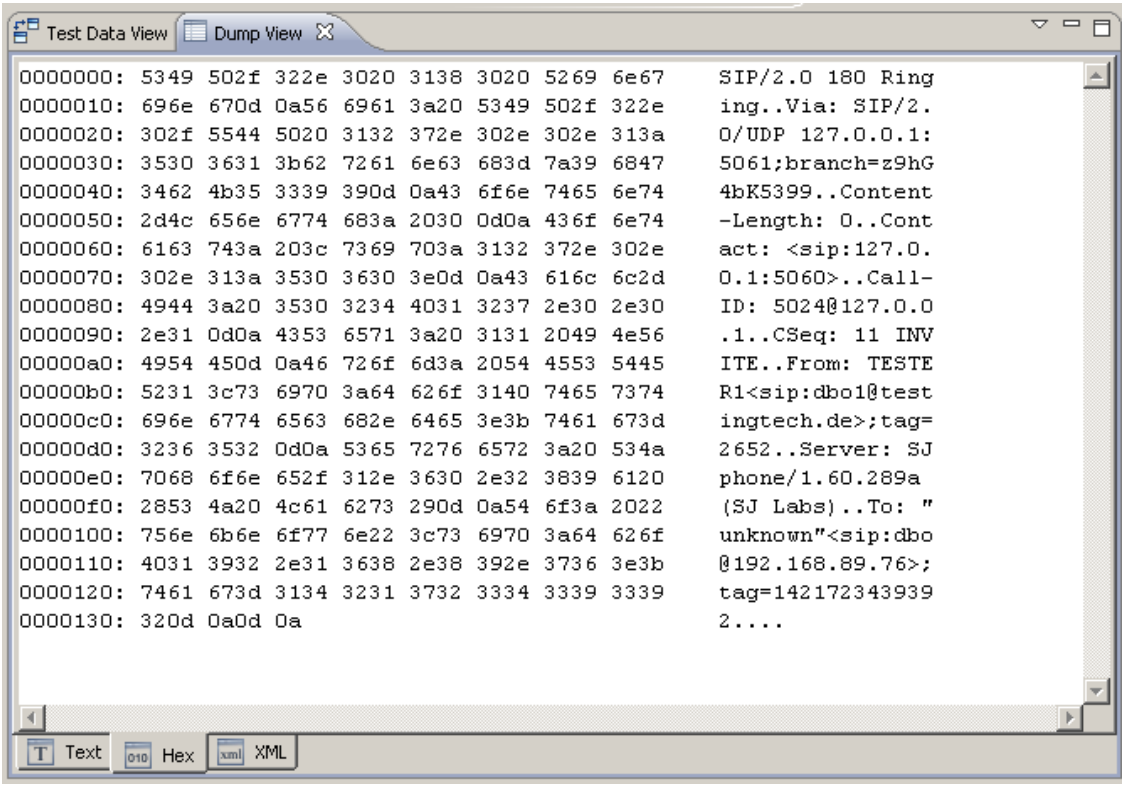
The dump view can be used to display outgoing and incoming data as plain text, as hex dump, as XML or other custom implementation. It is activated on selection of a send or enqueue event in the logging view. By default, the sent or received data will be displayed as plain text Figure 8.41, “Dump view interpreting the input as text”. The user can switch between the different presentations formats by selecting another tab.

Figure 8.41. Dump view interpreting the input as text



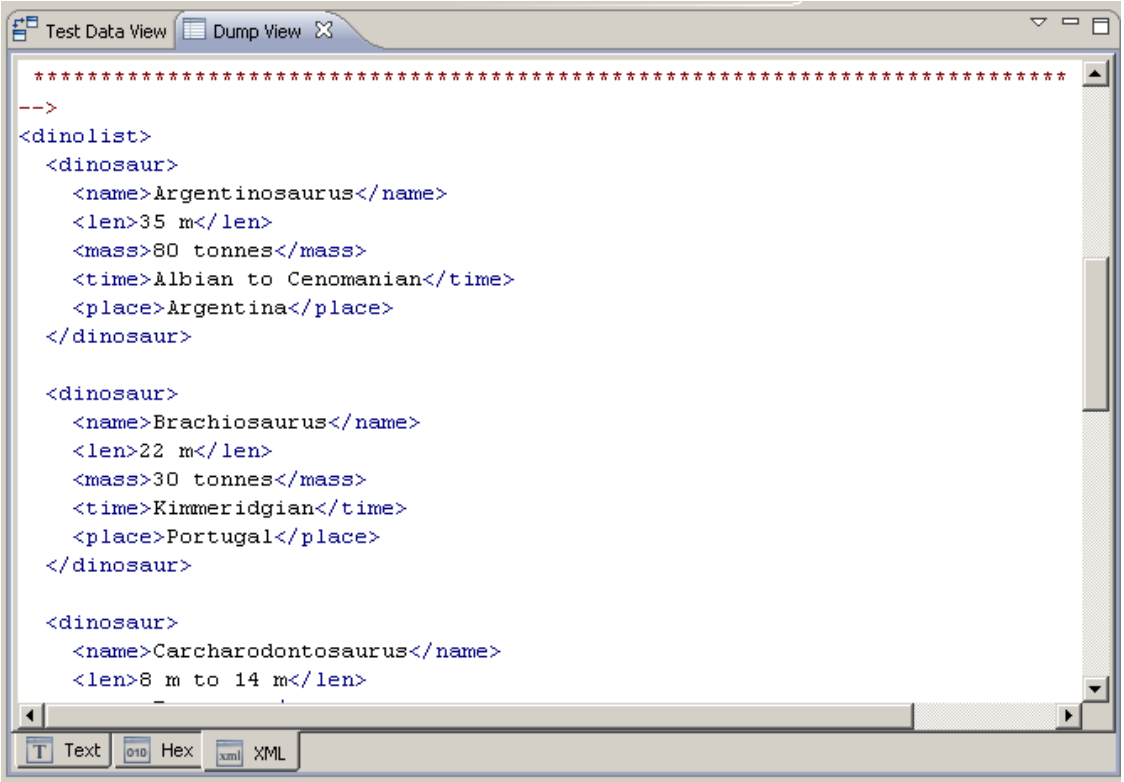
The hex mode is shown in Figure 8.42, “Dump view interpreting the input as hex” .

Figure 8.42. Dump view interpreting the input as hex



Another useful mode for XML based communication protocols is shown in the next figureFigure 8.43, “The Dump view interpreting the input as formatted XML with highlighting.” .

Figure 8.43. The Dump view interpreting the input as formatted XML with highlighting.



Preferences

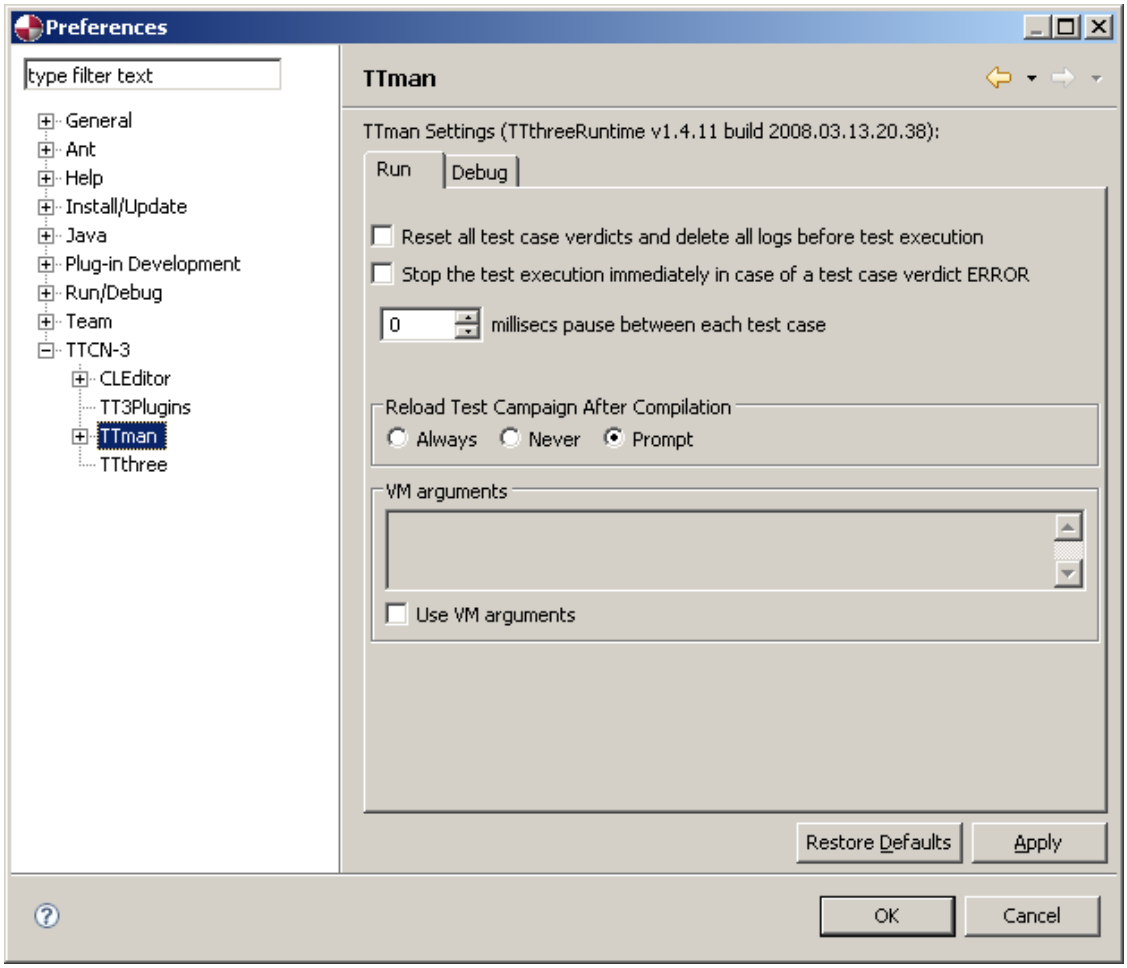
TTman

TTman provides a preference page for configuring different properties. These properties include:

- Behavior of test campaign execution
- Support of online logging (during the test execution logging will be displayed immediately in the TTCN-3 logging view) and graphical logging
- Debug level of TTman
- Information to be included in test reports
- TTthree plugins
- Mail settings to send test report as e-mail

Run

Figure 8.44. Complete test case execution is set by default



It is possible to reset all test case verdicts before test execution by checking the respective box.

If you want the test campaign execution to stop automatically after a test case has been completed with verdict error, you have to select the checkbox in the run tab. The default behavior is the execution of the complete test campaign independently of any test case results Figure 8.44, “Complete test case execution is set by default”.

If you want the execution to pause between each test case, the duration of the pause can be determined on this tab.

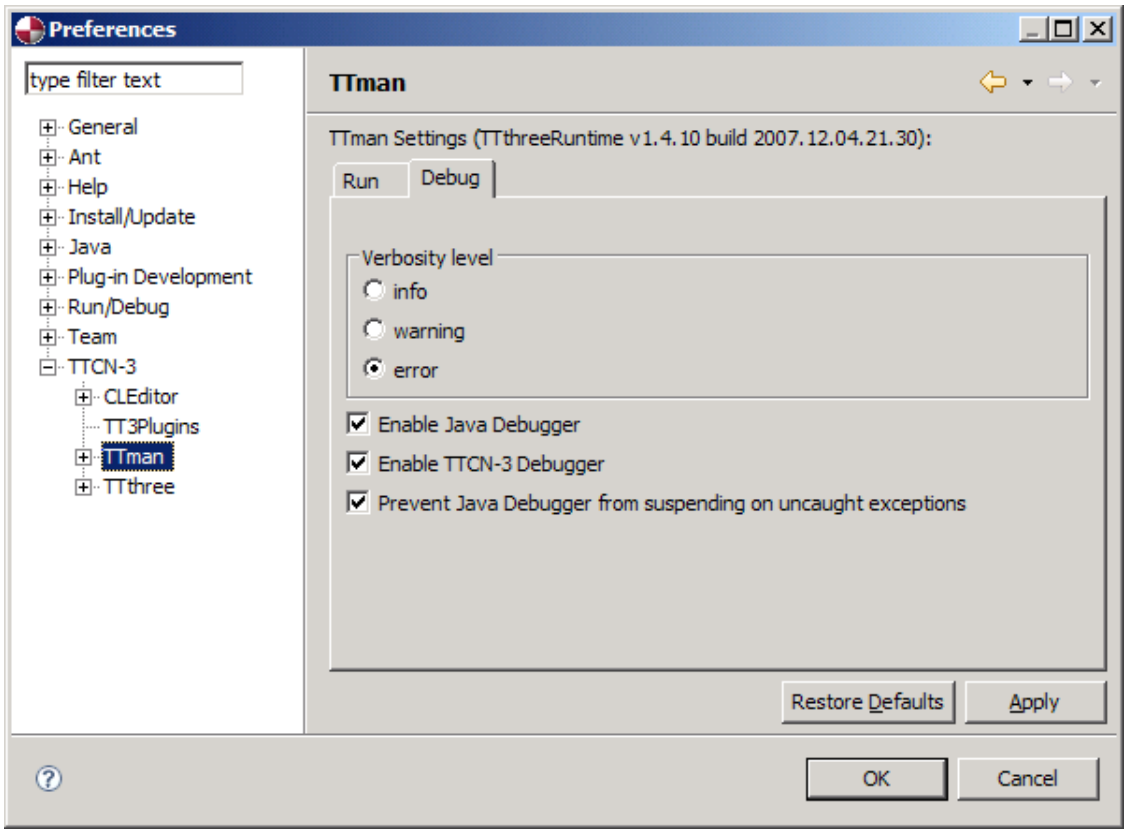
In the Reload Test Campaign After Compilation section, you can define how TTman behaves when you compile a test suite that is currently loaded. By setting this option on Prompt, you get asked by TTworkbench how to behave everytime this situation occurs.

The VM arguments editor allows you to set command line arguments that you need to be set in the Java Virtual Machine while executing test cases. The Use VM arguments checkbox allows you to easily enable or disable your personal VM settings.

Debug

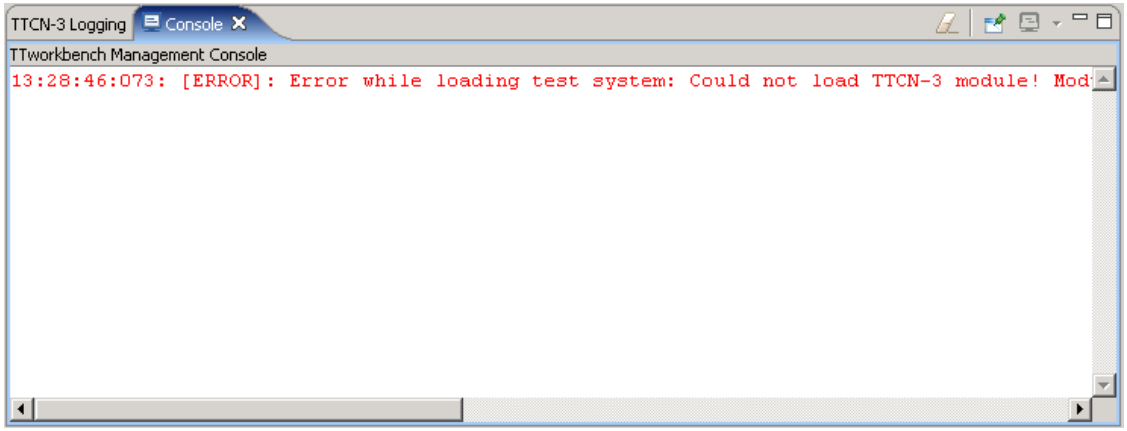
The user can specify different debug levels for TTman. Defined levels are debug, info, warning, and error. By default, the error level is selected Figure 8.45, “Error debug level is set by default”.

Figure 8.45. Error debug level is set by default



Thus, only errors will be displayed in the output Console Figure 8.46, “An error displayed in the output Console”.

Figure 8.46. An error displayed in the output Console



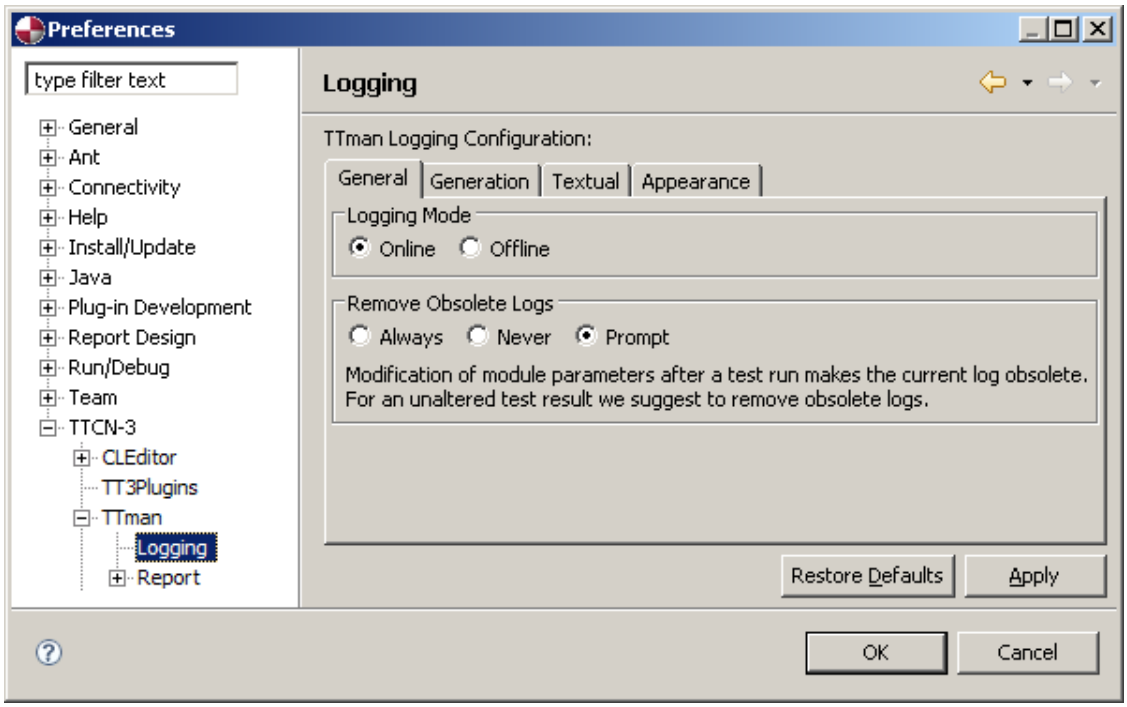
The other entries on this page belong to TTdebug. Their meaning is explained in the TTdebug preferences section.

Logging

General

It is possible to disable the online logging for a better performance. To do this, please use the respective radio button.

Figure 8.47. Online logging is supported by default



Generation

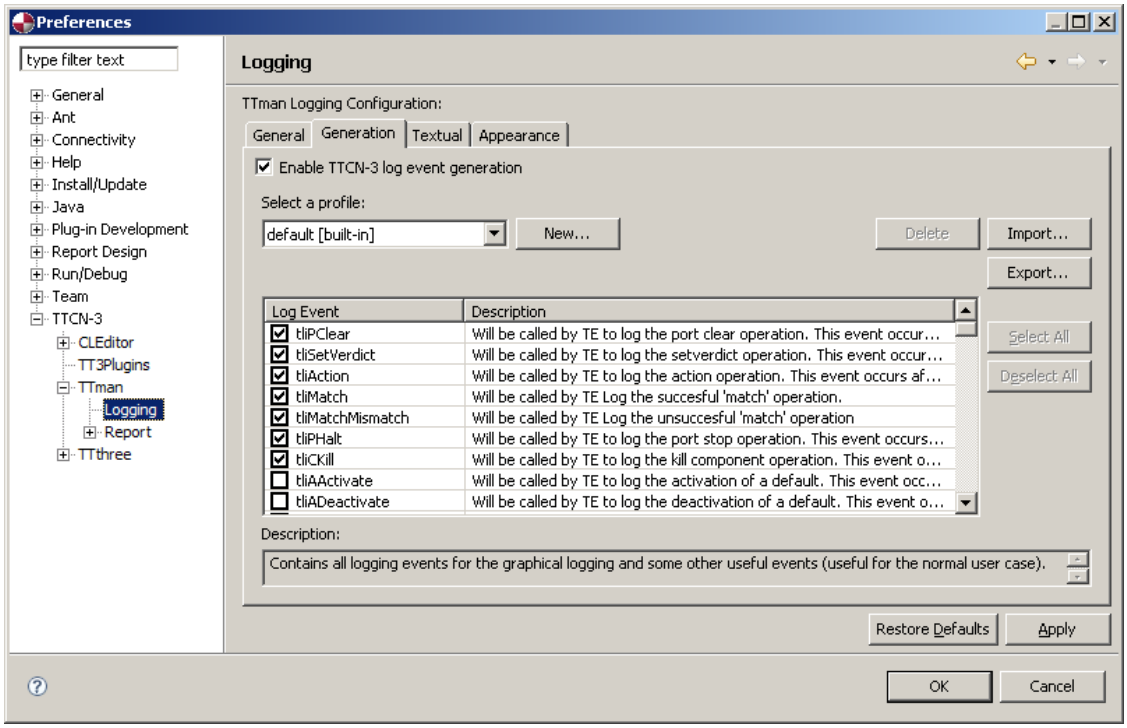
By default, TTCN-3 logging generation is enabled. The user is able to disable this to increase the performance.

Note

If TTCN-3 logging generation is disabled no logging events will be created.

To increase the performance during the runtime without loosing needed log information it is possible to define only distinct log events that should be created during the runtime. This can be defined in the Generation tab of the Logging properties page. There, all log events that should be created or not can be defined by enabling or disabling the checkboxes.

Figure 8.48. TTman preferences logging generation



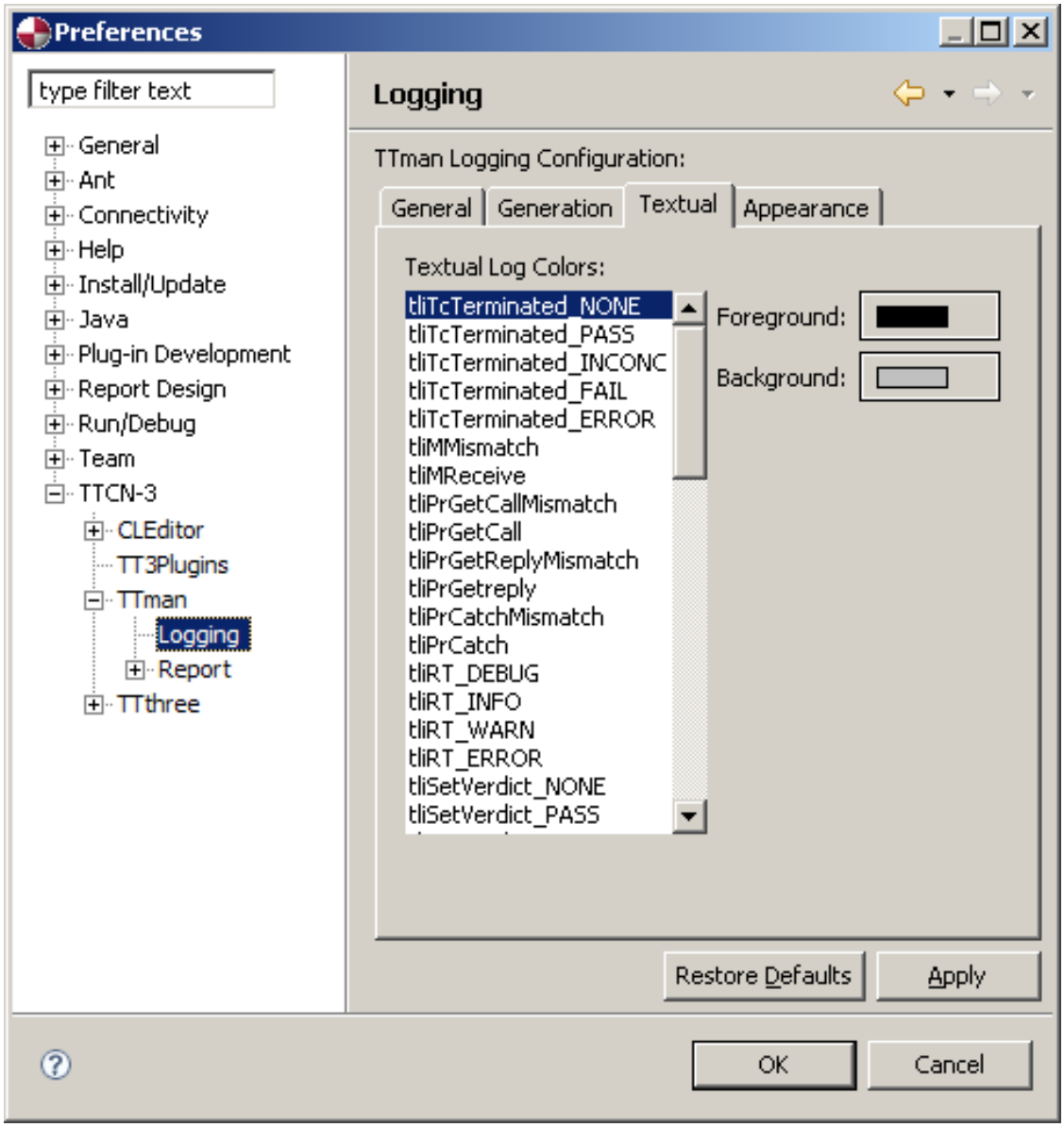
It is possible to export a defined logging profile for using it e.g. in the command line mode of TTman. Therefore, by pressing the export button it is possible to define the directory and file name of the properties to store.

It is also possible to import an existing logging profile by pressing the import button.

Textual

This tab allows you for each logging event to set the foreground and background colors in which it gets displayed in the Textual Logging View.

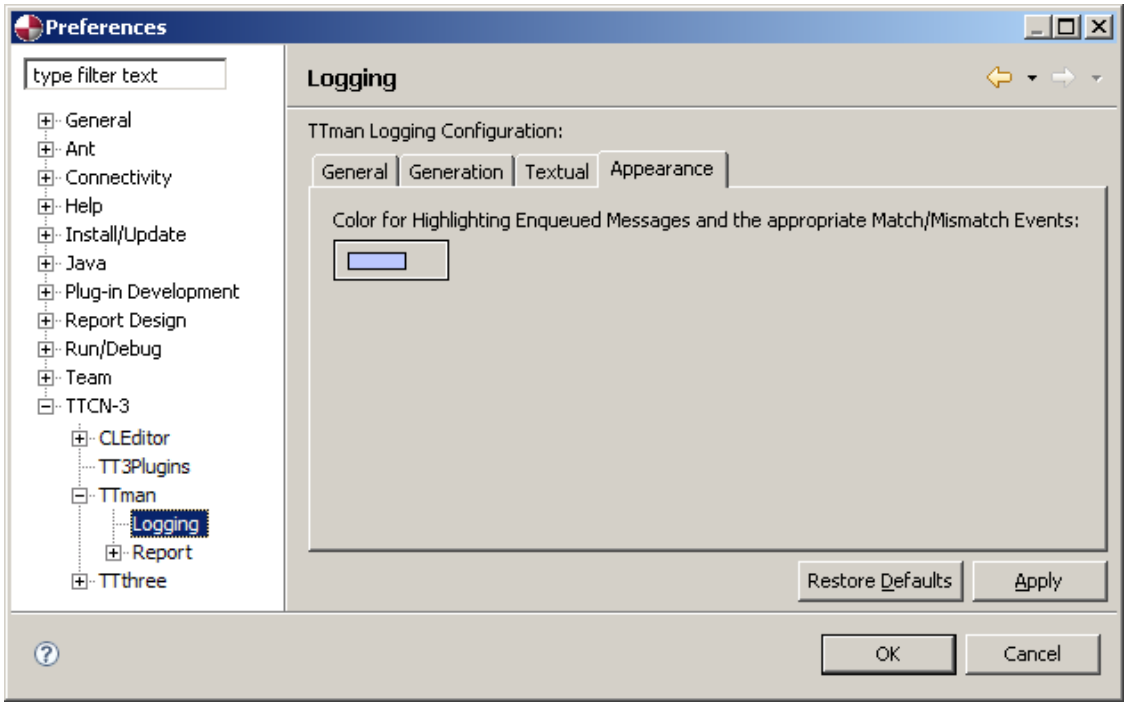
Figure 8.49. TTman preferences for textual logging



Appearance

This tab allows to set appearance settings.

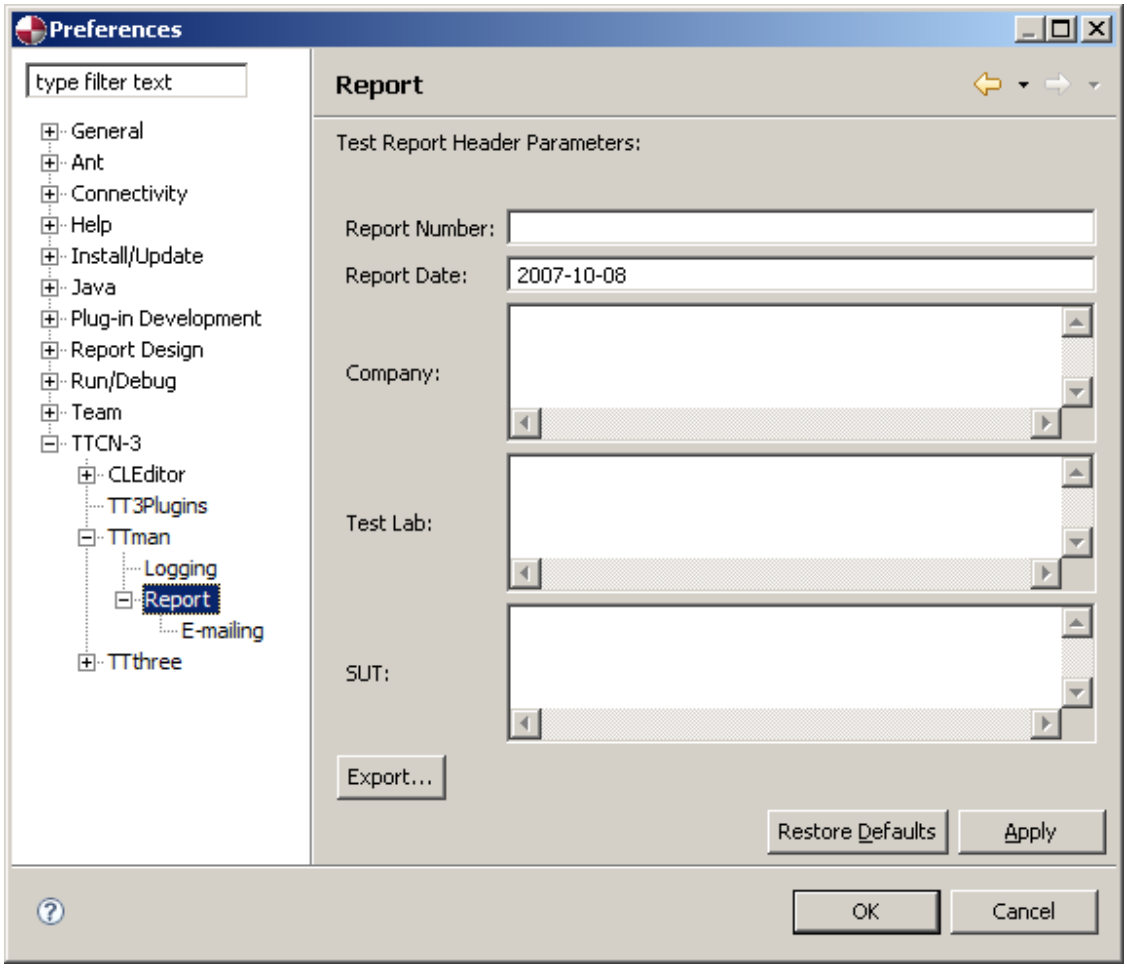
Figure 8.50. TTman appearance settings



Report

In this section, you can set user and test dependent information to be included in test reports.

Figure 8.51. User and test dependent information for test reports



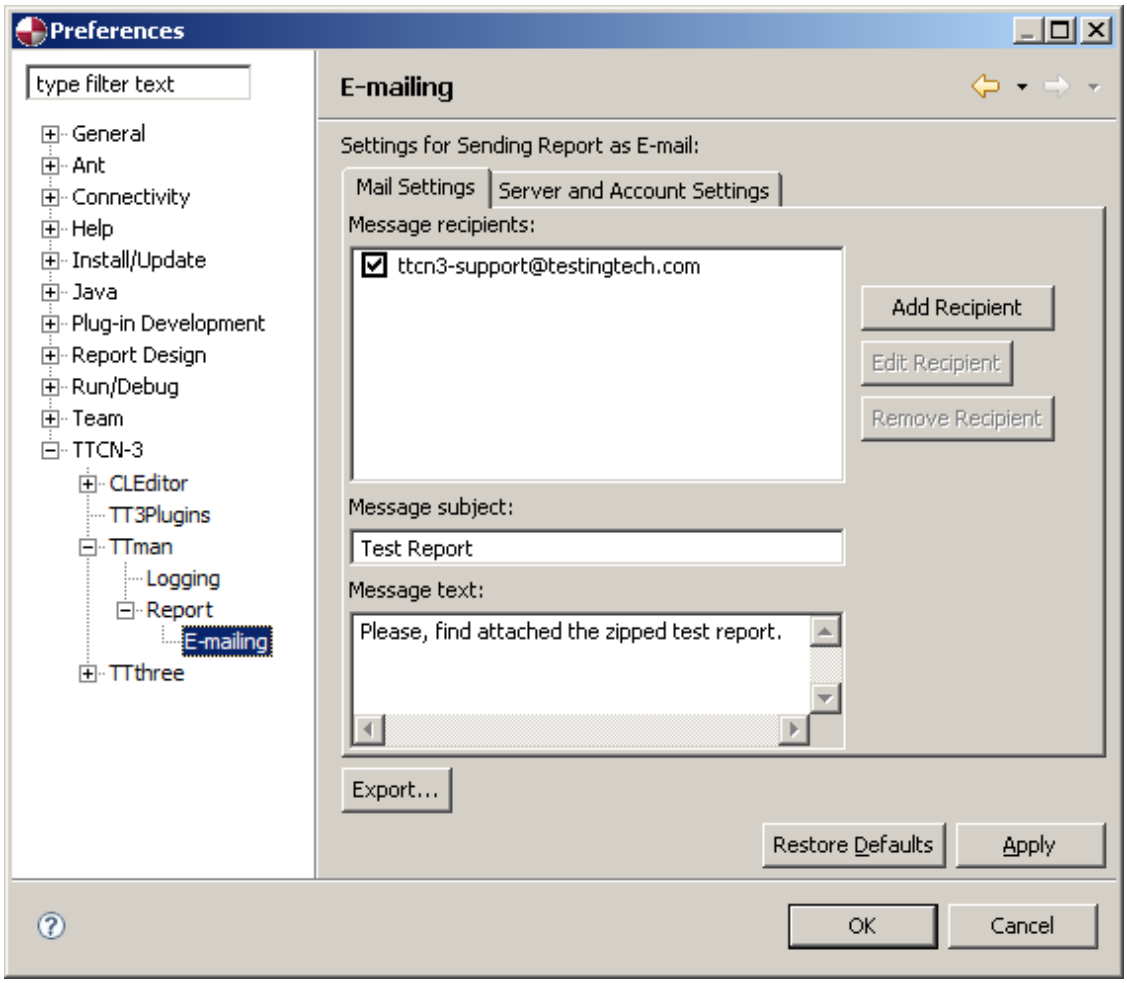
E-mailing

The properties of the feature to send test reports as e-mails can be configured via the TTman Emailing preference page.

Mail Settings

By using the Mail Settings tab, the recipients of the test report e-mail, the subject of the message, as well as the text in the message body can be configured.

Figure 8.52. Mail Settings

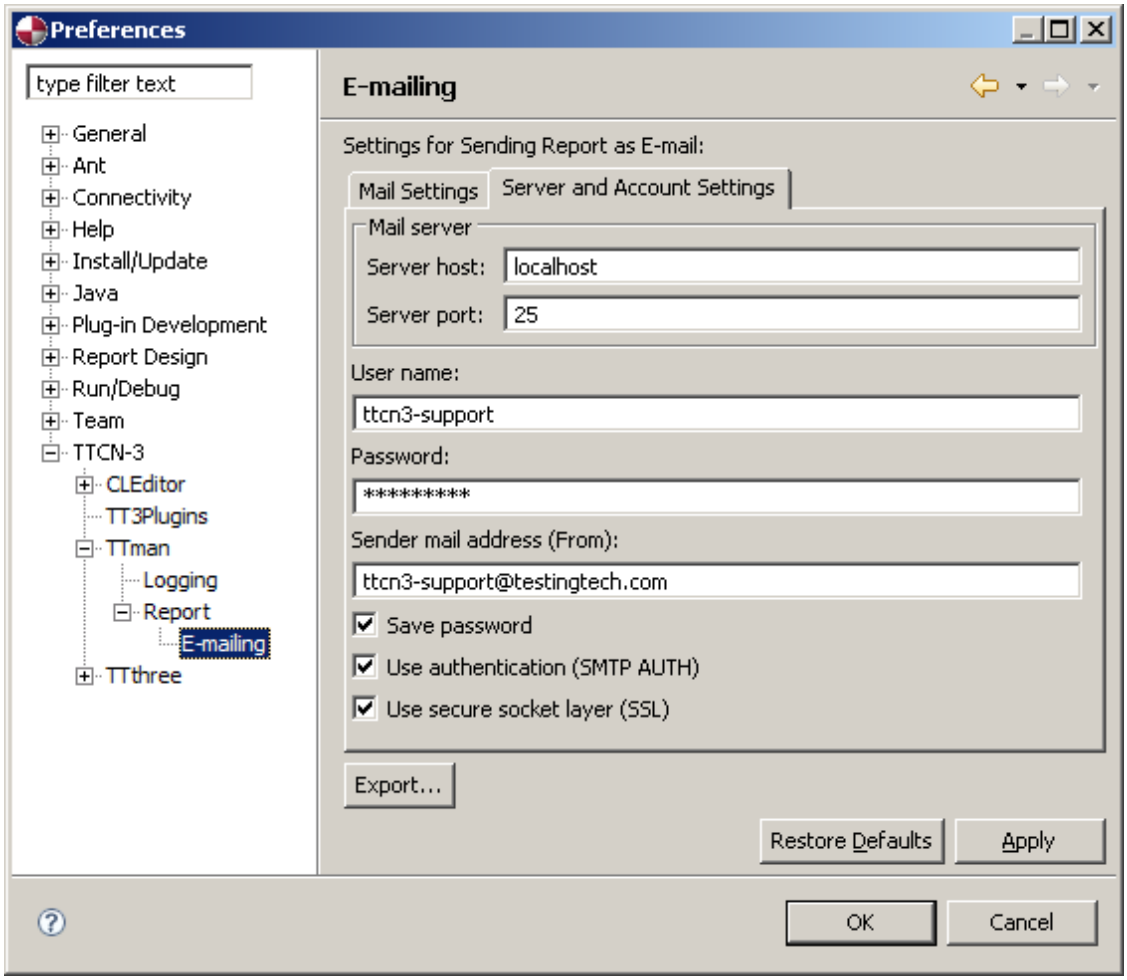


Server and Account Settings

In the Server and Account Settings tab the mail server and the mail sender can be configured. If the used mail server requires authentication the SMTP Auth option has to be enabled and a user name has to be configured. The user's password can be optionally set. If no password has been configured in the preferences, it can be entered later in interactive mode. Setting the password in the preferences, the user has also the option to store the password. It will be saved DES engrypted in the preference store. For the encryption, additionally a key will be generated and stored in the user's home directory in a file named ttwbExtReport.keystore. This key will remain after closing TTworkbench and reused in further TTworkbench sessions.

The Export button can be used to export all mail settings into a file. If using the TTman Console Manager (see the section called “Command-line Mode”), a mail settings file is required when sending of test report as e-mail option has been set.

Figure 8.53. Server and Account Settings



Command-line Mode

TTman can also be executed via command-line mode using the scripts located in TTworkbench installation directory:

- For Linux: TTman.sh
- For Windows: TTman.bat

The logging from the execution using the command line tool is automatically saved in a .tlz file located in the same directory as the executed .clf. You can then load this file into TTman and analyze the execution log.

Additionally, the command line tool can be used to generate test reports directly from log files (.tlz). Therefore the report specific options have to be used and a log file has to be passed instead of a campaign loader file.

Command-line synopsis:

`TTman` [*`options`*] *`loader_file`* | *`log_file`*

options	Command-line options. TTman Options may be in any order. For every option a short form (with one dash), and a long form (with double dash) exists.
loader_file	Module Loader File (`*.mlf`) or Campaign Loader File (`*.clf`).
log_file	Log File (`*.tlz`).

The following TTman options are available:

-c, *--control*	Execute the control part of this test suite.
-d, *--disable-logging*	Disable the TTCN-3 logging generation.
-e, *--error*	If set the execution will be stopped in case of a test case error. In case the option is not used, execution will continue.
-h, *--help*	Get help information on command line options and exit.
-l, *--log <log_dir>*	Define the destination folder where to store the log file; in case the option is not used, the log file will be stored in the same directory the given loader file is located in. It is possible to use absolute as well as relative paths.
--loop <loop_number>	Define how many times all test cases contained in the loader file should be executed.
-M, *--mail <mail_settings_file>*	Send a generated test report as e-mail (pass a mail settings file).
-p, *--logging-properties <property-file>*	Pass a configuration file to define which log events should be created during runtime (can be used to increase the performance).
-r, *--report <report_format>*	Create a test report from the executed test suite. The output format is either HTML (*html*), PDF (*pdf*), Excel (*excel*), or Word (*word*). The Excel format is an XML file that can be opened with Microsoft Office Excel 2003 and later versions as well as with OpenOffice 2.0. The Word format is also an XML file that can be opened with Microsoft Office Word 2003 and later versions.
-P, *--tt3PluginDir <tt3-plugin-dir>*	Pass the directory where possibly additional TT3 plugins are located.
-v, *--version*	Get version information of the currently installed TTman version and exit.
-w, *--wait <delay>*	Define a delay (in seconds) between the execution of two test cases.

Exit codes:

The exit code returned by the TTman command-line execution is the most severe verdict of the test run.

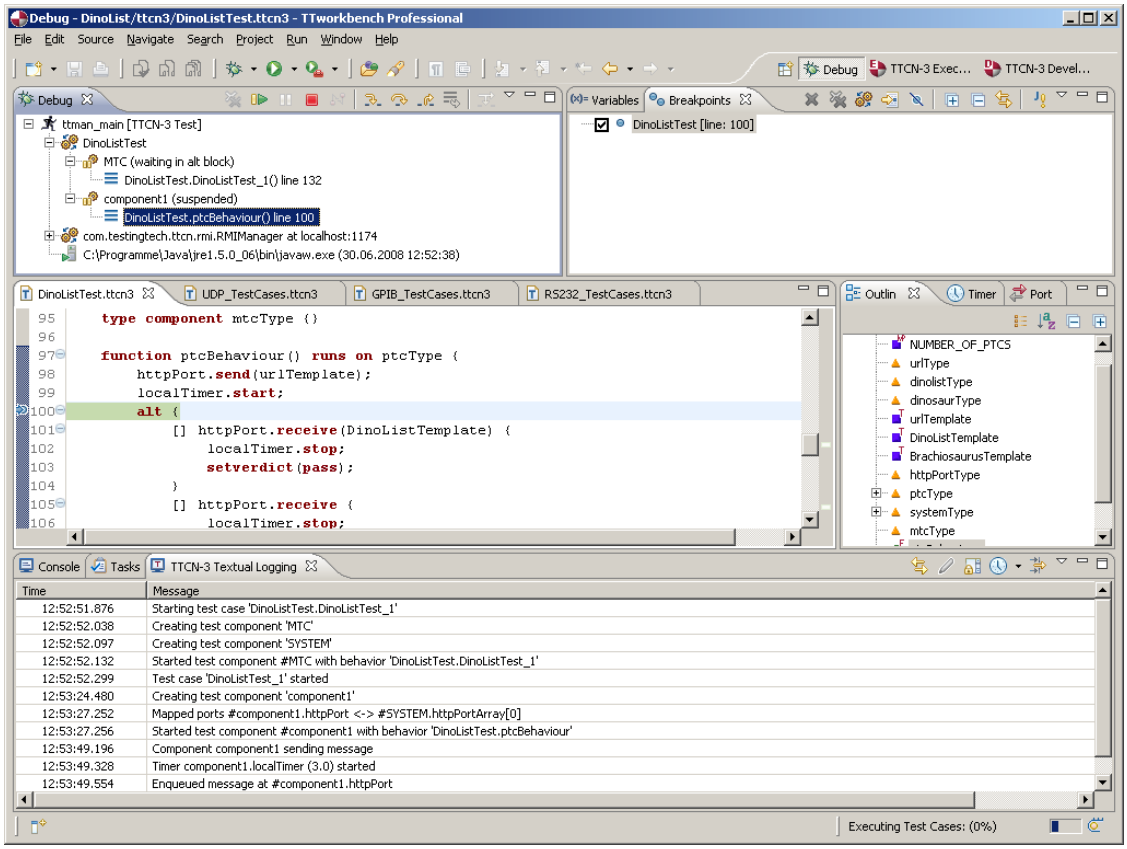
110	None
111	Pass
112	Inconclusive

113	Fail
114	Error

Chapter 9. Using TTworkbench TTdebug

With TTdebug the user obtains the possibility to execute his test cases instruction by instruction, view states and variables and manipulate the execution. It allows fast and efficient tracking of bugs and so eases to develop reliable test suites. TTdebug uses the eclipse debugger framework which provides standardized views, controls and extensive layout adaptability.

Figure 9.1. TTdebug



Overview

TTdebug provides the following functions:

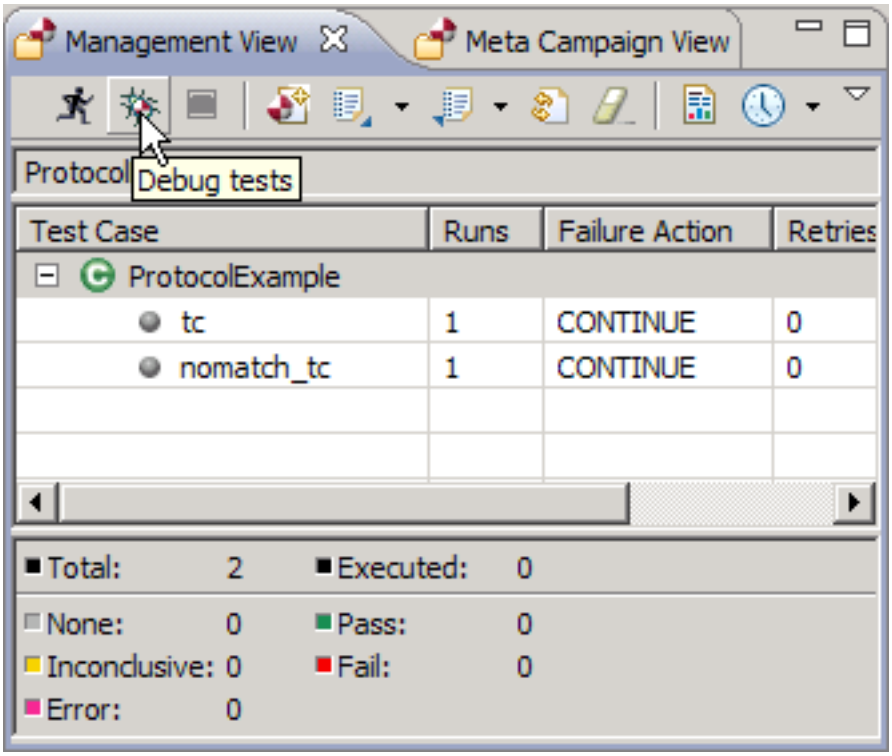
- GUI based setting/removal/disabling of breakpoints (see Breakpoints)
- Manually suspending/resuming a running testsuite (see Debug View)
- Stepping through a suspended test suite by using the common debugger functions "step into", "step over" and "step return" (see Debug View)
- Viewing the status and the stack traces of multiple components (see Debug View)
- Viewing and editing the content of local and component variables and local parameters (see Variables View)

- Viewing the status of timers and manually triggering timeouts (see Timers View)
- Viewing the content of port queues and manipulating the order of received messages (see Port Queue View).
- Transparent debugging of TTCN-3 and Java-based test adapters and codecs at the same time (see Debugging Java)

How to start debugging a test suite

1. Rebuild the test suite you wish to debug if it was compiled before installing TTdebug, otherwise the debugger will not work (see "Perform the Compilation" in the TTthree chapter).
2. Set breakpoints (see "Breakpoints").
3. To load the test suite and select the test cases that you would like to debug, proceed like you would do for starting a standard test session (see "Management View" in the TTman chapter). After that, use the Debug button or the according context menu entry to start the debugging session.

Figure 9.2. Debug button



Note

If the Debug button is disabled after loading the test campaign, maybe TTdebug is completely disabled. The according setting can be found in the TTdebug preferences.

Note

Choosing between the Execute and the Debug button does not affect the Java debugger. Its status only depends on the according option in the preferences.

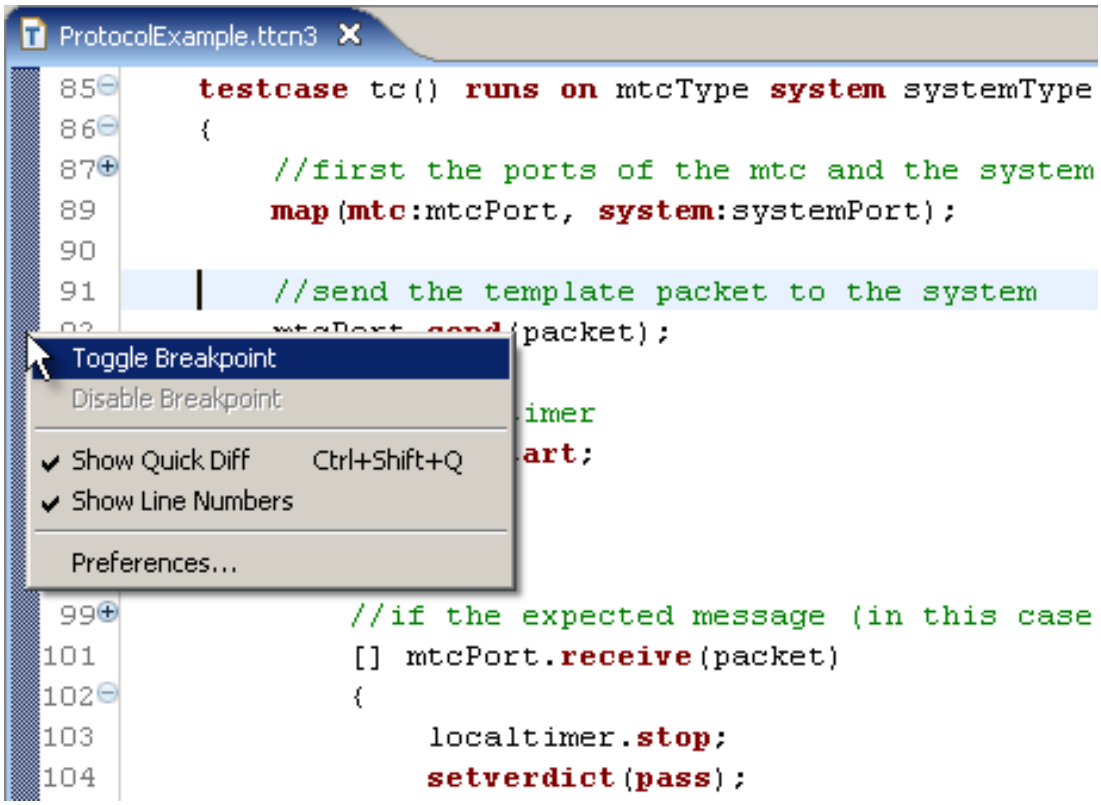
Breakpoints

Setting breakpoints

To set breakpoints you first have to open the source file you plan to debug. Then you have three options:

- Double click on the vertical ruler at the left side of CL Editor in the line where you want to place the breakpoint.
- Click with the right mouse button on the vertical ruler at the left side of CL Editor. In the context menu that now shows up, select **Toggle Breakpoint**.

Figure 9.3. Toggling a breakpoint by using the context menu of the vertical ruler



- Switch to the Debug Perspective. Place the cursor in the line that shall get a breakpoint, then select **Run > Toggle Line Breakpoint** in the TTworkbench menu bar.

All three methods do not explicitly set but toggle breakpoints, so they are also suitable for removing them. However, breakpoint removal can also be done in the Breakpoints View.

Note

Not every line of a TTCN-3 source file is valid for setting breakpoints. To prevent setting breakpoint in a line where TTdebug would never stop, newly set breakpoints get checked for validity and moved to the next proper position for a breakpoint, if needed.

Note

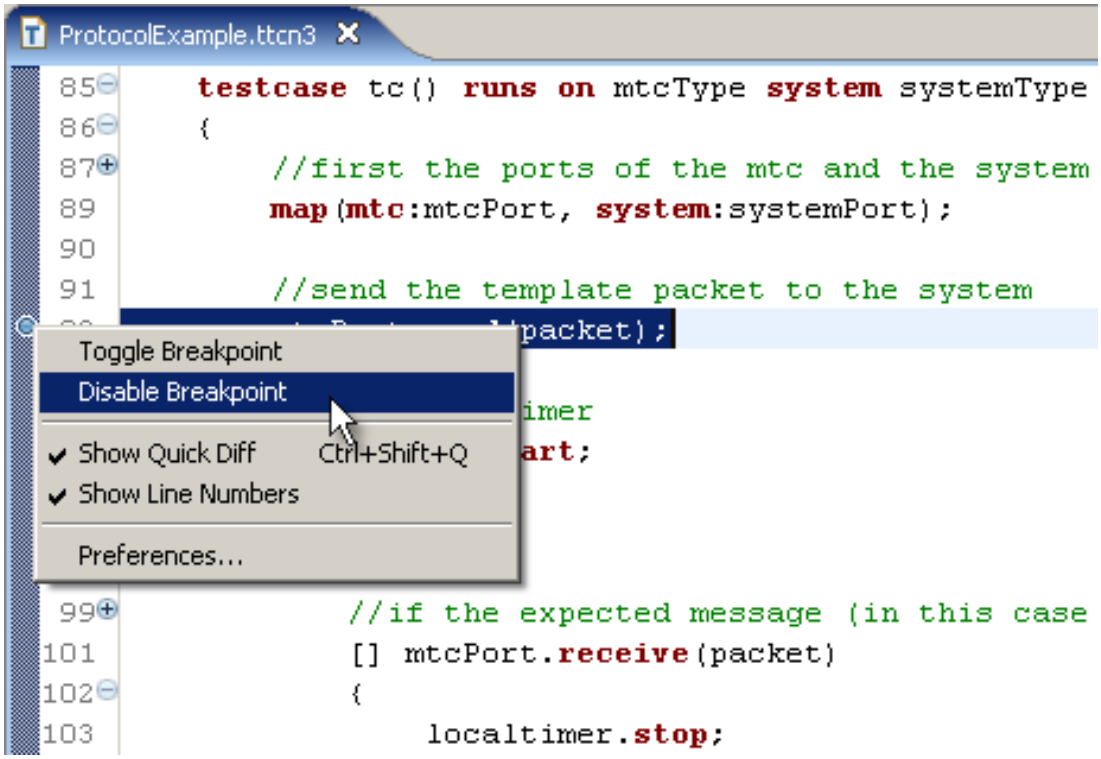
Renaming a project makes all breakpoints placed in that project invalid. It is strongly advised to remove all these breakpoints prior to renaming the project!

Temporarily disabling breakpoints

There are two ways to temporarily disable breakpoints:

- In the vertical ruler at the left side of the CL Editor click with the right mouse button on the breakpoint you wish to disable. In the context menu that now shows up, select **Disable BreakPoint**.

Figure 9.4. Disabling a breakpoint by using the context menu of the vertical ruler



- In the Breakpoints View, uncheck the checkbox at the left side of the breakpoint you wish to disable.

To enable the breakpoint again, both methods are suitable (the context menu entry now is named **Enable Breakpoint**.

Note

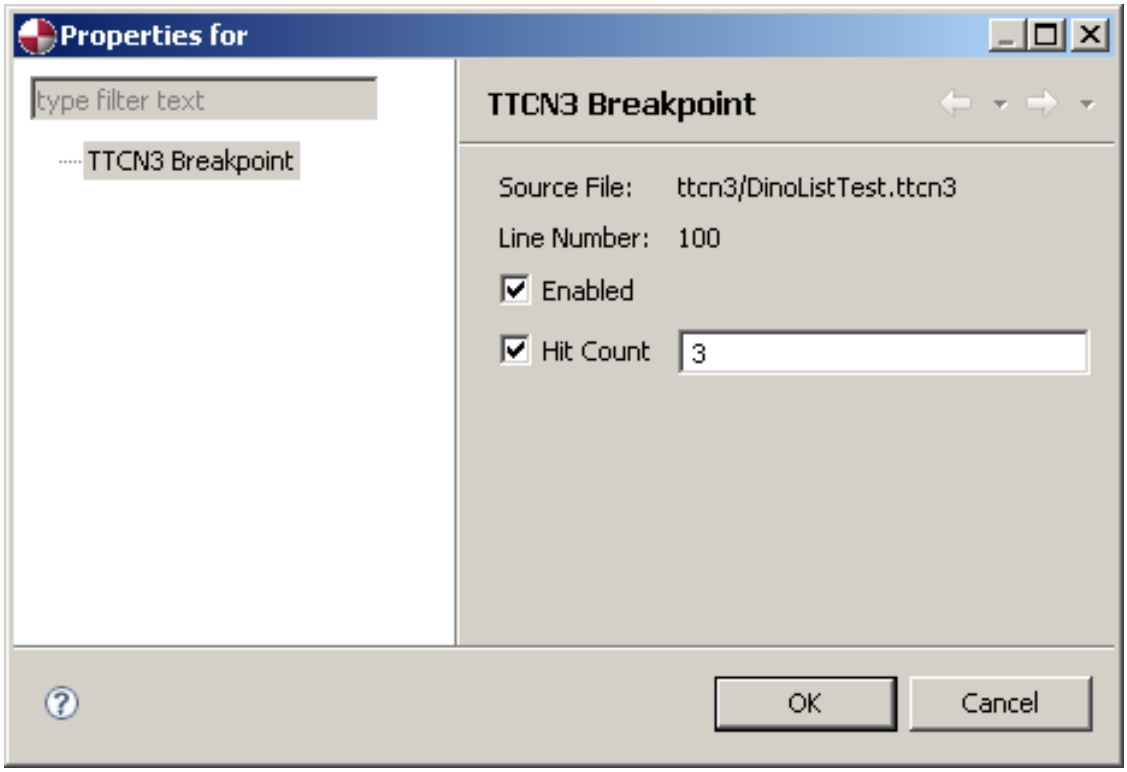
The Breakpoints View provides a button that allows to generally skip all breakpoints independent from their current state.

Setting a breakpoint hit count

By setting a hit count value, you can enforce the execution to only stop when the breakpoint was reached as many times as defined by the given value. The hit count value can be set on the breakpoint's property

page. To open a breakpoint property page, click with the right mouse button on the breakpoint either in the vertical ruler at the left side of the CL Editor or in the Breakpoints View and select Breakpoint Properties....

Figure 9.5. Breakpoint Properties Page

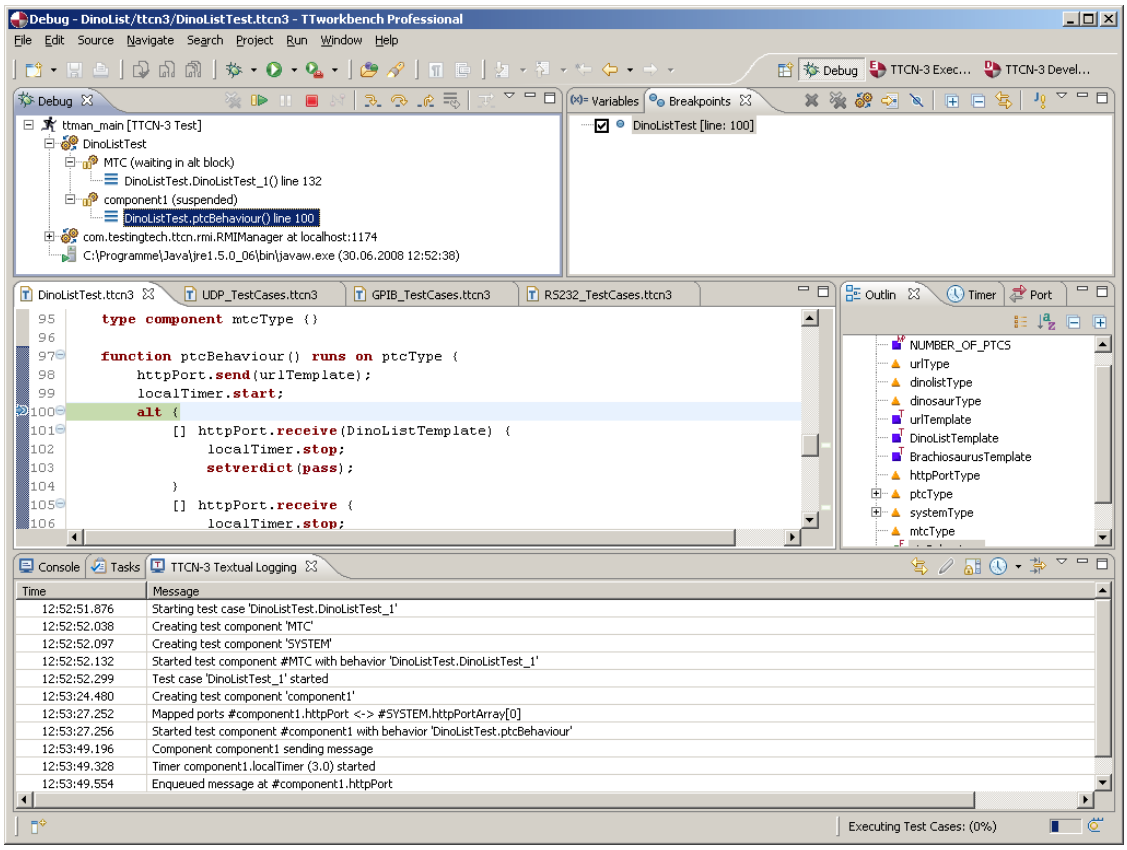


Debugging TTCN-3

Debug Perspective

TTdebug plugs into the standard Eclipse Debug Perspective.

Figure 9.6. Eclipse Debug Perspective



It utilizes the perspective's Debug View, Breakpoints View and Variables View to show its own content while providing a standardized user interface. TTdebug also adds a Timers View to this perspective and uses the Testing Technologies CL Editor to show the current execution position.

It is possible to automatically switch to the Debug Perspective after starting execution of a test suite in debug mode (see Automatic perspective switch). To open the perspective manually you have to press its icon in the perspective bar:

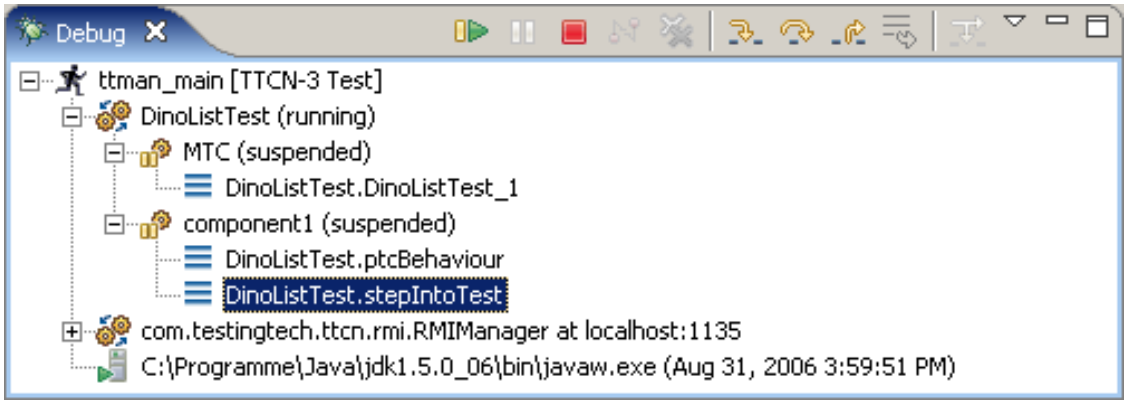
Figure 9.7. Perspective bar

If the Debug Perspective's icon is not shown in this bar, you can open it with **Window > Open Perspective > Other...** and selecting **Debug** in the upcoming window.

Debug View

The current execution status is shown in the Eclipse Debug View.

Figure 9.8. Debug View



The root of the tree stands for the executed suite. It has three children: The TTCN-3 debug target, the Java debug target and the Java Process on which the execution is based. For TTCN-3 debugging only the first one is of interest. It has a child for each component that is declared at this point of execution. Each suspended component contains a stack trace which shows the path of execution through functions and altsteps.

The Debug View provides buttons for the debugging control functions. The ones that are relevant for TTCN-3 debuggin now get described in detail:

- 

  Resume: Exits suspension mode and resumes normal execution from the current position (keyboard shortcut **F8**).

- 

  Suspend: Stops a running testsuite immediately.

- 

  Terminate: Terminates a running test suite immediately.

- 

  Step into: Executes the next instruction. If this instruction contains a call of a function or an altstep, the suite stops prior to the first instruction there (keyboard shortcut **F5**).

- 

  Step over: Executes the next instruction. If this instruction contains a call of a function or an altstep, the suite executes all contained behavior and stops prior to the next instruction in the current scope (keyboard shortcut **F6**).

- 

  Step return: Executes the current behavior scope to its end and stops prior to the next instruction in the scope above (keyboard shortcut **F7**).

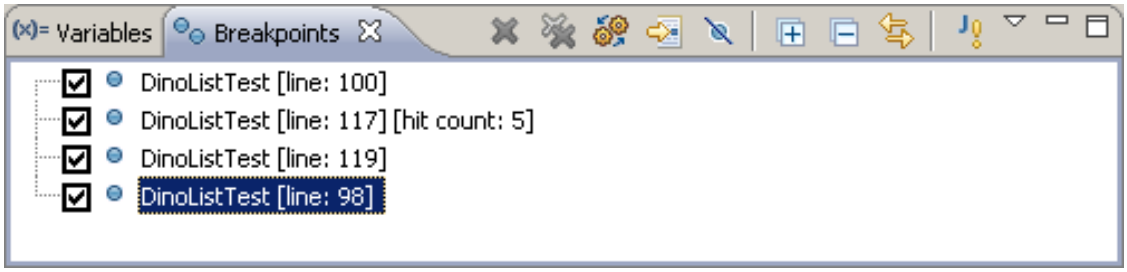
Note

If a component shows that it is "waiting in alt", it has to choose an alternative before you can continue to step! To enforce this, you can either manually trigger a timeout in the Timers View or step through another component (see Debugging multiple components).

Breakpoints view

Breakpoints can be seen and manipulated in the Breakpoints view.

Figure 9.9. Breakpoints View

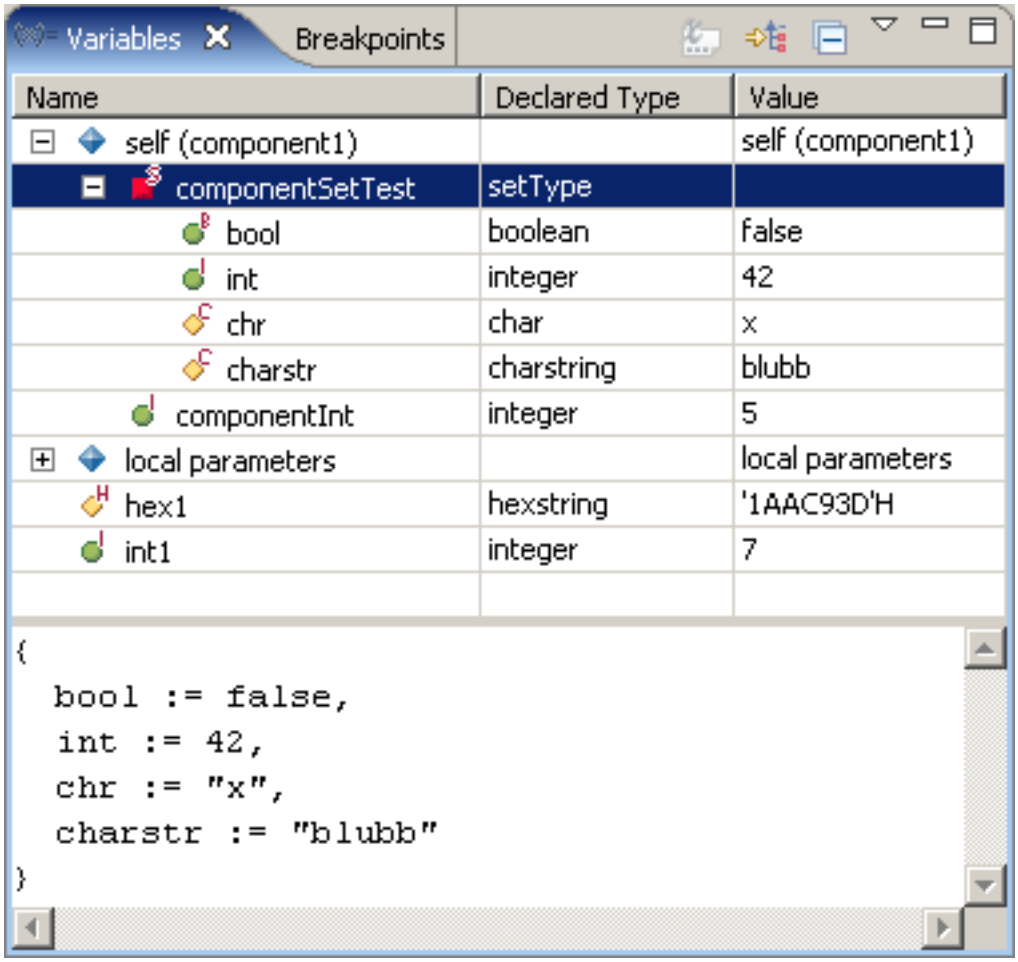


Each breakpoint has a checkbox on its left. It shows if the breakpoint is active, i.e. by unchecking it, the breakpoint gets ignored during execution. It is also possible to instruct TTdebug to ignore all breakpoints by using the "Skip All Breakpoints" button  . The context menu also provides diverse breakpoint manipulation functions including breakpoint removal and importing/exporting TTCN-3 breakpoints from/to an XML file on your file system.

Variables View

The Variables View displays all variables that are declared in the stack trace element that is currently selected in the Debug View.

Figure 9.10. Variables View

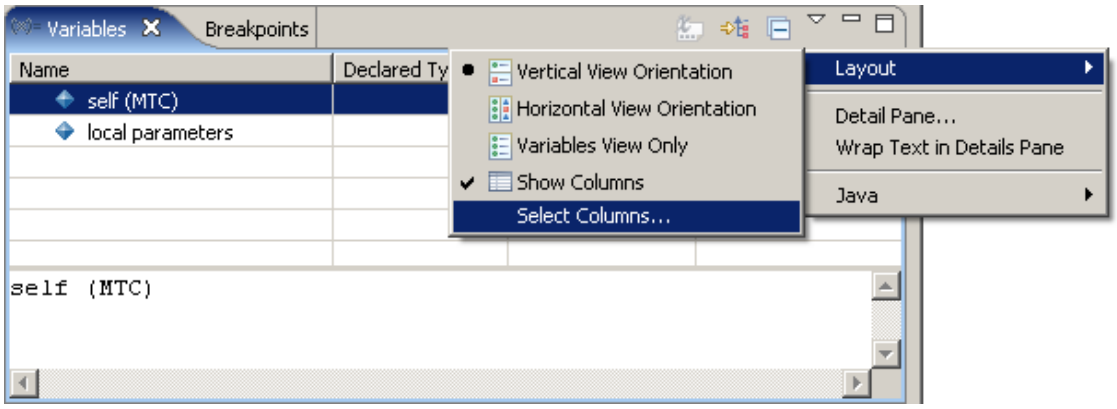


The Variables View shows the variables in a tree structure, where variables of a basic type are placed in a single line and variables of a structured type are organized into a tree which the user may browse and expand according to his needs. The first entry always contains the component variables, the second one the local parameters. The view also contains a Details Pane that displays the content of the variable that is currently selected in the variables tree. To show or hide the variable types, open the column selection dialog like shown in Figure 9.11, “Open the column configuration” and change the configuration.

Note

The **Declared Type** and the **Actual Type** columns currently display the same content.

Figure 9.11. Open the column configuration



To stop the execution when a specific variable gets modified, right click on that variable and choose **Toggle Watchpoint**. After resuming the execution, it will be stopped prior to the variable's modification.

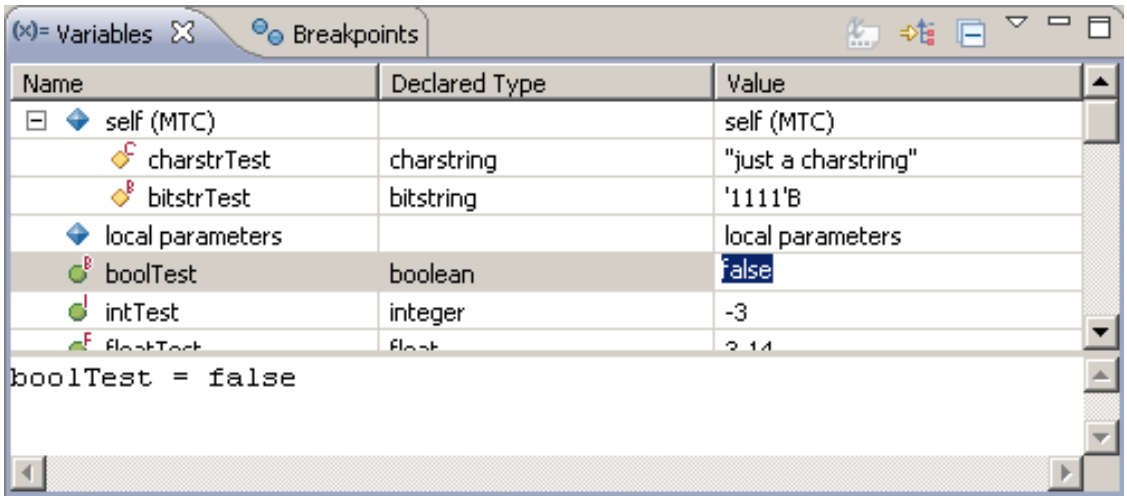
Note

Watchpoints are set for a specific instance of a variable, i.e. after the scope level in which the variable has been declared was left, the watchpoint automatically disappears, even if the same scope gets reentered. This also means that watchpoints only work for the component on which they were set.

You can set a hit count value for a watchpoint to enforce the execution to only stop when the variable was changes as many times as defined by the given value. The hit count value can be set on the watchpoint's properties page. To open a watchpoint properties page, click with the right mouse button in the Variables View and select Watchpoint Properties....

It is also possible to change the value of a variable; afterwards the current execution process will use the changed content. Variables or structured variables' subelements of a basic type like boolean or integer can be edited directly in the Variables View by clicking on the current value. To edit the structure of a structured variable, right click on it and choose **Change Value**, which causes an editor window to open. Now you can edit the value; by choosing **OK**, the new content will be saved. The value editor is based on the editor dialog of the Template Wizard, see there for additional info.

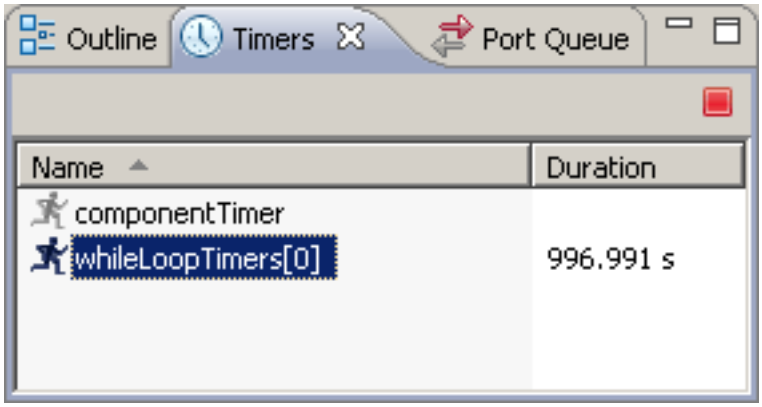
Figure 9.12. Editing a basic value in the Variables View



Timers View

The Timers View shows all timers that are declared in the stack trace element that is currently selected in the Debug View. Timers are stopped during suspension, but get reactivated while performing step commands.

Figure 9.13. Timers View



If the icon on the left of a timer is greyed out, then the timer is not running. The second column shows the remaining duration of the timer. By selecting timers and pressing the button in the Timers View menu bar the user can manually trigger timeouts.

Port Queue View

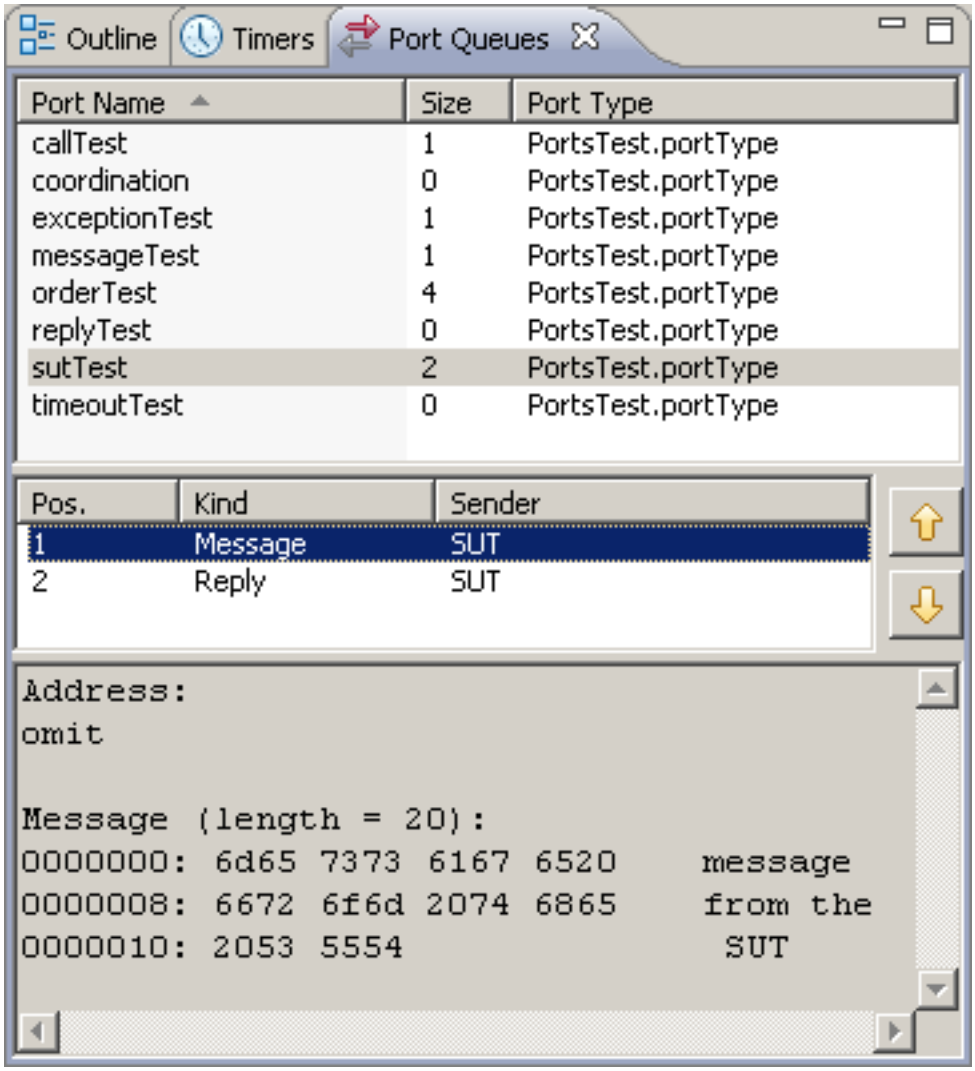
The Port Queue View is separated into three sections.

- The uppermost table contains all ports declared in the component that is currently selected in the Debug View. If you select one of the ports, the content of their message queue gets displayed in the section below.

- The section in the middle contains all messages that were received but not yet processed on the port that is selected in the section above. They are ordered by the time of their arrival; their position in the queue is additionally shown in the Pos. column. It is possible to manipulate the order by using the arrows right of this table. In the Operation column, the command that was used to send the message is displayed; it is either Message, Call, Reply, Exception or Timeout (timeouts are not actually received messages; they signal that a call command on their port could not be finished in the given timespan). The Sender column shows from where the message originates. If it was sent by another component, the component's id and the used port get displayed. If the sender was the SUT, SUT gets displayed.

- The lowest section displays the content of the messages that are selected in the table above. For messages sent by the SUT, the appropriate decoding hypothesis is not yet known. Because of that, for these messages the Port Queue View displays the raw received data in hexadecimal form and an interpretation of the data as ASCII code.

Figure 9.14. Port Queue View



Debugging multiple components

TTdebug provides a pure static component handling: If one component suspends, all components and all timers get suspended. Stepping through the suite only affects the component you are stepping with, all other components and their timers stay suspended. This allows the user to switch the component he is stepping with at any time.

Automatic perspective switch

When a test suite suspends on a breakpoint, the user gets asked if he would like to switch to the Debug Perspective. The dialog popup allows you to store the choices you made and, after that, Eclipse does not ask this again in future. To get to the location to change these setting afterwards you have to open **Window > Preferences...**. It is placed in **Run/Debug > Perspectives**, titled "Open the associated perspective when an application suspends".

Debugging Java

Since TTCN-3 suites rely on codecs and test adapters that are usually programmed in Java, it is important to be able to debug Java code as well as TTCN-3 code. TTdebug cooperates with the standard Eclipse Java debugger so that both can be used during the same test suite run. To use the Java debugger you have to set a breakpoint in the Java source file you wish to debug. When the Java debugger stops at this point, TTdebug automatically stops all other components and sets itself on hold until the user resumes Java execution.

To prevent the Java debugger to suspend every time an executable terminates, there is an option in the preferences which forces TTdebug to unset the according Java debugger option when loading a test executable. If you wish the Java compiler to suspend on uncaught exceptions, you have to uncheck that TTdebug option or to manually reactivate the Java debugger option while debugging (**Window > Preferences...**, then open **Java > Debug** and there set the checkbox "Suspend execution on uncaught Exceptions").

Note

If you wish to debug Java, but not TTCN-3, you may set the according options in the preferences to disable TTdebug without disabling the Java debugger.

Handling the SUT

Usually during the debugging process a test suite does not work as it should because the system under test does not suspend when the suite suspends. The standard Testing Technologies Testadapter provides the method "initializeDebugging()" which is called during suite initialization if the debugger is active. The standard implementation does nothing, but it allows the user to implement an own testadapter that inherits from the standard adapter and overwrites that method with code to prepare the SUT for debugging.

Preferences

To open the TTdebug preferences first select **Window > Preferences...** in the menu bar, then open **TTCN-3 > TTman** and there activate the tab "Debug". There are three options that affect TTdebug:

- "Enable Java Debugger": If this option is not set, it is not possible to suspend Java execution, neither manually nor by setting breakpoints (see Debugging Java).
- "Enable TTCN-3 Debugger": If this option is not set, the debugger does not get loaded together with the test executable and therefore cannot be used. This option is disabled, if the "Enable Java Debugger" option above is not set!
- "Prevent Java Debugger to suspend on uncaught exceptions": If this option is not set, the Java Debugger will suspend after the termination of the executable when TTdebug is enabled (see Debugging Java).

The meaning of the other entries on this page can be seen in the section called “Debug”.

Chapter 10. Additional Runtime Plugins

If additional runtime plugins are installed, you'll find accompanying documentation in the chapter tree view on the left.

Chapter 11. Frequently Asked Questions

Here you'll find the answers to some questions frequently asked by our customers.

1. *Q:* I've done an update of TTworkbench and now it starts to complain that it can't find several classes. Those classes seem to belong to TTworkbench.

   A: This might be a problem of Eclipse' configuration. To resolve this, start the eclipse.exe or TTworkbench.exe executable with the parameter `-clean`. The eclipse core runtime will rebuild caches of registered plugins. This has to be done only once. Afterwards Eclipse and thus TTworkbench should find all its classes.

2. *Q:* How to start the lmgrd FLEXnet Publisher license manager?

   A: To start FLEXnet Publisher license manager there are two possibilities:

   a. Interactive mode:
   ```
   lmgrd -z -c license.dat
   ```

   b. Daemon mode:
   ```
   lmgrd -l <logfile> -c license.dat
   ```

   Further information can be found in FLEXnet Publisher end user's guide at http://www.macrovision.com/services/support/enduser.pdf .

3. *Q:* In Execution perspective I always get an error when trying to load a CLF file.

   A: If you get following error message

   ```
   The error was detected in
    class:com.testingtech.ttworkbench.ttman.UIImpl
     java.rmi.ConnectIOException: Exception creating connection to:
    localhost; nested exception is:
     java.net.SocketException: Invalid argument or cannot assign
    requested address
     ...
   ```

   you've hit a known problem with Java 1.4.2 on RedHat Fedora Core 4. There are three possible workarounds:

   a. Install and use Java 1.5.

   b. Disable SELinux extension system-wide (using the configuration tool system-config-securitylevel as root).

   c. Start TTworkbench with additional parameters on command-line: **-vmargs -Djava.net.preferIPv4Stack=true** If this works on your system, you can change this setting permanently by changing the file `ttworkbench.ini` inside TTworkbench's installation folder like this:

   ```
   -nl
   ```

```
en
-vmargs
-Xms256m
-Xmx512m
-XX:MaxPermSize=256m
-Dfile.encoding=UTF-8
-Djava.net.preferIPv4Stack=true
```

4. *Q:* How can I tell TTworkbench to use a specific Java JRE/JDK?

   A: Start TTworkbench with additional parameters on command-line: **-vm /path/to/java/bin/java** on Linux or **-vm c:\path\to\Java\bin\javaw.exe** on Windows respectively to specify the JVM executable. If this works on your system, you can change this setting permanently by changing the file `ttworkbench.ini` inside TTworkbench's installation folder like this:

   ```
   -nl
   en
   -vm
   /path/to/jre/or/jdk/bin/java
   -vmargs
   -Xms256m
   -Xmx512m
   -XX:MaxPermSize=256m
   -Dfile.encoding=UTF-8
   -Djava.net.preferIPv4Stack=true
   ```

   Please mind the line break after `-vm` .

5. *Q:*I observed that the execution of the first test case in my campaign needs much more time than the execution of the next test cases. What can I do to improve the performance of the first test case.

   *A:*There are several system properties that can be set in order to increase the performance of the first executed test case. All these options trigger a specific initialization in the TE which would be normally made during the execution of the first test case. The options can be given as VM arguments having the syntax **-Dmyoption=value**. For setting the additional VM arguments please refer to TTman runtime preferences.

   Following options are defined:

   - **preload.modules** - if set enforces pre-loading of all classes from the compiled module jars.
   - **preload.templates** - if set enforces the initialization of all not parameterized templates defined in the loaded modules.
   - **preload.tt3runtime** - if set enforces pre-loading of all classes built-in runtime libraries.
   - **preload.ta** - if set enforces pre-loading of all classes from the test adapter jar file.
   - **preload.modulepars** - if set enforces accessing (and thus also initialization) of all module parameters.
   - **logging.binary** - if set enables the internal use of a binary instead of the default XML encoding of the logging.

Note

Please note that using any of these option will increase the load time of the test campaign.

6. *Q:* How to migrate TTthree 1.3

   A: To migrate TTthree 1.3 you have to do the following steps:

   a. **Removing old TCI**

   The most significant API change made within TTthree 1.3 is removing of the old proprietary TCI interfaces. That is, some classes/interfaces from the package `com.testingtech.ttcn.tci` were removed, as from TTthree 1.3 on only the standardized TCI is supported.

   Following classes from this package were removed:

AnyOrOmitValue.java	AnyOrOmitValueIf.java	BitstringValue.java	BitstringValueIf.java
BooleanValue.java	BooleanValueIf.java	CharstringValue.java	CharstringValueIf.java
CodecServer.java	Component.java	Decoder.java	DecodingException.java
Encoder.java	EncodingException.java	EnumeratedValue.java	EnumeratedValueIf.java
ExtendedTciTypeClass.java	FloatValue.java	FloatValueIf.java	HexstringValue.java
HexstringValueIf.java	IntegerValue.java	IntegerValueIf.java	Module.java
OctetstringValue.java	OctetstringValueIf.java	RecordOfValue.java	RecordOfValueIf.java
RecordValue.java	RecordValueIf.java	TciBehaviourIdInterface.java	TciClient.java
TciException.java	TciLogging.java	TciLoggingImpl.java	TciManagement.java
TciManagementIf.java	TciMessageImpl.java	TciOperationalIf.java	TciValue.java
Type.java	TypeClassWrapper.java	TypeServer.java	UnionValue.java
UnionValueIf.java	UniversalCharstringValue.java	UniversalCharstringValueIf.java	VariantValue.java
ValueServer.java	ValueServerImpl.java	ValueWrapper.java	VerdictValue.java
VerdictValueIf.java			

   Instead of the interfaces defined in this package, the ETSI TCI interfaces located in the package `org.etsi.ttcn.tci` have to be used. In order to use the ETSI TCI interfaces, the `TTorg.jar` file has to be present in CLASSPATH.

   As the standardized TCI does not provide any exception, the old `TciException`, `EncodingException` and `DecodingException` have to be replaced by user defined exceptions.

   b. **ModuleParameterServer**

   As the old TCI is no more supported, the method from the interface `com.testingtech.ttcn.tci.ModuleParameterServer` using the old TCI Value interfaces was removed.

   That is, the implementations of the method

```
public               com.testingtech.ttcn.tci.Value
getModuleParameter(com.testingtech.ttcn.tci.Type    parameterType,
String parameterIdentifier, String value) ;
```

have to be removed from the classes implementing this interface.

c. **RB**

As the old TCI is no more supported, the following methods have been removed from the class RB:

```
public TciManagement getTciManagement()

public void setTciManagement(TciManagement tciManagement)

public TciValue getTciValue()
```

The public field `runtimeBehavior` was also removed, as it was also used by the old TCI. Statements like `RB.runtimeBehavior.function()` have to be replaced by `RB.function()`.

To enable the usage of multiple logging implementations at the same time, the class RB uses internally a logging dispatcher. That is, if the method `setLogging` is called, the given object is not set as logging in RB, but the logging implementations is added to the list of loggers. This means, that a call to `getLogging` will return the logging dispatcher not the implementation passed to `setLogging`.

d. **LoggingInterface**

The interface `LoggingInterface` has been deprecated, as the "real" logging interface is now `TXILoggingInterface`. For backwards compatibility reasons, the interface `LoggingInterface` is still used, but is only extends now `TXILoggingInterface`. In the next major TTthree release the `LoggingInterface` will be probably replaced by the `TXILoggingInterface`.

Some logging methods are now deprecated and similar methods with additional parameters have been introduced.

The following methods are new deprecated:

```
   public void logCallMapped(long time, Object source,
       TriComponentId componentId, TriPortId tsiPortId,
TriAddress sutAddress,
       TriSignatureId signatureId, TriParameterList
parameterList,
       String message);

   public void logReplyMapped(long time, Object source,
       TriComponentId componentId, TriPortId tsiPortId,
TriAddress sutAddress,
       TriSignatureId signatureId, TriParameterList
parameterList,
       TriParameter returnValue, String message);

   public void logRaiseMapped(long time, Object source,
       TriComponentId componentId, TriPortId tsiPortId,
TriAddress sutAddress,
       TriSignatureId signatureId, TriException exception, String
message);
```

The new methods that have to be implemented by logging implementations are:

```
   public void logCallMapped(long time, Object source,
       TriComponentId componentId, TriPortId tsiPortId,
TriAddress sutAddress,
       TriPortId sourcePortId,
       TriSignatureId signatureId, Value[] parameters,
       TriParameterList parameterList, String message);

   public void logReplyMapped(long time, Object source,
       TriComponentId componentId, TriPortId tsiPortId,
TriAddress sutAddress,
       TriPortId sourcePortId, TriSignatureId signatureId,
       Value[] parameters, TriParameterList parameterList,
       Value tciReturnValue, TriParameter returnValue, String
message);

   public void logRaiseMapped(long time, Object source,
       TriComponentId componentId, TriPortId tsiPortId,
TriAddress sutAddress,
       TriPortId sourcePortId, TriSignatureId signatureId, Value
tciValueException,
       TriException exception, String message);
```

e. **TTorg.jar**

The standardized interfaces (TCI and TRI) are now packaged separately in the library `TTorg.jar`. This files has to be added to the CLASSPATH when compiling test adapters and codecs that use these interfaces.

7. *Q:* I have problems running my tests but the configuration seems to be okay. What's wrong?

   A: On most machines a personal firewall is running. Sometimes the security options are too high so TTworkbench is not able to communicate. Please make sure that your security level is not too high, or create exception rules for it. For further instructions please refer to the users guide of your security software.

8. *Q:* I am using TTworkbench under Linux and for example German umlauts, Japanese or Chinese characters are not properly displayed in the data view of TTman.

   A: TTworkbench uses UTF-8 encoding. A proper display of all characters in the TTman data view or in a eclipse console view requires the appropriate language setting on your Linux machine before starting TTworkbench. You can check your setting by typing **locale** in a Linux shell, the `LANG` environment variable has to include UTF-8 encoding, i.e. `en_GB.UTF-8`. If necessary, set the environment variable `LANG` accordingly or ask your system administrator to configure the language globally on your machine (e.g. for SuSE by using **yast** (system -> language and system -> environment -> language).

9. *Q:* I've imported an older TTCN-3 project into my workspace and I get the error "Project MyProject is missing required library: 'C:\TTwb\plugins\com.testingtech.ttworkbench.ttthree.core_1.0.4\lib\TTtools.jar'". With a previous version of TTworbench the project was running without any problems.

A: The location of the file TTtools.jar was changed so that the old path is no longer valid. You can update the build path of the project by right clicking on the project and selecting from the pop-up menu the item "Update TTtools path".

Figure 11.1. Update TTtools path

Chapter 12. What's New

This chapter provides the current TTworkbench "What's new" information.

Chapter 13. Contacting Technical Support

Telephone and Email support is available Monday through Friday (except holidays). When contacting Technical Support through Email, please include the following information along with a detailed description of your problem:

- Name, telephone number, and company name
- Make and version number of operating system
- Product release number
- Your Log Id (if you are calling about a previously reported problem)

Upon receipt of your request, Testing Technologies Technical Support will send you an response with your Log Id # and point of contact for your issue.

Testing Technologies Technical Support Contact Information

This information was accurate at the time of printing. If you experience any difficulty contacting us using this information, please check our web site at www.testingtech.com for the most up-to-date information.

Testing Technologies Technical Support
Michaelkirchstrasse 17/18
10179 Berlin
Germany

phone	+49 30 726 19 19 0
fax	+49 30 726 19 19 20
internet	www.testingtech.com
ticket system	support.testingtech.com

Index

A

Abstract Syntax Notation One (ASN.1), 1
ASN.1 Encoding Rules (BER, CER, DER, XER), 1

C

Campaign, 151
Campaign Wizard, 154
Checked Non-Standard Language Extensions, 126
Clean jar files, 133
Command Line Mode
  PluginHomeResolver.sh, 138
  TTman.bat, 201
  TTman.sh, 201
  TTthree.bat, 133
  TTthree.sh, 133
  TTthree2.sh, 138
  ttthreeenv.sh, 138
Core Language
  Display, 59
  Mapping, 56

D

Debug mode, 133
Default Campaign, 151
Destination path, 134
Diagram
  Creation, 69

E

Eclipse, 1
  Eclipse Download, 4
  Eclipse Graphical Editing Framework - GEF, 2
  Eclipse Modeling Framework - EMF, 2
  EMF Download, 5
  GEF Download, 5

J

Java code, 135

K

Known limitations, 125

L

Language, 136
Language Features, 124
lmgrd , 3
Logging
  Export, 173
  Filters, 185, 189
  Generation, 195
  Graphical, 185
  Import, 173
  Offline, 195
  Textual, 183

M

Meta Campaign, 158
Meta Campaign Wizard, 158
Meta model, 134
Module Loader File
  Package attribute, 137

O

OSGI file locking, 135
Other Non-Standard Language Extensions, 127

P

PluginHomeResolver.sh, 138
Project path, 136
PTCC, 1

R

Reports, 174

S

Session, 173
Symbols
  Attributes, 60

T

T3Doc, 140
t3server, 139
Target package, 137
TCI Specification, 1
Technical support,
  Contact information, 226
  Log Id, 226
Test Adapter, 162
TRI Specification, 1
TTCN-3 Core Notation
  Specification, 1
TTman exit codes, 202
TTman Options, 202
TTman.bat, 201
TTman.sh, 201
TTrun
  JAR File, 136
TTthree exit codes, 138
TTthree Options, 133
TTthree.bat, 133
TTthree.sh, 133

TTthree2.sh, 138
ttthreeenv.sh, 138

V

Verbosity, 138
Version, 138, 202