Rational. Systems Tester

IBM.

User Guide

Copyrights

This edition applies to IBM Rational Systems Tester version 3.3 and to all subsequent releases and modifications until otherwise indicated in new editions.

Copyright Notice

IBM Patents and Licensing

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send written license inquiries to the following:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send written inquiries to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Intellectual Property Dept. for Rational Software|
IBM Corporation
1 Rogers Street
Cambridge, Massachusetts 02142
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Disclaimer of Warranty

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MER-

CHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions. Therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Confidential Information

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Additional legal notices are described in the legal_information.html file that is included in your software installation.

Sample Code Copyright

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs.

IBM Trademarks

For a list of IBM trademarks, visit this Web site www.ibm.com/legal/copytrade.html. This contains a current listing of United States trademarks owned by IBM. Please note that laws concerning use and marking of trademarks or product names vary by country. Always consult a local attorney for additional guidance. Those trademarks followed by ® are registered trademarks of IBM in the United States; all others are trademarks or common law marks of IBM in the United States.

Not all common law marks used by IBM are listed on this page. Because of the large number of products marketed by IBM, IBM's practice is to list only the most important of its common law marks. Failure of a mark to appear on this page does not mean that IBM does not use the mark nor does it mean that the product is not actively marketed or is not significant within its relevant market.

Third-party Trademarks

Adobe, the Adobe logo, Acrobat, the Acrobat logo, FrameMaker, and PostScript are trademarks of Adobe Systems Incorporated or its subsidiaries and may be registered in certain jurisdictions.

AIX and Informix are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Introduction

IBM Rational Systems Tester 3.3 is a tool for testing of advanced software systems using TTCN-3.

To fully take advantage of IBM Rational Systems Tester and be able to start working quickly, it may prove useful to start with:

- Chapter 12, Useful Shortcut Keys will provide a listing of possible shortcuts, this chapter can provide you with information on how to work faster and more efficient once you are familiar with what IBM Rational Systems Tester can achieve.

1

Introduction to IBM Rational Systems Tester 3.3

TTCN-3

IBM Rational Systems Tester contains a complete set of tools for designing, creating, and executing TTCN-3 test suites. With this you create projects in a workspace where you administer, edit and analyze different sets of files – your test suite. You can develop, visualize, and execute tests – all in the same development environment.

The TTCN-3 tool set is often referred to as Rational Systems Tester. Rational Systems Tester includes editors for TTCN-3, ASN.1, and text; project build facilities, and an integrated MSC viewer.

To fully take advantage of the TTCN-3 tool set and be able to quickly start working with TTCN-3, the topics listed below may prove useful to start with:

- Editing Abstract Test Suites

  Manage your workspace, projects, and files. Explore how to create, edit, and maintain your abstract test suites.
- Creating an ETS

  Generate C code from your test suite in order to build an executable test suite.

- Execute Tests

  In this section, you find all related information on test execution: reports, execution progress, and test verdicts. Also descriptions of the various log formats.

- Tutorial

  A basic tutorial that allows you to be familiar with Rational Systems Tester and the syntax.

Overview of IBM Rational Systems Tester User Interface

The complete IBM Rational Systems Tester user interface contains several different areas that can be switched on and off at the will of the user: the **Desktop**, the **Workspace** window and the Output window. In addition there is a status bar in the lower part of the frame and a menu and Toolbar area at the top of the interface frame. The windows are all possible to dock according to the user's preferences. It is also possible to drag-and-drop frequently used toolbars to a Shortcuts window.

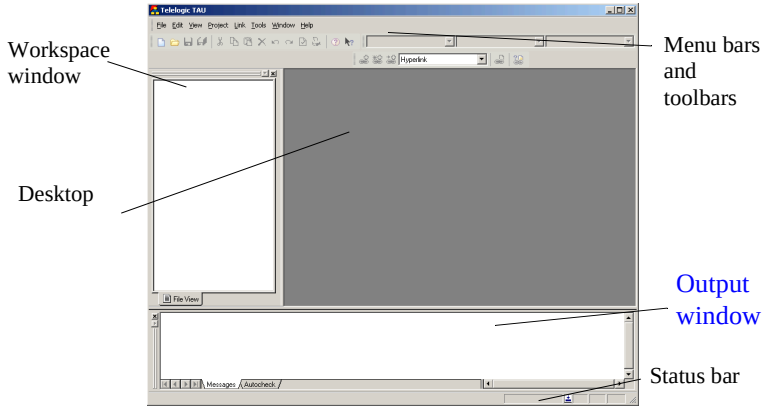

Figure 1: The IBM Rational Systems Tester Desktop.

Desktop

The Desktop, or editing area, is the area of your working documents. This is where the actual development takes place. Here you will see diagrams, documents, source files, etc. Once you have opened them for editing or viewing. Which editor or viewer that is displayed depends on the file types that are included in your project.

If you have more than one document opened on the desktop, you can move between with commands on the Window menu, or by pressing CTRL + TAB (forwards) or CTRL + SHIFT + TAB (backwards).

Hint

If you want to have your editor expanded to full screen, select Full Screen in the View menu. To go back, press ESC or ALT + '1'.

See also

"Working with windows" on page 7

Workspace window

The Workspace window is a graphical tool that presents and manages the structure of the workspace information in a number of Views.

The Workspace window shows the information structure with expandable nodes. By collapsing and expanding these nodes, and by using different views, you can focus on different sets of the information in the workspace.

It is possible to move the Workspace window and to make it a floating palette. When not floating, it is docked at this left-most position. When docked, the window can only be resized horizontally. The vertical boundary is determined at the top by the toolbars and at the bottom by the Output window.

Edit markers

Gray bar

In the File View elements that can not be edited are marked with a gray bar between the element symbol and the element name.

Asterisk

When a text file has been edited but not yet save an asterisk will appear in the window title bar after the file name.

Views

In the Workspace window you have access to a number of views. They are each accessible via separate tabs. The views show different aspects of the model.

File View

The File View shows your workspace with all elements that are represented as files.

You select the File View by clicking the **File View** tab in the **Workspace window**. Here you can open, edit and save all files. However, if you delete a file it is only deleted from the File View. The file is still available in the file system of your operating system.

To get a better overview of your files, you can create folders. You can drag and drop files between folders. You can display properties by selecting an item, right-clicking it, and clicking Properties in the shortcut menu.

Structured View

The Structured View displays the structure of test suites, sorted by language elements. When you edit a TTCN-3 file, this is reflected in the structure. This view is updated each time you save your work.

Files

Files are represented by icons in the File View in the Workspace window.

You may insert any file into your projects that you feel belongs there. The inserted files will show up as icons in the File View of the Workspace window. The Workspace window will start the associated program for viewing or editing if you double-click the file icon.

Shortcuts window

The Shortcuts window may contain toolbars, but it will only do so if you put them there. To put a Toolbar in the Shortcuts window, right-click the toolbar and click **To Shortcut** from the shortcut menu. You can reverse the process by right-clicking on the Shortcuts shown and clicking **As Toolbar** from the shortcut menu that appears.

The **View** menu **Shortcuts** command controls if the Shortcut window is shown.

Note

Not all toolbars will be possible to submit to the shortcuts window.

Output window

The Output window consists of a number of different tabs that records and displays information for the corresponding tool. This information is typically error messages, warnings, result of actions, logging of events, etc. Each tab represents a different tool.

For some of the tabs, it is possible to navigate from the located element (in the subject column) to a presentation in a diagram.

The **View** menu **Output** command controls if the window is shown.

General tabs

Messages

The Messages tab shows general information regarding the project loading and other executed actions.

No navigation is available from this tab to other parts of the tool.

Search result

This tab displays the result of a Find operation.

Presentations

This tab displays the result of a List presentations operation.

References

This tab displays the result of a List references operation.

Script

The Script tab shows the result of scripts, for example Tcl scripts.

TTCN-3 tool tabs

Find in Files 1

This is the default tab for search results.

Find in Files 2

When you search in multiple files, you have the option to display your search results in a second tab. This enables you to perform a second search without losing the result from the first.

Tester Analyzer

When TTCN-3 is analyzed, all logs are displayed in this tab. The logs include error messages, warnings, and so on.

TTCN to C Compiler

In the TTCN to C Compiler tab, analysis and code generation messages, such as warnings and error messages, are presented in a clear and readable way.

It is possible to navigate from error messages or warnings to the source in the test suite.

Build

All information generated when you build your executable test suite is displayed in the Build tab.

Execution

In the Execution tab, information from the execution, together with verdicts, is displayed to give you all necessary information regarding the outcome of a test run.

From a given verdict, you can navigate to the source of a given test case by double-clicking the log item.

Objects

When you click the List TTCN-3 Objects button in the toolbar, a list of all TTCN-3 objects (or language elements) in your active project is displayed in this tab.

Working with windows

Arrange windows

Tile all document windows:

- On the **Window** menu, click **Tile Horizontally** or **Tile Vertically**.

Overlap document windows:

- On the **Window** menu, click **Cascade**.

To change the position of document windows:

Note

The docking state can not be changed for a window with tabbed documents.

1. Right-click the title bar of a document window.
2. In the menu select:
   - **Docked** to dock the window within the application window. There are also options for where it should be docked.
   - **Floating** to be able to move it outside the application window.
   - **MDI Child** to make the window float within the editing area. There are also options for maximizing, minimizing and restoring the window.

To view the active document in full screen:

- On the **View** menu, click **Full Screen**

  Or

- Press **ALT + 1**

To view the active document in normal size:

To display the active document in normal size again after you have viewed it in full screen, do one of the following:

- Move the cursor to the top of the screen. When the menu bar appears, on the **View** menu, click **Full Screen**.

  Or

- Press **ALT + 1**

Show and hide windows

Show or hide the workspace window:

- On the **View** menu, click **Workspace**

  Or

- Press **ALT + 0**.

Show or hide the output window:

- On the **View** menu, click **Output**

  Or

- Press **ALT + 2**.

Close windows

To close a document window:

- On the **Window** menu, click **Close**.

To close all document windows:

- On the **Window** menu, click **Close All**.

Create a new window

To create a new document window:

- On the **Window** menu, click **New Window**.

Tabbed documents

If the option “Tabbed documents” is enabled in general options page, documents will be opened as tabs in a single window.

A document can be detached from a tabbed window by right-clicking the tab and selecting the Detach menu item. It will then function as a normal MDI child and the docking state can be changed.

Docking windows

There are three different modes for editor windows in the IBM Rational Systems Tester framework. These are applied to each diagram window individually by right-clicking the diagram title bar.

Note

The docking state can not be changed for a window with tabbed documents.

Docked

A docked editor window will align into the IBM Rational Systems Tester framework like the workspace window or the Output window. It will be possible to move the windows around to arrange a suitable view.

Floating

A floating window is on top of the IBM Rational Systems Tester framework. It will turn into a docked window if it is moved towards the framework frame.

MDI child

An MDI child window is positioned into the desktop area. Adjusting can be done manually inside this area or with commands from the **Window** menu.

Auto-hide docked window (Windows)

A window that has a pin symbol in the gripper bar can set to auto-hide mode. If the pin is pressed, the window will be hidden and a label representing the window will be displayed instead. By hovering with the mouse over the label the hidden window will be displayed. Window can be docked again by clicking on the pin symbol again.

Expand/Contract docked window

When two docked windows share the same side of the main window, a window can be expanded to take the whole side in possession by clicking on the arrow symbol on the gripper bar of the window (if available). By doing this the other windows on the side are minimized. The window can be contracted by clicking on the arrow symbol again.

Stored workspace windows

All windows opened during a session will be reopened when the workspace is loaded again. The information will be saved in a .ttx file with the same path and name as the workspace.

See also

"Organizing the view" on page 171

Menu bar and toolbar

When you first start IBM Rational Systems Tester, the toolbars appear just below the menu bar.

Depending on your workspace preferences, and the size of your screen, you can display as many toolbars as you want, or none at all. You can add a button with a command to a toolbar, change the size of the buttons, and move the toolbars to different locations to suit your needs.

Menu Bar

The menu bar contains well-known menus such as File, Edit, Project, and so on. Depending on the task you are performing the number of menus differ.

Most menu commands have a shortcut assigned. A list of Useful Shortcut Keys is found in Common Reference.

You can add commands to the Tools menu allowing you to easy access to non-IBM Rational tools. This is done from the Tools tab.

As an example, the following procedure demonstrates how to add the Windows Notepad accessory to the Tools menu.

To add a command to the Tools menu:

1. From the **Tools** menu select Customize, and then click the Tools tab.
2. Click the New (Insert) button.
3. Type the name of the tool, as you want it to appear on the Tools menu, and press ENTER.

   For example, if you want to add a command for the Windows Notepad accessory, you might type Notepad.
4. In the **Command** box, browse or type the path and name of the program, for example, C:\Windows\notepad.exe.
5. In the **Arguments** text box, browse or type any arguments to be passed to the program. Leave this field empty for the Notepad accessory.

Note

You can use the drop-down arrow next to the Arguments text box to display a menu of arguments. Select an argument from the list to insert argument syntax into the Arguments text box.

6. The **Initial Directory** box is used to specify the file directory where the executable file for the command is located. For the Notepad accessory this field is left empty.

When the command appears on the Tools menu, you may click it to run the program.

You can add arguments to be passed to the program by typing them in the Arguments text box, or set the initial directory for your program by typing it in the Initial Directory text box.

If the program you are adding to the Tools menu has a `.pif` file, the startup directory specified by the `.pif` file overrides the directory specified in the Initial Directory text box.

Toolbar

The toolbar allows you to set up a palette of your most common used tools in order to have quick access to them. Once you have made any changes to the toolbar, these changes are saved and retrieved for your next work session.

The standard toolbar corresponds to the operations available in the menu bar. The standard toolbar can be toggled on and off from the View menu (Standard command) or from the toolbar area's shortcut menu, the other toolbars only from the shortcut menu.

Note

Not all toolbars and commands can be modified. This feature belongs to the IBM Rational Systems Tester framework and is not supported for toolbars related to editors.

To add a toolbar button:

1. Make sure that the toolbar you want to change is displayed.
2. From the **Tools** menu select Customize, and then click the **Commands** tab.
3. Add a button by clicking the name of the category in the Categories box, and then dragging the button or item from the Buttons area to the displayed toolbar.

To delete a toolbar button:

1. Make sure that the toolbar you want to change is displayed.
2. From the **Tools** menu select Customize, and then click the **Commands** tab.
3. To delete a button, drag it off the toolbar.

When you delete a default button from a toolbar, the button is still available in the Customize dialog box. However, when you delete a toolbar button with a custom appearance, its appearance is permanently lost, although the command is still available (Customize dialog box, Commands tab).

Hint

To save a toolbar button with a custom appearance for later use, create a toolbar for storing unused buttons, move the button to this storage toolbar, and then hide the storage toolbar.

Show or hide toolbars:

1. From the **Tools** menu select Customize, and then click the **Toolbars** tab.
2. Select and clear toolbars to show or hide in the **Toolbars** list.
3. Click Close.

alternatively

1. Right-click anywhere in the toolbar area in the user interface.
2. Click the toolbar you want to show or hide. The menu closes automatically.

To change the appearance of toolbar buttons:

1. From the **Tools** menu select Customize, and then click the **Toolbars** tab.
2. Select the following options:
   - **Show Tooltips** to enable tooltips to be displayed when the cursor moves over a button or field in the toolbars.
   - **Large Buttons** to display larger sized buttons in the toolbars.
3. Click **Close**.

Status bar

The status bar presents useful information about status of several different types of tasks, for example it lists errors and tooltips. Here will also be presented information about progress and current actions.

For TTCN-3 projects and text files the current line number and column position are shown in the right most corner of the Status bar.

Line navigation

Navigation to a specific line in a text file is done by pressing CTRL + SHIFT + G. Enter the wanted line number in the dialog that opens.

Progress bar

There is one progress bar displayed showing the overall progress when opening a workspace to the right of the status bar.

There can also be one displayed for the progress of the separate parts of the loading process displayed in the message field. In this case there will also be a message explaining the current action in progress.

Options

Tool options affect IBM Rational Systems Tester, not just the current project or workspace. There are different ways of changing these options:

In the Options dialog, there are different tabs for different options that you can change. The number of tabs differs depending on what type of project that is active. To see a description of an option in the Options dialog box, click the question mark in the dialog box title bar, and then click the option.

In the Advanced tab you may use a tree view for all options available in the Options dialog. The Advanced tab is activated from a check box in the General tab. To change the option values in the Advanced tab, select the value and click F2.

Options file

The option settings can be saved in an options file, `.tot`. This file can later be edited.

If the options file is added to a project, you can double-click it in the File View to open it in an options editor. It is possible to save several options files in a project and select which one should be used. This is useful if you often switch options: Instead of changing the options directly, you just change the priority of the options files.

In the installation there will be a number of files with a `.tot` extension containing internal framework settings and options. These files should normally not be edited by the user. Editing a file with a `.tot` extension may cause loss of data and incorrect behavior of the tool set. The options controlled should be edited from the Tools menu Options dialog.

Change options

To change options:

1. On the **Tools** menu, click **Options**.
2. In the **Options** dialog box, select and clear options in the different tabs. In the Advanced tab press F2 to access an option value.
3. Click **OK**.

Work with options Files

Save the current options in a new options file:

1. On the **Project** menu, click **Options** and then **Save As**.
2. In the Save As dialog, select a name and location for the options file (.tot).
3. Click **Save**.

4. Click **Yes** when you are asked to include the options file in your active project.

   By including it in your project, you will be able to open it from the File View and edit it.

To edit an options file:

1. Make sure that an options file is included in your project, that is, visible in the File View.
2. Double-click the options file. It will be opened in the options editor.
3. In the options editor, expand the options tree until you are able to see your options.
4. Select an option.
5. On the selected option, click the Value field and press F2. This will make the field editable.
6. Enter a new value.
7. Close the file when you are finished. You will be prompted to save your changes.

Select which option file to use:

1. This is only possible when there are more than one options file included in your project, that is, visible in the File View.
2. On the **Project** menu, click **Options** and then **Files**.
3. In the Option Files dialog box, select an options file in the list and click the arrow buttons to move it up or down. The options file on top is the options file that will be used.
4. Click **OK**.

Customizing

It is possible to customize the appearance of the user interface. In the **Customize** dialog box (**Tools** menu), there are options for customizing toolbars, the Tools menu, window layouts, and add-in modules. This is further described in Chapter 73, Customizing Tau.

Local setup (UNIX)

Windows directory

In your home directory a new directory named **windows** will appear containing a set of files used to align the properties for IBM Rational Systems Tester between Windows and UNIX. The information stored in these files is not to be edited by the user.

Copy and Paste

Selection of text and using the middle mouse button for directly pasting it in another terminal window is supported only from the tabs in the Output window.

The dedicated buttons for Cut, Copy and Paste commands found on Solaris native terminals are not supported.

File dialogs

It is possible to filter displayed files in file dialogs with for example “*.u2” to show only this file type.

Generate support request

A tool for sending information to support can be opened by clicking **Generate Support Request** on the **Help** menu.

From the support tool it is possible to create screen dumps and video clips of Tau and send information directly to IBM Rational support.

Working with Workspaces

Workspaces - overview

A workspace is a personal working area, where you can add your projects. It allows you to organize your projects in a logical way. You can define a number of different workspaces, but you can only work in one workspace at a time. There is no upper limit regarding the number of added projects.

You cannot share your workspace with other users, since the content of your workspace may not coincide with the needs of other users.

Projects are stored with path names relative to the workspace. This allows you to move a workspace and all its contents from one location to another without losing any information.

The information included in a workspace is stored in a text file with the `.ttw` extension.

Create a new workspace

IBM Rational Systems Tester always starts with an empty active window. If you have not recently used workspaces, you may create a new one.

1. On the **File** menu, click **New**.
2. Click the **Workspaces** tab.
3. Type a name for your workspace.
4. Choose the location for your workspace. A folder with the same name as the workspace is created by default.
5. Click **OK**.

Open a workspace

Workspaces may be opened, closed, and saved, just as in any other standard application. Workspaces that you recently have worked with are available in a shortcut list. The list can contain up to eight different workspaces.

Open a workspace:

1. On the **File** menu, click **Open Workspace**.
2. In the Open dialog, select or browse to the file you want to open. Click **Open**.

Open a recently opened workspace

1. On the **File** menu, point to **Recent Workspaces**. A list of the eight most recent workspaces is displayed.
2. Select a workspace.

Save and close a workspace

- Save a workspace: On the **File** menu, click **Save Workspace**.
- To close a workspace: On the **File** menu, click **Close Workspace**.

Add a project to a workspace

There are two methods of adding a project to your workspace:

From the File View:

1. In the **File View**, right-click your workspace, and click **Insert project...** from the shortcut menu.
2. In the Open dialog box, select the project you want to add.
3. Click **Open**.

From the Project menu:

1. On the **Project** menu, click **Insert Project into Workspace...** The Open dialog box appears.
2. In the Open dialog box, select the project you want to add.
3. Click **Open**.

See also

"Create a new project with a new workspace" on page 20.

Working with Projects

Projects - overview

A project contains a number of references to source files that together with instructions on how to compile them produce a program or final binary files.

A project must be inserted in a workspace and you can work with many projects within the same workspace, allowing for diagrams and documents to be moved within and between projects. The same project can also be included in different users' workspaces. Projects that you add to a workspace can be located on other paths, on different drives, or directly in the root directory. This enables your team to work collectively with the same projects.

If there is no workspace open when you create a project, a directory for the workspace and a workspace file are also created. Alternatively, you may add a project to an existing, open workspace.

The files that are referenced in a project can be of any type and for convenience they can be organized in user-created folders.

A project is not individual and can therefore be shared between users. Some settings are global: if one user adds a file, it is added to the projects of other users as well. Some settings, such as which font to use, are not global.

The information included in a project is stored in a text file with the `.ttp` extension. Files included in a project are stored with relative paths in the project file.

Active project

When you add a project to a workspace, it will become the active project. All commands on the Project menu will be applied to the active project. If you have more than one project in the open workspace, you must actively choose which project that should be active in the Active Project list that is available in the toolbar.

Starting to work with projects

When using the tool for the first time, you can, for example, create a file first and later add this to a project. But the recommended workflow is to create a project in a workspace, and subsequently add files. By doing this, you are in a better position to control your project.

You can either create a new workspace or work with an existing one for each project. Files in a project may be stored locally, or on other locations. The objective is to offer you a working environment that optimally suits your, and your project's, needs.

As you create and modify a project, you can view the components included in the various views in the workspace window.

You have multiple choices to change and set your preferences, either on tool level or for your current project. It is for example possible to create custom toolbars, menu commands, and buttons.

Create a new project with a new workspace

1. On the **File** menu, click **New**.
2. Click the **Projects** tab.
3. Select the type of project you want to create.
4. Type a name for your project. Using the browse button, you can also change the location of the project. The option **Create new workspace** is selected by default.
5. Click **OK**.
6. If you want, you can change the project settings. Click the help button to receive more information about the settings.
7. Click **Next** and **Finish**.

Create a new project in an existing workspace

1. Open the workspace you want to add the new project to.
2. On the **File** menu, click **New**.
3. Click the **Projects** tab.
4. Select the type of project you want to create.
5. Type a name for your project and click **Add to current workspace**.
6. Click **OK**.
7. If you want, you can change the project settings. Click the help button to receive more information about the settings.
8. Click **Next** and **Finish**.

Insert an existing project in a workspace

A project is handled within the workspace it is located in. The workspace can be opened, closed, and saved, just as in any other standard application. A project file (.ttp) can be inserted in any workspace, not just the one it is created in.

To insert a Project in an existing workspace:

1. Go to the File View tab in the Workspace window.
2. Right-click the workspace icon, on the **shortcut** menu select **Insert File**. The Open dialog box appears.
3. Find the desired project, and click **Open**.

Note

To be able to view project files it is sometimes necessary to change the file filter in the Open dialog

See also

"Menu bar and toolbar" on page 10.

"Add a project to a workspace" on page 18.

Add files and folders to your project

Add files to your project

You can add any type of files to your project, including project and workspace files. A workspace added to a project is treated like a normal text file, without any functionality. Using drag and drop you can move your files to and from folders.

There are two ways of adding files to your project.

From the File View:

1. In the **File View**, right-click the project or a folder, and click **Insert files...** from the shortcut menu.
2. In the Open dialog box, select the files you want to add
3. Click **Open**.

From the Project menu

1. On the **Project** menu, point to **Add to Project** and click **Files**.
2. In the Open dialog box, select the files you want to insert
3. Click **Open**.

Note

When you add file from the project menu, you cannot decide the target folder. All files will be added at the root level of the project.

Open a recently accessed file

1. On the **File** menu, point to **Recent Files**. A list of the most recent files is displayed.
2. Select a file.

Add folders to a project

You can add folders to a project to logically organize your files. These folders are only defined in the project files – they are not represented in your file system. The file path in the operation system will remain unchanged.

When adding folders you can define a list of file extensions. These indicate what type of files that will be included in the folder. The list will be used as a filter when you add files to a folder. Only the files with the listed extensions will be displayed in the add file dialog by default.

There are two ways of adding a folder to your project:

From the File View:

1. In the **File View**, right-click the project, and click **New Folder** from the shortcut menu.
2. In the New Folder dialog box, type a name in the **Folder name** box.
3. Type one or more optional extensions in the **Folder extension** box in the form of `*.<extension>`. If you type more than one extension, separate them with a semicolon (;).
4. Click **OK**.

From the Project menu:

1. On the **Project** menu, point to **Add To Project** and click **New Folder**.
2. In the New Folder dialog box, type a name in the **Folder name** box.
3. Type one or more optional extensions in the **Folder extension** box in the form of `*.<extension>`. If you type more than one extension, separate them with a semicolon (;).
4. Click **OK**.

Activate a project

To enable any functionality for a project, it must be activated. Only one project in your workspace can be activated at a time. To activate a project do one of the following:

- In the **File View**, right-click the project. Click **Set as Active Project** from the shortcut menu.
- In the **Project** toolbar select the desired project in the **Active project** list.
- In the **Project** menu open the **Configurations** dialog and use the Set active button in the Configurations dialog.

Note

If you only have one project in your workspace, this project will automatically be set as active.

See also

"Project settings and configurations" on page 24

File and folder properties

You can view, and in some cases edit, properties for selected workspace items. Properties will only be available when a single item is selected in File View.

Properties for files, projects, and workspaces are view-only. Properties for folders are fully editable.

To view file and folder properties:

- In the **File View**, right-click an item and select **Properties**. You can also press ALT + ENTER.

Set or change folder properties:

1. In the **File View**, right-click a folder and select **Properties**.
2. In the Properties Editor box, change the **Folder name** and **File extensions**.
3. Optional: press ENTER to apply and verify changes.
4. Unless you want to view or change preferences for other workspace items, close the dialog box by clicking the close button.

To close the dialog box without saving any changes:

- Press ESC.

Create, open and close files

To create a new file:

1. On the **File** menu, click **New**.
2. Select the file type you want to create.
3. Specify File name and File location.
4. If you have an open project, you can decide if you want to include the file in the project.
5. Click **OK**.

To open a file:

1. On the **File** menu, click **Open**.
2. In the Open dialog box, select or browse to the file you want to open.
3. Click **Open**.

Or:

1. On the **File** menu, click **Recent Files**. A list of the 8 most recent files is displayed.
2. Select file.

To close a file:

- On the **File** menu, click **Close**.

Project settings and configurations

Project settings, available in the Settings dialog box, include options for analysis, code generation, building, execution, and logging.

When you change options in the Settings dialog box, they will be saved in the currently active configuration. A project may contain several configurations. You can Activate a project to set its configuration to be active in the Configurations dialog box, or in the corresponding lists in the toolbar.

Project settings

There are two ways of changing the settings for your project:

From the File View:

1. In the **File View**, right-click the project, and click **Settings** on the shortcut menu.
2. The Settings dialog box appears. Change the settings according to your needs.
3. Click **OK**.

From the Project menu:

1. In the Project toolbar select the desired project in the Active configuration list.
2. On the **Project** menu, click **Settings**.
3. The Settings dialog box appears. Change the settings according to your needs.
4. Click **OK**.

Project Configurations

To add a new configuration:

1. On the **Project** menu, click **Configurations**.
2. In the Configurations dialog box, click **Add**.
3. In the Add Configurations dialog box, type a name for the configuration, select an existing configuration to copy the settings from, and select which project the new configuration should be associated with.
4. Click **OK**.

To remove a configuration:

1. On the **Project** menu, click **Configurations**.
2. In the Configurations dialog box, select a configuration.
3. Click **Remove**.
4. Click **OK**.

Discovery based storage

Introduction

Discovery-based storage is necessary to support scenarios where the user does not wish to have one file (the project file) to have explicit knowledge of all files contained in a model.

Discovery Path

A discovery path, is a property of a project, and contains a list of locations in which IBM Rational Systems Tester will search for files. Searching is recursive.

This property is edited through a text field called “discovery location” on the folder “Discovery” property page.

A discovery location may be a URN reference or a relative path, or a full path to a u2 model.

- If a path is entered manually (without using the browse button) then the path will not be altered by IBM Rational Systems Tester.
- If the path is entered using the browse button the path will be adjusted with respect to the IBM Rational Systems Tester->Options->General->URN map. That is, if the file is located under a URN location, then a URN reference will be calculated and saved.
- If the path is entered using the browse button, and the path is not located under a URN location but is relative to the project, then this relative path is calculated and saved.

If a path is entered manually (without using the browse button) then the path will not be altered by IBM Rational Systems Tester.

However, if the path is entered using the browse button the path will be adjusted with respect to the IBM Rational Systems Tester->Options->General->URN map. That is, if the file is located under a URN location, then a URN reference will be calculated and saved.

If he path is entered using the browse button, and the path is not located under a URN location but is relative to the project, then this relative path is calculated and saved.

If the path is not in the same filesystem as the project then path is left as-is and saved.

A Shallow modifier may be added to a discovery location and has the same semantics as the `.u2shallow` file - see below.

Wild cards are not supported in names of discovery locations.

Resource Match Rules

Resource Match Rules are regular expressions which specify which files should be covered by Discovery-based storage.

The set of Match Rules may be either inclusive or exclusive.

The rules are editable through the “discovery file filter” text field on the folder property page.

The rules are a semicolon separated list of file matching patterns.

If the list is empty (or non-existent) then everything is included

The entries are applied in the order they appear in the list - first match wins.

Directory Match Rules

Directory Match Rules are regular expressions which specify which directories should be covered by Discovery-based storage.

The set of Match Rules may be either inclusive or exclusive.

The rules are editable through the “discovery directory filter” text field on the folder property page.

The rules are a semicolon separated list of directory matching patterns.

If the list is empty (or non-existent) then everything is included.

The entries are applied in the order they appear in the list - first match wins.

Directives

- Ignore Directives

  If a file named `.u2ignore` exists in a directory being searched, the directory and all files within it will be ignored.

  If a file named "`File.u2.u2ignore`" exists then this means that the file "`File.u2`" will be ignored.

  If a file named directory.u2ignore exists then this means that the directory "directory" will be ignored.

- Shallow Directives

  If a file named `.u2shallow` exists in a directory being searched, all subdirectories within it will be ignored.

  If a file named `directory.u2shallow` exists in a directory being searched, then all subdirectories within "`directory`" will be ignored.

- Discover Directive

  A file with the extension `.u2discover` may be used to provide an additional list of discovery locations, as well as a resource inclusion and exclusion list.

Note

*Files with the extension *.u2x are always ignored.*

- .u2discover file format

```
#This is a .u2discover file
DiscoveryPath:
#Do not recurse into subdirectories of the following
location
Loca*tion1(Shallow)
Loca*tion2
ResourceInclusion(inclusive):
*.foo
*.boo
DirectoryInclusion(exclusive):
#Skip CVS subdirectories
CVS
```

The Shallow modifier can be added to discovery locations. This is interpreted in the same way as the `.u2shallow` file extension (see above).

A `ResourceInclusion` section may be qualified by either “inclusive” or “exclusive” (default is “inclusive” if not present). Lines starting with '#' are ignored. All whitespace-only lines are ignored. Sections can come in any order. Sections must start in first column. All sections are optional.

A `DirectoryInclusion` section may be qualified by either “inclusive” or “exclusive” (default is “inclusive” if not present).

Case is not significant in keywords and directives.

I18N

The `.u2discover` file is assumed to contain valid UNICODE characters in a UTF-8 stream. BOM or similar magic numbers are ignored.

Interpretation Of Directives

Directory/Resource Inclusion rules are applied per directory/resource. The inclusion rules present in the project are added to the match rules first, then when each directory is visited list of match rules present in the `.u2discover` file (if present) is appended. When a directory/resource is applied to the match rules, the first match wins. The additional `.u2ignore`/`.u2shallow` files are used to override any of the previously named matches.

Generic SCC/Synergy

The Generic SCC/Synergy integration works for files that are discovered in the same way as for normal files.

Making Discovered Files Explicit In A Project

To make a discovered file explicit in a project, simply drag it from its discovery folder (in the File View) and drop it on the project node.

Allowing An Explicit File To Be Discovered Instead

Delete the file reference from the File View (choose “remove elements” from the subsequent pop-up). And make sure that the directory, in which the file is contained, is in the `DirectoryPath` property. Save and reload the project.

Filter syntax

The only file matching wildcard supported is “*”.

Manual Rediscovery

It is possible to force a discovery after the project is loaded. The project context menu has an entry called “Discover Files”. This only adds new files, it does not remove files that no longer exist.

Project Reports (TTCN-3 projects only)

You have different alternatives to create reports from your project:

- By generating an HTML report on all language elements, that is TTCN-3 objects, in your active project.

  Click the **Report on TTCN-3 objects** button in the toolbar. An HTML viewer is activated, allowing you to navigate through the objects through different lists:

  – Modules
  – Imports
  – Constants
  – Ports
  – Components
  – Groups
  – Simple types
  – Structured types
  – Templates
  – Signatures
  – Test cases

- By generating a list of all TTCN-3 objects in the output window, sorted by when it appears in the source.

  Click the **List Objects** button in the toolbar. All objects are displayed in the Objects tab in the output window, with information on line number, what kind of object it is, and path.

- By using the Script Wizard.

  Click the **Script Wizard** button in the toolbar. An HTML viewer is activated, allowing you to navigate in an UML representation of your project, and to create Tcl scripts to extract information from this model.

See also

How to Use Help

This help file includes basic and advanced topics covering the supported functionality.

Additional product documentation is available in the section "Additional Resources" on page 827. There you can find tutorials, language descriptions, the installation guide and links to external sites, for instance the IBM Rational Systems Tester Support site.

Additional documentation in Adobe PDF format is also available on the installation CD.

Navigate in the help file

The help file contains functionality that helps you to easier find the information that you are looking for:

Search

To perform a full-text search:

1. In the help viewer, click the **Search** tab.
2. Type your search string in the **Type in the word(s) to search for** field. You may use regular expressions, operators, and nested expressions when searching.
3. Optionally, you may check some of the following options: **Search previous result**, **Match similar words** and **Search titles only**.
4. Click **List Topics**.
5. To open a topic, double-click the topic in the Select topic list or select a topic and click **Display**.

Example 1:

To search for words beginning with “link”, type the following in the search field:

```
link*
```

Search highlighting

The words that you are searching for are highlighted on all pages where they are found. If you want to, you can turn off this functionality.

Turn off search highlighting:

1. In the help viewer, click the **Options** button and then click **Search Highlight Off**.
2. If you have already performed a search, click the **Display** button in the help viewer and the search highlighting disappear.

The search highlighting functionality is now turned off until you enable it again.

Turn on search highlighting:

1. In the help viewer, click the **Options** button and then click **Search Highlight On**.
2. If you have already performed a search, click the **Display** button in the help viewer and the search highlighting re-appear.

The search highlighting functionality is now turned on until you disable it again.

Index

To see the list of index entries, select the Index tab. To find the entry you are looking for, type the first letters of the word or scroll the list. To view the entry, double-click the entry or select the entry and click **Display**.

Locate search or index results

When you are using the search or the index functionality, the topic you are looking for will be displayed in the right-hand window. To locate where this topic is listed in the table of contents, click the **Locate** button. This allows you to easily find related topics or to learn where this topic is located the next time you are looking for it.

Bookmark topics in the help file

If you know that there are topics that you will refer to often or that there are topics that you consider important for your work, you can bookmark them as you would do in a regular web browser.

Bookmark a topic:

1. Find your topic using the Contents, Index or Search tabs.
2. Click the **Favorites** tab. The name of the topic is listed in the **Current topic** field.
3. Click **Add**. The topic is now displayed in the topics list.

Print help topics

You can print a single topic or you can select to print several topics within the same chapter.

Print an active topic:

- Right-click the displayed topic in the right-hand window and click **Print**. The print dialog opens.

Print a single topic from the table of contents

1. Right-click the topic window in the table of contents and click **Print**. The **Print Topics** dialog opens.
2. Click **Print the selected topic** and click OK. The Print dialog opens.

Print multiple topics:

1. Right-click a book icon in the table of contents and click **Print**. The **Print Topics** dialog opens.
2. Click **Print the selected heading and all sub-topics** and click **OK**. The print dialog opens.

Search syntax in help

The help viewer supports full text search, and you can search for any combination of letters (a-z) and numbers (0-9). Words like “the”, “a”, “and”, “but”, are reserved and cannot be searched for. In addition, you cannot search for punctuation marks such as colon (:), semicolon (;), hyphen (-) and period (.).

You can group search elements by using quotes and parenthesis.

Match similar words

The **Search** tab in the help viewer includes a **Match similar words** option. If you select this, you will be able to find all occurrences of a word, including common suffixes. For example, if you search for “run”, the words “run”, “running”, and “runner” will be found, but not “runtime”.

Regular expressions

The following regular expressions may be used when searching the help:

- * for matching 0 or more characters.
- ? for matching 1 characters.
- A string within quotes (“ab cd”) for matching the string literally.

Search for this:	Type this in the search field
Topics containing “analyze”, “analysis”, “analyses”, “analyzed”, and “analyzing”	analy*
Topics containing “analyzer” and “analyzed”, but not “analyze” or “analyzers”	analyze?
Topics containing the literal phrase “analyze and generate”	“analyze and generate”

Operators

You may use the following operators to refine a search in the help: AND, OR, NOT, and NEAR. The search string is evaluated from left to right. See table below for examples:

Search for this:	Type this in the search field
Topics containing both “workspace” and “file”	workspace AND file **or** workspace & file **or** workspace file
Topics containing either “workspace” or “file”	workspace OR file **or** workspace | file
Topics containing “workspace” but not “file”	workspace NOT file **or** workspace | file
Topics containing “workspace” and “file” close together, that is “workspace” within 8 words of “file”	workspace NEAR file
Topics containing “workspace” but not “file”, or topics containing “workspace” but not “directory”	workspace NOT file OR directory

Nested expressions

By using parentheses, you may nest expressions to perform a complex search in the help. An expression within parentheses will be evaluated first, before the rest of the search expression. Expressions may not be nested more than 5 levels.

Search for this:	Type this in the search field
Topics containing “workspace” without either of “file” or “directory”	workspace NOT (file OR directory)
Topics containing “workspace” with and “file” and “project” close together; or topics containing “workspace” with “directory” and “project” close together	workspace AND ((file OR directory) NEAR project)

TTCN-3 Projects

The chapters that are listed under TTCN-3 projects describe functionality that is exclusive to TTCN-3 projects.

2

Supported Languages

This section describes the main languages involved in the IBM Rational Systems Tester tool set.

About TTCN-3

From a syntactical point of view, TTCN-3 is very different from TTCN-2. However, much of the well-proven basic functionality of TTCN-2 has been retained, and in some cases enhanced.

- Core format is a text-based notation
- Core can be viewed as text or in various presentation formats
  - Tabular format for conformance testing
  - Other standardized formats in the future
  - Proprietary formats

These are the objectives for developing TTCN-3:

- Modernization – technology has changed since TTCN-2 was first developed
- Wider scope of application – should be applicable to many kinds of test applications not just conformance (development, system, integration)

- Harmonization – should be the first choice for test specifiers, implementers, and users both for standardized test suites and as a generic solution in product development.

The main capabilities of TTCN-3 are listed below:

- Dynamic concurrent testing configurations
- Various communication mechanisms (synchronous and asynchronous)
- Data and signature templates with powerful matching mechanisms
- Specification of encoding information
- Display and user-defined attributes
- Test suite parameterization
- Test case control and selection mechanisms
- Assignment and handling of test verdicts
- Harmonized with ASN.1
- Different presentation formats
- Well-defined syntax, static semantics and operational semantics.

Migrating test suites from TTCN-2 to TTCN-3

IBM Rational Systems Tester provides command line converter `t2tot3` that translates test suites written in TTCN-2 into TTCN-3 language. Conversion rules are based on ETSI TR 101 874 “TTCN-2 To TTCN-3 Mapping“ document. The usage is following:

```
t2tot3 <root 'mp' file> [<'mp' and 'asn' files>]
```

Converter accepts root test suite module file in ‘mp’ format and additional test suite ‘mp’ and ‘asn’ files (if any). Converter always produces two files, one TTCN-3 file and one ASN.1 file (which may be empty).

Note

All TTCN-2 and ASN.1 modules are merged into one TTCN-3 and one ASN.1 module during conversion.

Some TTCN-2 constructs may not be translated automatically and require user intervention. Converter prints messages regarding every such encountered construct. Additionally it places ‘TODO’ comments into generated TTCN-3 module that help user to locate places that probably require manual changes to the generated code.

Support for TTCN-3 Edition 3 extended log() statement

IBM Rational Systems Tester supports an extended log statement that was introduced in TTCN-3 Edition 3. Unlike the log statement in previous editions of TTCN-3, which allowed to log only constant character strings, the new log statement may log almost every object of the test suite: constants, variables, module parameters, functions returning values, component references, timers, port references etc.

For example:

```
var integer arr[3] := {1,2,3};
var integer i;
for (i := 0; i <3; i := i+1) {
  log("The value of arr[",i,"]=",arr[i]);
}
```

The output will be:

- The value of arr[0]=1
- The value of arr[1]=2
- The value of arr[2]=3

Support for TTCN-3 Edition 3 component extension

IBM Rational Systems Tester supports the component extension mechanism introduced in TTCN-3 Edition 3. The component extension is a mechanism for inheriting structure of a component type.

For example:

```
type component BaseComp {
        port MyPortType P ;
}
type component ExtendedComp extends BaseComp {
        const integer N := 110;
}
```

This is equivalent to:

```
type component ExtendedComp {
        port MyPortType P;
        const integer N := 110;
}
```

When defining component types by extension, there is no name clash between a definition taken from the parent type and a definition added in an extended type, i.e. there shall not be a port, variable, constant or timer identifier that is declared both in the parent type (directly or also by means of extension) and the extended type.

Following example is incorrect:

```
type component BaseComp {
  timer T
}
type component InterimComponent extends BaseComp{
  port MyPortType P[3] ;
}
type component ExtendedComp extends InterimComp {
  var integer T; //compiler error, T is defined in
BaseComp
}
```

It is allowed to have one component type, extending several parent types, in one definition. This is specified as a comma-separated list of types in the definition:

For example:

```
type component MyComp extends BaseCompA, BaseCompB,
BaseCompC {
  ...
}
```

Besides providing a well-structured mechanism for reuse of the component type declarations, the component extension mechanism clarifies the component compatibility rules with regards to “runs on” clause. If a function F() is declared to run on component `CompF`, then it is allowed to execute F() on any component that directly or indirectly extends `CompF`. For component type compatibility, this means that a component of type `ExtendedCT`, which extends `BaseCT`, is compatible with `BaseCT`. Test cases, functions and alt steps specifying `BaseCT` in their “run on” clauses can be executed on `ExtendedCT`.

Note

Component extension may be treated as macro substitution of base component body into extended component. Therefore all referenced entities (e.g. types) should be visible inside the module that contains extended component (see example below).

Example 2

```
module BASE {
```

```
    const integer gloC := 1

    type component CT {
      const integer compC := gloC
    }
}

module INH {
    import from BASE {type CT} //gloC is not imported
    type component MTC extends CT
    {
      // ERROR "gloC not defined"
    }
}
```

This example is not correct because gloC constant is not imported into INH module and hence effective definition of MTC component references undefined gloC object. To fix this example import statement should either include gloC constant or import all definitions from BASE module.

About ASN.1

ASN.1 is the acronym for Abstract Syntax Notation One, a standardized notation for describing structured information, and in particular, data types.

The first appearance of ASN.1 was made public in 1984 by ITU-T. Since then, it has been standardized internationally and widely used in the specification of communication protocols and in many other standard protocols. With ASN.1, it is possible to view and describe the relevant information and its structure at a high level with no need to be concerned with low-level information representation.

Abstract syntax and transfer syntax are the main concepts that are covered in the ASN.1 language definitions. The abstract syntax is the form that would normally appear in a standard and is used to describe protocol data unit and other data structures at the level of human readability. The transfer syntax defines the specific encoding rules that are used in a real implementation of a protocol that converts the abstract form to the stream of bits that are sent over a communication channel. Typical encoding rules are Basic Encoding Rules (BER) and Packed Encoding Rules (PER).

Supported Character Encoding

In the list below, you find the character encoding supported. The support means that a file may be edited and saved with the same encoding as it has been opened with. However, when you create a new file, the encoding will always be ANSI. For all files, you may select an encoding when saving as.

When you search by using **Find in files**, Unicode files will not be recognized.

The following character encoding is supported in Rational Systems Tester editors:

- ANSI
- UniBE with and without BOM
- UniLE with and without BOM
- UTF-8 with BOM
- UTF-8 without BOM

Note

UniBE = 2-byte Unicode Big Endian
UniLE = 2-byte Unicode Little Endian
BOM = Byte Order Mask

TTCN-3 Language Support

IBM Rational Systems Tester supports TTCN-3, as defined in ETSI ES 201 873-1 V2.2.1.

TTCN-3 Limitations

- Only 64 bit size signed integers are supported.
- Predefined sizeoftype function may not be used for imported ASN.1 types.
- Language specification for the TTCN-3 module is used only to resolve backward compatibility issues between TTCN-3 language editions.
- ‘char’ datatype still is a predefined type and may not be redefined. Type compatibility rules between char and charstring datatypes conform to TTCN-3 Edition 2.2.1.

TTCN-3 Constructs Not Supported

- Object Identifier – is supported as datatypes but not as qualifier for import declarations

Backward compatibility in TTCN-3

TTCN-3 Edition 3 is almost fully backward compatible with the TTCN-3 Edition 2. However there is a couple of issues that you need to be considered:

- The behavior of “oct2str” and “str2oct” predefined functions has been changed. New functions “oct2char” and “char2oct” have been introduced in Edition 3 that perform exactly the same as “oct2str” and “str2oct” in Edition 2. In other words “oct2str” in Edition 2 equals to “oct2char” in Edition 3 and “str2oct” in Edition 2 equals to “char2oct” in Edition 3.
- “str2int” function reports runtime error if provided string doesn’t represent valid integer value (e.g. contains characters). In Edition 2 “str2int” returned zero in erroneous situations without reporting errors.

Rational Systems Tester provides easy and convenient way of resolving above-mentioned issues. It’s based on explicit module language specification for a TTCN-3 module.

Example 3

```
module My_Edition_3_module language “TTCN-3:2005”
{ … }
```

“TTCN-3:2001”, “TTCN-3:2003” and “TTCN-3:2005” are three predefined language specifications that correspond to Edition 1, 2 and 3 of the TTCN-3 language.

By default (without language specification) Rational Systems Tester TTCN-3 compiler treats TTCN-3 files as defined in Edition 3 and it reports warnings for all places with changed behavior. You may explicitly define language for the module and resolve the ambiguities. If you define language as “TTCN-3:2005” then compiler stops reporting warnings while keeping behavior for this module. If you specify “TTCN-3:2003” language then behavior of “oct2str”, “str2oct” and ”str2int” functions will conform to Edition 2.

ASN.1 Language Support

IBM Rational Systems Tester supports a subset of the basic ASN.1 notation as defined in X.680-X.683, X.690, and X.691.

ASN.1 EXTERNAL type support

Rational Systems Tester supports ASN.1 EXTERNAL type as defined in ASN.1 1990 standard. ASN.1 module that contain definitions with EXTERNAL type and that is imported into TTCN-3 module should specify compiler option: `STANDARD-VERSION=1990`.

Example 4(Using EXTERNAL type in ASN.1 module) ----------

```
--$STANDARD-VERSION=1990
MyASNModuleDEFINITIONS AUTOMATIC TAGS ::=
BEGIN
        MyExternal ::= EXTERNAL
END
```

ASN.1 1990 defines EXTERNAL type as:

```
EXTERNAL ::= [UNIVERSAL 8] IMPLICIT SEQUENCE
{
   direct-reference      OBJECT IDENTIFIER OPTIONAL,
   indirect-reference    INTEGER OPTIONAL,
   data-value-descriptor ObjectDescriptor OPTIONAL,
   encoding CHOICE {
           single-ASN1-type [0] ANY,
           octet-aligned    [1] IMPLICIT OCTET STRING,
           arbitrary        [2] IMPLICIT BIT STRING,
   }
}
```

This definition is translated into TTCN-3 type as defined in ASN.1->TTCN-3 translation rules in TTCN-3 standard.

Although compiler generates BER encoders and decoders for EXTERNAL type user has to provide user-defined encoder and decoder for "encoding" field of EXTERNAL type (since ANY type may hold value of arbitrary ASN.1 type not known to the decoder). This inline CHOICE type is explicified by the compiler and usually receive name "EXTERNAL_expl_encoding".

However this name should not be used to check whether certain type pointer corresponds to the "encoding" type. Such check should be performed by obtaining type pointer to the EXTERNAL type and then getting type pointer for "encoding" field.

For example, following code may be used to set user-defined encoder and decoder for explicified "encoding" type. Usually it's placed in codec setup function.

Example 5(Setting encoder and decoder for EXTERNAl type) ----------

```
if (strcmp(t3rt_type_definition_name(type, ctx),
"EXTERNAL") == 0)
{
  t3rt_type_set_encoder(t3rt_type_field_type(type,
t3rt_type_field_index(type, "encoding", ctx), ctx),
encode_external, ctx);

  t3rt_type_set_decoder(t3rt_type_field_type(type,
t3rt_type_field_index(type, "encoding", ctx), ctx),
decode_external, ctx);
}
```

When implementing encoder and decodeer for EXTERNAL type it’s necessary to keep in mind following issues.

The task of the user-defined encoder is to encode selected CHOICE alternative into bit string as specified in BER encoding rules. The trivial encoder obtains the value of selected alternative and invokes encoder for it.

Example 6(Example encoder for the EXTERNAL type) ----------

```
t3rt_codecs_result_t encode_external(t3rt_value_t value,
t3rt_binary_string_t encoded_data, t3rt_context_t ctx)
{
  //get alternative value
  t3rt_value_t union_value = t3rt_value_union_value(value,
ctx);
  //invoke encoder for it
  return t3rt_encode(union_value, encoded_data, ctx);
}
```

If "single-ASN1-type" alternative is selected then "encode_external" function simply calls generated encoder for the type that is the type of the value stored in "single-ASN1-type" field.

The task of the user-defined decoder is to decode received bitstring that conforms to BER encoding rules. Prior to calling decoder TTCN-3 runtime system instantiates new value of "encoding" type and sets certain alternative depending on the service information stored inside the received message. Selected alternative may be queried using `t3rt_value_union_index` function. `t3rt_value_union_value` function may be used also.

Decoder may instantiate new value instance of "encoding" type and overwrite those instantiated by the runtime system.

Since ANY type may represent ASN.1 value of arbitrary type decoder needs some hints on what is the actual type of transmitted value. This information is usually passed in the "direct-reference" or "indirect-reference" fields. Decoder (as well as encoder) may query this information. This may be done by obtaining pointer to the enclosing "EXTERNAL" value and then querying "direct-reference" and "indirect-reference" fields using `t3rt_field_by_name` function. Enclosing "EXTERNAL" value may be obtained using `t3rt_value_parent` function.

Example 7(Example decoder for the EXTERNAL type) ----------

```
t3rt_codecs_result_t
decode_external(t3rt_binary_string_iter_t * encoded_data,
  t3rt_type_t type,
  t3rt_alloc_strategy_t strategy,
  t3rt_value_t * decoded_value,
  t3rt_context_t ctx)
{
  t3rt_value_t value;
  t3rt_codecs_result_t result;
  t3rt_type_t actual_type;
  t3rt_value_t parent_value;

  //get reference to the enclsosing "EXTERNAL" type value
  parent_value = t3rt_value_parent (*decoded_value, ctx);
  //calculate type of the transmitted value
  actual_type =
getTypeOfTransmittedValue(t3rt_field_by_name(parent_value,
"direct_reference", ctx);
  //invoke decoder
  result = t3rt_decode(encoded_data, actual_type, strategy,
&value, ctx);
  if (result == t3rt_codecs_result_succeeded_c)
  {
    <fill "decoded_value" with "value">
  }
  return result;
}
```

Note

User-defined encoder and decoder should conform to BER encoding rules.

ASN.1 Limitations

- An ASN.1 file may only contain one module.
- An ASN.1 file must be encoded in ASCII.
- PER is not supported for EXTERNAL type.
- EXTERNAL type values defined inside ASN.1 module are not translated into TTCN-3 module.
- “data-value-descriptor” field in EXTERNAL type may contains only ASCII characters and its length is limited to 1000 symbols.

Compatibility with TTCN-2

TTCN-2 test suites (for example created in IBM Rational TTCN Suite) cannot be opened directly in IBM Rational Systems Tester. You may use converter supplied with Rational Systems Tester to translate test suites from TTCN-2 to TTCN-3 language.

3

Editing Abstract Test Suites

Using IBM Rational Systems Tester, you can create, edit, and analyze your abstract test suites (ATS).

You organize your projects, files, and project configurations in a workspace. A workspace consists of a workspace file in a workspace directory. This file describes the workspace and its contents. When a project is created for the first time, a directory for the workspace and a workspace file is also created. In addition, a makefile configuration file is also created.

A project may include various file types, such as TTCN-3, ASN.1, HTML, and Tcl. When editing, you may perform advanced find and replace operations in a single file or multiple files, including using regular expressions and fast incremental searching. It is possible to navigate through sections of TTCN-3 code by matching group delimiters, by using bookmarks, or by using the Go To dialog box. You can also use the Script Wizard to navigate in a UML representation of a TTCN-3 project – presented as HTML files – and to create Tcl scripts to extract information from this model.

Editing Abstract Test Suites

When editing TTCN-3 files, you have access to several editing features, such as syntax coloring, entity lists, and bookmarks to simplify your test suite development. In addition to this, you can also edit several other file types such as ASN.1, text, C, and makefiles.

Most of the editing procedures should seem familiar if you have used other Windows-based text editors. For example:

- Cutting, copying, pasting, and deleting text
- Undoing and redoing editing actions
- Using the drag-and-drop feature

Hint

While editing, you can right-click to display a shortcut menu of frequently used commands such as cut, paste, and copy.

When you cut text from a file, the text is removed from your file and placed on the clipboard. When you delete text from the file, the text is removed from your file, and the clipboard is not used. Commands that use the clipboard overwrite whatever was previously placed onto the clipboard by other commands or other applications.

General Editing

Save Files, Projects, and Workspaces

Save changes while editing:

- On the **File** menu, click **Save**,

  or

  Click **Save** in the toolbar.

Save all files in the workspace:

- On the **File** menu, click **Save All**,

  or

  Click **Save All** in the toolbar.

Save as different file type:

1. On the **File** menu, click **Save as**.
2. In the Save as box, enter the new file name.
3. Click **Save**.

Save workspace:

- On the **File** menu, click **Save Workspace**.

Note

This procedure saves your workspace, that is your `.ttw` *file, not the projects, or files, in your workspace. The* `.ttw` *file contains the name of the workspace, and links to all files in the workspace.*

Save project options in an Options file:

1. On the **Project** menu, click **Options** and then click **Save as**.
2. In the Save as box, enter name for the for the options file (*.tot).
3. Click **Save**.

Find and Replace

To find a text string in the current file:

1. Move the insertion point to where you want to begin your search. The location of the insertion point selects a default search string.
2. On the Edit menu, click Find.

   or

   On the toolbar, click the Find button.
3. In the Find dialog box, type the search text or a regular expression.
4. Select search options: case-sensitive, match whole words, or use regular expressions.

   If you use regular expressions, be sure the Regular expression check box is selected. You can also use the drop-down list to select from a list of up to 16 previous search strings.
5. Start the search by clicking Find Next or Mark All.

6. The Find dialog box disappears when the search begins. To repeat a find operation, use the shortcut keys or toolbar buttons.

   If you selected Mark All, all lines containing one or more occurrences of found text strings will be marked with an unnamed bookmark. You may navigate between these by pressing **F2** (forward) and **SHIFT+F2** (backward). The bookmarks disappear when you close the file.

To find a text string in several files:

1. On the Edit menu, click Find in Files

   or

   on the toolbar, click the Find in Files button.
2. In the Find in Files dialog box, find the search text or use a regular expression.

   You may click the button to the right of the Find what box to display a list of regular search expressions. When you select an expression from this list, the expression is substituted as the search text in the Find what box.
3. Select files or file types to search in.
4. Select a folder to search in.
5. Optionally, you may click the More button to be able to specify additional folders to search in.
6. Optionally, you may select Output to pane 2 if you want to display the search result in the Find in Files 2 tab in the Output window.

   If not selected, the result will be displayed in the Find in Files 1 tab. This means that you may keep one previous search result when you perform a new search.
7. Click Find. The search result will be displayed in a Find in Files tab.
8. In the Output window, double-click a found occurrence to navigate to the source.

To find a string using incremental search:

1. Move the insertion point to where you want to begin your search.
2. Press **CTRL + I**. The status bar reads Incremental Search.
3. Type a search string. As you type each character, the first matching string in your file will be selected.

4. Press **CTRL + I** to go to the next match or press **CTRL + ALT + I** to go to the previous match.
5. Press **ESC** to end the search.

To replace text:

1. Move the insertion point to where you want to begin your search. The location of the insertion point selects a default search string.
2. On the Edit menu, click Replace.
3. In the Find what dialog box, type the search text or a regular expression. You can also use the drop-down list to select from up to 16 previous search strings.

Note

If you use regular expressions, be sure the Regular expression check box is selected.

4. In the Replace with box, type the replacement text.
5. Select Find options.
6. Start the search by clicking Find Next, Replace, or Replace All.

To find a line number or named bookmark:

1. On the Edit menu, click Go To.
2. In the Go To what dialog box, select the type of item you want (line or bookmark).
3. Specify line number or bookmark name.
4. Click Go To.

Hint

Click the pushpin button to keep the dialog box open, on top of other windows.

Note

If the Go To what item is undefined, the Go To button appears dimmed, for example if you have not defined any bookmarks.

Cut, Copy, or Paste Text

1. Select the text you want to cut or copy.
2. On the Edit menu, click Cut or Copy.

3. The cut or copied text is placed onto the clipboard and is available for pasting.
4. Move the insertion point to any window where you want to insert the text.
5. On the Edit menu, click Paste.

Hint

You can also use Drag-And-Drop Editing.

Delete Text

1. Select the text you want to delete.
2. On the Edit menu, click Delete

   or

   On the toolbar, click the Delete button.

The deleted text is not placed onto the clipboard, and cannot be pasted.

Drag-And-Drop Editing

To move text using drag-and-drop editing:

1. Select the text you want to move.
2. Drag the selected text to the new location.

You may cancel the drag-and-drop editing by right-clicking.

To copy text using drag-and-drop editing:

1. Select the text you want to copy.
2. While pressing CTRL, drag the selected text to the new location.

Undo and Redo Editing

To undo editing:

- On the Edit menu, click Undo
- Press CTRL+Z.

To redo editing:

- On the Edit menu, click Redo
- Press CTRL+Y.

Note

It is only possible to redo editing that was previously undone.

Select All in an Active File

Select all in an active file, either:

- On the Edit menu, click Select All
- Press CTRL+A
- Right-click in a file and click Select All.

Analyze Source Code

Set Analyzer options:

1. On the **Project** menu, click **Settings** and then click the **Misc.** tab.
2. In **Analyzer settings**, specify number of errors allowed.
3. Click the **TTCN-3** tab.
4. Select a Generation type:
   - Only Syntax Analysis

     Performs a syntax analysis only, without generating any code.
   - Run Analyzer but do not generate code

     Performs a full semantic and syntactic analysis, without generating any code.
5. Click the ASN.1 tab, and change settings for Generation type, as above.
6. Click OK.

To analyze a test suite:

- Click the Analyze button in the toolbar.
- On the Project menu, click Analyze.

To cancel the analysis:

- Click the Stop action button in the toolbar
- On the Project menu, click Stop action.

Colors and Fonts

You may change the colors of text and background and change the font for different windows and different file types.

It is also possible to use different colors for various code elements, such as keywords, comments, and operators. This syntax coloring gives you a visual guide to the structure and state of your code.

Note

Different file types, for example TTCN-3 or ASN.1, may use different syntax coloring to simplify editing in several parallel editors.

You find options for changing colors and fonts in the Options dialog box. It is also possible to make changes directly in `.ini` files.

Entity List

When you edit a TTCN-3 file, the command CTRL+SPACEBAR displays an entity list functioning as an advanced completion feature that speeds up your editing. The list contains possible entities next to an icon, indicating type of entity. When you select an entity from this list, the appropriate entity is inserted into your source code.

The entity list box is displayed below the edited line in order not to block the view of the editing area.

As you type extra characters, an incremental search in the entity list is performed. If the invocation of the command results in a non-ambiguous selection of an entity, that entity is added directly.

The functionality is context sensitive. When you apply this globally, it lists all the entities at that level. If you prefix an identifier with a module name, it only lists entities from that module (and imported by that module).

Type Definitions for User-Defined Entities

When editing, you can also type a "." (PERIOD) after a type. This command displays a list of all defined fields in this type.

Scope Line

When editing TTCN-3, a red scope line to the left of your code is displayed, showing the active language element your are editing in.

A missing scope line indicates that an error in the code within the active segment has been detected. If no correction is made, most of the editing assistance, such as entity lists, cease to function.

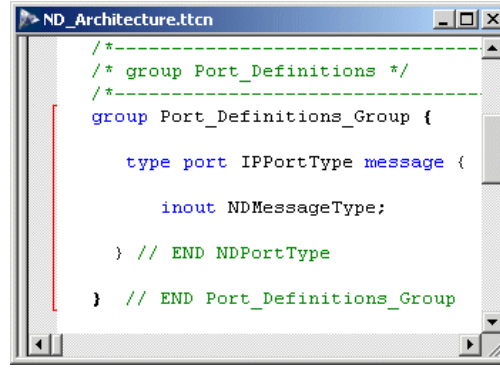

Figure 2: Example of a scope line

Bookmarks

Bookmarks make it possible to navigate back and forth between different places in the same TTCN-3 file. This is useful when you edit large and complex files.

You may place named bookmarks at the cursor position in the current file by using the Bookmark dialog box. Later, you may navigate between named bookmarks – independently of the current file – by using the Bookmark or Go To dialog boxes.

You may toggle an unnamed bookmark on or off by pressing **CTRL+F2**. Press F2 to navigate between unnamed bookmarks in the current file.

Set a named bookmark:

1. Move the insertion point to the line and column where you want to set a named bookmark.
2. On the **Edit** menu, click **Bookmarks**.
3. In the **Name** box, type a name for the bookmark.
4. Click **Add** to add the named bookmark to the list of bookmarks and to set the bookmark at the insertion point.
5. Click **Close**.

Set an unnamed bookmark:

1. Move the insertion point to the line where you want to set a bookmark.
2. Press **CTRL+F2**.
3. The line is marked in the left margin.

Remove a named bookmark:

1. On the Edit menu, click Bookmarks.
2. Select the name of the bookmark to remove. You may select several names.
3. Click Delete.
4. Click Close.

To remove an unnamed bookmark:

1. Move the insertion point to anywhere on the line containing the unnamed bookmark.
2. Press **CTRL+F2**.

To go to a named bookmark:

1. On the **Edit** menu, click **Bookmarks** or **Go To**.
2. Select the name of the bookmark to jump to.
3. Click **Go To**.

Note

If the file containing the bookmark is not currently open, it will be opened.

To go to the next unnamed bookmark:

- Press **F2**

To go to the previous unnamed bookmark:

- Press **SHIFT+F2**

Go To

The Go To dialog box allows you to jump quickly to:

- Named bookmarks: Enter a bookmark name or select one from the drop-down list.
- Lines: Enter the line number.

Go To Definition

IBM Rational Systems Tester provides “Go To Definition” feature in the text editor. For every TTCN-3 reference (constant, variable, type reference, template, function, etc.) it is now possible to jump to the declaration of this reference.

Right-click on any word in a TTCN-3 source file. In the shortcut menu you can choose “Go To Definition”, which will bring you to the declaration of the word under the cursor if this word corresponds to a valid TTCN-3 declaration visible at this point.

Outline View

When editing TTCN-3 files, you also have access to an outline view. This view displays your code linearly, whereas the Structured View is sorted by language elements.

You find the outline view on the left hand pane of the editor. It is minimized by default.

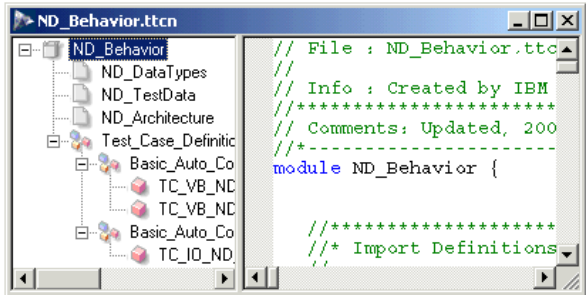

Figure 3: The outline view

TTCN-3 Quick Information

Pop-Up Parameter Information

When you edit a TTCN-3 file, a pop-up parameter information box is displayed on mouse-over hovering. This feature provides help when a call to a given function or method is to be performed. This pop-up box is displayed below the currently edited line to minimize interruption when you are editing.

The parameter in bold indicates the next parameter required as you type the function or test case call.

Pop-Up Type Information

In a very similar way as for functions and their parameters, the type definition for a given identifier in a pop-up window can be displayed. When writing test suites you define sequences of send and receive messages, and this information speeds up development work by minimizing navigation back and forth among your files.

Identifier information is displayed when you place the pointer over an identifier.

Note

This only works for identifiers that are of a defined, given type.

Search

There are two methods for searching: full string search and incremental search. With full string search, you specify the entire search string before the search begins. With incremental search, the search is performed as you type the string.

Full string search

With the advanced find and replace capabilities, you can search for text in a single source file or in multiple files. You can search for literal text strings or use regular expressions to find words or characters. A regular expression is a search string that uses special characters to match a text pattern in a file. You can use regular expressions, including tagged regular expressions, with the Find, Find in Files, and Replace commands.

Besides the Find dialog boxes, you may also enter your search string in the Find text field and click the Find or Find in Files buttons.

Incremental search

Incremental search is performed in the status bar. Start incremental search by pressing **CTRL + I**. Then type a text string.

Note

Regular expressions are not supported with incremental searching.

Analysis Settings

When editing TTCN-3 and ASN.1, you can perform an advanced analysis to ensure correctness, either for a whole file, or for a specific module.

You perform a so called pure analysis session with no code generation as result as part of your editing, when clicking Analyze on the Project menu.

A simple analysis is also performed continuously during editing, with possible errors indicated by a missing scope line.

When your test suite is analyzed, the result is presented in the Output window. There is a reserved tab, Tester Analyzer, for this information and it is possible to navigate from this to the TTCN-3 or ASN.1 source. You do this by double-clicking the analysis message, or by right-clicking and selecting Locate in the drop-down menu.

Correctly written code always pass the analysis, but if your code includes features not supported by the Compiler, a warning message about lack of functionality is displayed. Show-stoppers are reported as errors.

You can select different levels of analysis. One level is to perform simple syntax analysis only, and another level is to perform a full semantic and syntactic analysis.

Note

These options are set in the Settings dialog box, in the TTCN-3 and ASN.1 tabs respectively.

Reports

All reports from the analysis session are displayed in the Output window. Errors or warnings can be set to be written to file.

Hint

It is possible to set to ignore warnings.

You can specify the amount of messages that is allowed before terminating the analysis (default setting is 100). This is set in the Misc. tab, in the Settings dialog box.

Converting TTCN-3 to XML

TTCN-3 files may be converted to XML, optionally including a DTD. This conversion is a command-line feature that you invoke with the command `t3doc`.

The syntax for a `t3doc` command is the following:

```
[-dtd] [-m] [-o <output filename> | -stdout] <input filenames>
```

All options will be described below, but you may also enter `t3doc -?` to get help on options.

Option	Explanation
`-dtd`	Includes a DTD first in the XML file
`-m`	Generates an XML file for each TTCN-3 file. If this option is not set, the output for each TTCN-3 file will be combined into one single XML file. In any case, the information will be the same.
`-o <output_filename> | -stdout`	`-o <output filename>` specifies the name of the generated XML file. If this option is set, it overrides the option -m. If this option is not set, the XML file will have the same name as the first TTCN-3 file. `-stdout` generates XML to stdout instead of to a file.
`<input filename>`	The names of the TTCN-3 files that are to be converted, separated by spaces.

Example 8: ----------

To generate one XML file with a specified filename, with a DTD, and with several TTCN-3 files as input:

1. Change directory to \bin in the Rational Systems Tester installation directory.
2. Enter the following:

```
t3doc -dtd -o my_xml_output my_test1.ttcn my_test2.ttcn
my_test3.ttcn
```

Example 9: ----------

To generate an XML file for each TTCN-3 file:

1. Change directory to \bin in the Rational Systems Tester installation directory.
2. Enter the following:

```
t3doc -m my_test1.ttcn my_test2.ttcn my_test3.ttcn
```

4

Creating an ETS

Before you are able to build an executable test suite (ETS), you will have to generate C code from your abstract test suite (ATS), by using the IBM Rational Systems Tester Compiler. The Compiler is composed of separate, but closely integrated, internal tools. One for analysis to ensure correctness, and one for C code generation. You are able to set different options for compilation, both on a global, project level, and on a module specific level.

The generated code contains ANSI-C representations of the ATS. By using a third-party ANSI-C compiler, you build the C files, together with integration files, into an executable test suite. When building, you need to specify a makefile. This file is also created during code generation.

See also

"Compiler Settings Overview" on page 70 on different options for generating code from TTCN-3 and ASN.1.

"Makefile Generation" on page 79

"Build Settings" on page 115 in Chapter 5, *Building*

Compiler Settings Overview

All settings for the Compiler are set on a global level, that is for all modules and files in your active project. In the Settings dialog box, you change options for your TTCN-3 and ASN.1 files.

All options are set in the General Module Settings in the TTCN-3 and ASN.1 tab respectively, where you select your settings through standard check boxes and drop-down menus. But you also have the possibility to state exceptions for individual modules in the Module Specific Settings, where you may enter option flags in a text field.

It is not possible to state exceptions on file level.

When generating code, the Compiler also uses settings in the Build tab, namely TRI integration, Root module, and Generate makefile.

You can also run the Compiler from the command-line.

TTCN-3 Settings

General settings are selected in the TTCN-3 tab in the Settings dialog box.

Note

The browser pane in the Settings dialog box is provided to give you an overview of the active project.

The following options are available:

Verbosity Level

All messages are displayed in the TTCN to C Compiler tab in the Output window.

- **Report errors only**

  Errors indicate that you will not be able to generate any code.
- **Report errors and warnings** (default)

  Warnings allows code generation, but it is not recommended to proceed unless you correct your test suite.
- **Report errors, warnings and compiler progress**

  Allows you to inspect the code generation progress, in addition to errors and warnings.

- **Full report**

  Maximum verbosity.

Generation Type

- **Only Syntax Analysis**

  Performs a syntax analysis without generating any code.
- **Run Analyzer but do not generate code**

  Performs a full semantic and syntactic analysis, without generating any code.
- **Run Analyzer and Generate Code**

  Performs a full semantic and syntactic analysis and generates code.

Output Directory

Allows you to set a different output directory. The default output directory is your working directory.

TTCN-3 module specific settings

Allows you to state exceptions to the general settings for specified modules. These settings override the general settings.

Note

*There are also settings relevant for code generation in the Build tab.**"Build Settings" on page 115*.

See also

"Running the Compiler from the Command-Line" on page 73

ASN.1 Settings

General settings are selected in the ASN.1 tab in the Settings dialog box.

Note

The browser pane in the Settings dialog is provided to give you an overview of the active project.

The following alternatives are available:

Verbosity Level

All messages are displayed in the TTCN to C Compiler tab in the Output window.

- **Report errors only**

  Errors indicate that you will not be able to generate any code.
- **Report errors and warnings** (default)

  Warnings allows code generation, but it is not recommended to proceed unless you correct your code.
- **Report errors, warnings and compiler progress**

  Allows you to inspect the code generation progress.
- **Full report**

  Maximum verbosity.

Generation Type

- **Only Syntax Analysis**

  Performs a syntax analysis without generating any code.
- **Run Analyzer but do not generate code**

  Performs a full semantic and syntactic analysis, without generating any code.
- **Run Analyzer and Generate Code**

  Performs a full semantic and syntactic analysis and generates code.

Use target code identifier prefix

Allows you to specify a prefix in free text, to avoid type or identifier name clashes in the generated code.

Use output base name for files

Allows you to specify a name in free text for split generated files. If no option is specified, the module name is used.

Output Directory

Allows you to set a different output directory. The default output directory is your working directory.

ASN.1 module specific settings

Allows you to state exceptions to the general settings for specified modules. These settings override the general settings.

Note

*There are also settings relevant for code generation in the Build tab.**"Build Settings" on page 115*.

See also

"Running the Compiler from the Command-Line" on page 73

Running the Compiler from the Command-Line

You may invoke the Compiler from the command-line with the command `t3cg`. All options available are described below. This information is also available from the command-line. In the bin directory, type `t3cg -h` or `t3cg -?`

Most options are also applicable to ASN.1 module specific settings and TTCN-3 module specific settings in the Settings dialog box.

t3cg command syntax and semantics

A `t3cg` command may consist of the following:

```
[Global Options] -M [-A [Module Options] <ASN.1 module names>
-T [Module Options] <TTCN-3 module names>] -F [-A [File
Options] <ASN.1 file names> -T [File Options] <TTCN-3 file
names>]
```

Module specific settings syntax and semantics

The ASN.1 module specific settings and TTCN-3 module specific settings in the Settings dialog box allows you to state exceptions for specified modules. These settings override the general settings. A module specific setting consists of the following:

```
[Module Options] <Module names>
```

Note

It is not possible to specify a TTCN-3 module name in the ASN.1 tab, or the other way around.

Global options

Note

All options are case sensitive.

[Global Options]	Explanation
`-? | -h`	Displays help on options
`-V`	Displays version information.
`-t`	Specifies that a generated makefile will be adapted to TRI integration.
`-m <number>`	Specifies the number of analysis errors that should be allowed before compilation stops. Default <number> is 100.
`-f <filename>`	Overrides default mcfg file name. By default compiler looks for mcfg file with hardcoded name that depends on platform used (Windows, Linux, Solaris). Use this switch to specify arbitrary mcfg file.
`-k`	Generates a makefile with extension `.new`, without overwriting an existing. (This option requires the `-r` option)
`-K`	Generates a makefile and overwrites an existing, after saving a copy of it. (This option requires the `-r` option)
`-E`	Forces generation of a file containing “main“ function. This switch may be used to generate “main” function without generating makefile. When makefile generation is enabled it overrides MAINFILE option specified in mcfg file.
`-B`	Skips rebuilding of unchanged ASN.1 and TTCN-3 files.

[Global Options]	Explanation
`-i`	Enables TCI support. This option is necessary when using TCI functions in the integration (e.g. when implementing coders using TCI CD interface).
`-R`	Generates code and makefile with support for Rational Systems Tester GUI test management. Requires “-i“ switch.
`-p`	Specifies project directory. This option is necessary when relative paths are used to specify output directories.
`-G`	Generates code and makefile for debugging.
`-O`	Skips code generation for unused (not referenced) module-level declarations.
`-D`	Forces strict TTCN-3 standard compliance. Without this option compiler enables several deviations (e.g. nested comments) from TTCN-3 standard.
`-l <number>`	Specifies the maximum length of description that is generated for every template. Default <number> is 100.
`-r <name of module>`	Specifies a root module. If no root module is specified then first compiled module is assumed to be root.
`-T [Module Options]`	Specifies that [Module Options] will be globally applied to all TTCN-3 modules.
`-A [Module Options]`	Specifies that [Module Options] will be globally applied to all ASN.1 modules.

Module options

Note

All options are case sensitive.

```
-M [-A [Module Options] <ASN.1 module names> -T [Module
Options] <TTCN-3 module names>]
```

`-M` means beginning of modules section.

`-A` means the following `[Module Options]` should be applied to `<ASN.1 module names>`.

`-T` means the following `[Module Options]` should be applied to `<TTCN-3 module names>`.

`<ASN.1 module names>` means names of ASN.1 modules separated by space.

`<TTCN-3 module names>` means names of TTCN-3 modules separated by space.

Note

A [Module Option] set on module level overrides a [Module Option] set on global level.

[Module Options]	Explanation
-v <number>	Sets verbosity level according to <number>. <number> is the verbosity level value, where 1 means Report errors only 2 means Report errors and warnings (default) 3 means Report errors, warnings and compiler progress 4 means Full report.
-I <number>	Specifies that the analysis should detect non-initialized variables in your TTCN-3 code, according to <number>. <number> is how serious non-initialized values should be treated, where 0 means this type of analysis is switched off 1 means warnings will be issued (default) 2 means errors will be issued, which prevents code generation.
-w	Weak typing analysis: The type checker will not check for the violation of subtype restrictions.
-s	Sets generation type to Only syntax analysis.

[Module Options]	Explanation
-a	Sets generation type to Run analyzer but do not generate code.
-c	Sets generation type to Run analyzer and generate code. This is the default behavior.
-d <path>	Sets output directory to the directory specified in <path>. Default is your project directory. If you put the path within quotes, you will avoid interpretation of special characters in a command shell.
-o <file name>	Specifies the generated file base name. Default is the module name. If you put the file name within quotes, you will avoid interpretation of special characters in a command shell.
-C <ASN.1 codec>	Specifies the type of codec generated for ASN.1 modules. Valid values are “none”, “all”, “no-translation”, “BER:1997”, “BER:1997-length-form-3”, “PER-BASIC-ALIGNED:1997”, “PER-BASIC-UNALIGNED:1997”. Default value is “all”.

File options

Note

All options are case sensitive.

```
-F [-A [File Options] <ASN.1 file names> -T [File Options]
<TTCN-3 file names>]
```

`-F` means beginning of files section.

`-A` means the following `[File Options]` should be applied to all modules in `<ASN.1 file names>`.

`-T` means the following `[File Options]` should be applied to all modules in `<TTCN-3 file names>`.

`<ASN.1 file names>` means names of ASN.1 files separated by space.

`<TTCN-3 file names>` means names of TTCN-3 files separated by space.

[File Options]	Explanation
-e <character encoding>	Specifies the character encoding used in the file. <character encoding> may one of the following: ASCII UTF-8 UCS-2 UCS-2LE UCS-2BE UNKNOWN FORMAT (default) If you do not specify a file type, the Compiler will try to determine it automatically. If you accidently specify the wrong format, the compilation will stop.

Example 10: Using t3cg from the command-line ––––––––––––––––––––

In this example you wish to stop the Compiler after more than 30 errors and change the output directory for code generated for all TTCN-3 modules in all TTCN-3 files. The generation type for the TTCN-3 module `module_a` should be syntax analysis only, for `module_b` it should be syntax and semantic analysis, while for `module_c` it should use the default generation type (analyze and generate code). All three modules are contained in a single file, modules.ttcn. The file my_asn_types.asn contains an ASN.1 module that should be analyzed and for which code should be generated.

1. Change directory to \bin in the Rational Systems Tester installation directory.
2. Enter the following:

```
t3cg -m 30 -T -d "/another/directory" -M -T -s module_a
-a module_b module_c -F -T -e ASCII my_modules.ttcn -A -
e my_asn_types.asn
```

––

Example 11: Using the Settings dialog box for specifying module specific

options ––

In this example, you wish to set verbosity level to Report errors only and generation type to Run Analyzer but do not generate code for the module TestModule1, and verbosity level Full report and generation type Only syntax analysis for the module TestModule2.

1. In the Specific TTCN-3 Module Settings text box, enter:

   ```
   -a -v 1 TestModule1 -s -v 4 TestModule2
   ```

2. Click OK.

––

Compiler Output

The Compiler will generate ANSI-C files, header files, and a makefile, according to your settings in the Settings dialog box, in the TTCN-3, ASN.1, and Build tab. The makefile generation is also based on a makefile configuration file.

Error messages and warnings will be produced according to the following format:

```
<absolute pathname>(<line>):Error/Warning Message
```

Other messages, like progress, will be given in free form.

All output is displayed in the TTCN to C Compiler tab, in the Output window.

Makefile Generation

When a makefile is generated, information is gathered from the Compiler and from user-specific input that is provided in an OS specific makefile generation configuration (mcfg) file.

There are two choices for mcfg file creation. By default mcfg file is created automatically when you invoke compiler according to the options that you specify in the project settings. All new projects use this type of configuring makefile generation. Another choice is to manually define mcfg file using template provided in the installation.

In the `/include` directory in the installation directory, there is a makefile configuration file, `make_win.mcfg` (Windows) or `make_solaris.mcfg` (UNIX) available. If you choose manual project configuration then you need to copy this file to your project directory. By default the name of mcfg file used for makefile generation is determined using OSTYPE environment variable. If the name of mcfg file that you are using doesn't correspond to the OSTYPE value then you need to provide mcfg file name on the "Build" tab in the project settings.

You can edit the makefile configuration file to create a makefile that is suitable for your needs. Instructions how to do this are included in the makefile configuration file itself.

Important!

Avoid editing makefile configuration file in the `Include` *directory, as it's used as a template by the New Project wizard.*

During code generation, the makefile will be generated in your project directory.

Defining makefile configuration manually

Ensure that you have a copy of makefile configuration file in your project directory and on the **Build** tab in the project settings "use mcfg file" choice is selected. If the name of the mcfg file differs from the default one then provide it in the entry field.

Before code generation, you have to specify which TRI integration to use. This is done in the **Settings** dialog box, in the **Build** tab. The TRI integration alternatives are:

- Rational Systems Tester TRI
- Example
- None

The selection you make affects the contents of the generated makefile.

Whether you are generating a makefile based on TRI, the example integration or none, you will probably need to modify either the makefile configuration file or the actual makefile before building the ETS. Both files are located in the projects directory.

When you open the makefile configuration file, you will be presented with information how to edit the file and what values that can be used for the different settings.

The modifications you can do include:

- Name of the generated makefile
- Name of third-party compiler
- Third-party compiler location
- Linkers and linker flags, when applicable
- Output name for the ETS (default name is set to the same as your root module)
- Whether to create a main function C file
- Locations of ASN.1 and RTS libraries
- Control extensions on generated files
- Type of the ETS (executable or static library)

It is possible to add C files necessary for your integration.

Example 12: Specify extensions on generated files ––––––––––––––––––

It is possible to specify arbitrary extensions that will be used in the generated makefile for object files and for executable files.

Use `CC_OBJ_EXT` in the configuration file (`.mcfg`) to specify the object file extension and `LD_EXE_EXT` to specify the extension for the executable file.

Windows:

```
CC_OBJ_EXT = obj
LD_OBJ_EXT = exe
```

UNIX:

```
CC_OBJ_EXT = o
LD_OBJ_EXT =
```

If these makefile variables are not specified the makefile generator will use default extensions, according to current platform.

––

Automated makefile configuration

You may automate makefile configuration by delegating this issue to the IBM Rational Systems Tester framework. In this case before invoking compiler Rational Systems Tester generates temporary makefile configuration file using the options that you specify in the project settings and files attached to the workspace.

The main advantage of using automated makefile generation is that C and LIB files attached to the workspace are treated as adapter files and are automatically put into generated makefile. You may exclude certain files from build be right clicking on them and choosing **Settings** in the context window. Go to **Build** tab and check **Exclude this file from Build.** This tells Rational Systems Tester to ignore the file during makefile generation. Additionally Rational Systems Tester automatically links libraries from the dependent projects (if the project target is a library).

Open **Build** tab in the **Settings** dialog, choose “Use Makefile Options“ choice and press “Makefile Options...“ button. The opened window contains several tabs that represent all settings inside a makefile configuration file. So defining settings here is exactly the same as manually defining them in the mcfg file.

You may load contents of existing makefile configuration file by pressing the button located on the top of the window. Locate mcfg file and press “Ok”.

Note

When you load makefile configuration file Rational Systems Tester uses “TRI integration” value on Build tab to select certain section inside the mcfg file.

Hint

In the Build tab in the Settings dialog box, you have the option either to overwrite an existing makefile when generating a new, or to save the new makefile with the extension `.new`.

Make build settings location independent

To make for easier migration of projects between computers IBM Rational Systems Tester offers <TESTER_DIR> template and TELELOGIC_TESTER_DIR environment variable.

You may use <TESTER_DIR> template for the settings inside Makefile Options dialog in the project settings to specify that actual Rational Systems Tester installation directory should substitute this template during build process. This allows using one and the same project on different computers that have Rational Systems Tester installed in different locations without changing project settings. All new projects use this mechanism

TELELOGIC_TESTER_DIR environment variable may be used to override <TESTER_DIR> default substitution. If this variable is set in the environment its value is used instead of actual Rational Systems Tester location. If Rational Systems Tester determines that this variable is not set then it defines this varible when executing external tools (like “make”) during build process. User provided makefiles may rely on this variable.

Note

<TESTER_DIR> template may be used only inside Makefile Options dialog. Using it for other project settings will not perfrom any substitutions.

Target project type

Project may be built into an executable or a static library. Usually projects are built into test executable. If you have several projects that share common TTCN-3 code then you may decide to create multi-project workspace, move common code to the separate project and built it into library. Other projects links this library during the linking stage.

By default project generated makefile creates executable. In order to create the library do the following:

- Set `BUILD_LIBRARY` switch in makefile configuration file to `yes`.
- Open **Makefile Options** on **Build** tab in project settings and select **Build Library** choice**.**

After all rebuild your project.

Note

Only static link libraries are supported, you may not create dynamic link libraries.

When you compile project into library and specify dependencies between this project and another projects that are built into executable you need to keep in mind following issues:

- All TTCN-3 and ASN.1 files inside a library project are added to the list of compiled files in the depending executable project with “analyze only“ switch.
- When you use automated makefile configuration libraries created during library project builds are automatically linked to the depending executable project. You do not need to care about include and library directories and paths to the library files. When configuring makefile generation manually it’s your task to link this libraries correctly.

Generate a makefile

To generate a makefile from the user interface

To generate a makefile from the user interface, follow the instructions below:

1. When manually defining makefile configuration a makefile configuration file must be located in the project directory. For automated makefile configuration all adapter files should be either attached to the project or specified in the **Makefile Options** in the project settings.
2. Optionally, add the makefile configuration file to your project. From the **Project** menu, point to **Add To Project** and click **Files**. Find the file and click **Open**. This means that you can edit the makefile configuration file directly from the Files tab in the Workspace window.
3. From the **Project** menu, click **Settings**. In the dialog box that opens select the **Build** tab.
4. Make sure that the **Generate Makefile** check box is selected.
5. In the **Root module** field, type the name of your root module.
6. Click **OK**.
7. From the **Project** menu, click **Compiler**

   or

   click the **Compiler** button on the toolbar.

To generate a makefile from the command-line

To generate a makefile from the command-line, follow the instructions below:

1. A makefile configuration file must be located in the project directory. If you did not copy the makefile configuration file when you created your project, copy the file `make_win.mcfg` (Windows) or `make_solaris.mcfg` (UNIX) from the `Include` directory to your project directory.
2. On the command-line type:

   ```
   t3cg -k -r <name of root module>
   ```

See also

"Running the Compiler from the Command-Line" on page 73

ASN.1 in TTCN-3 modules

TTCN-3 provides a clean interface for using ASN.1 version 1997 in TTCN-3 modules. TTCN-3 is fully harmonized with ASN.1, which may optionally be used with TTCN-3 modules as alternative data type and value syntax.

Example 13: ----------

```
import from MyASN1Module language "ASN.1:1997" {
        type MyASN1Type
}
```

Rational Systems Tester uses separate compilers to analyze ASN.1 files and generate BER and PER encoders and decoders for ASN.1 definitions. The set of ASN.1 tools in Rational Systems Tester include:

- `mkprefil` - tool for defining mapping between ASN.1 module names and ASN.1 files
- `casn` - ASN.1 analyzer
- `asn2c` - ASN.1 to C compiler and BER encoder/decoder generator
- `asn2per` - ASN.1 PER encoder/decoder generator
- `popvar` - tool for post-processing ASN.1 variables

If Rational Systems Tester project contains ASN.1 modules then TTCN-3 compiler automatically invokes all needed ASN.1 tools and generates make-file for building executable test suite (ETS) no matter whether Rational Systems Tester project is compiled from GUI or in batch mode. All ASN.1 tool invocations are done transparent to the user but he/she can use project setting to provide options for ASN.1 files.

ASN.1 files may be analyzed separately by invoking **casn** in batch mode. **casn** requires that each file contains exactly one ASN.1 module. The **casn** checks the files and produces compiler listing files that contain information about the modules.

How to run casn:

The command line for running **casn** is as follows:

```
casn.exe \
        [-<option>=<value>]... \
        [+<option-file>]... \
        <prefix>...
```

The <prefix> arguments specify names of the ASN.1 module definition files to be compiled.

A prefix is a file name without a suffix. For example if an ASN.1 module resides in the file “MyAsnFile.asn” then file prefix that must be specified for **casn** is “MyAsnFile”.

By default the maximum length of a file prefix is five characters. It is possible to raise the limit by using the **casn** option `PREFIXLEN`.

Example 14: ----------

```
casn.exe -prefixlen=9 MyAsnFile
```

In case when Rational Systems Tester project contains several ASN.1 files it’s necessary to provide mappings between ASN.1 module names and ASN.1 files. This may be done either manually by specifying `PREFIX` and `PREFIXLEN` command line parameters or automatically with the help of `mkprefil` tool.

Example 15: ––

The files TopLayer.asn, MidLayer.asn and BottomLayer.asn contain definitions of the ASN.1 modules `TopLayerPDUs`, `MiddleLayerPDUs` and BottomLayerPDUs.

To compile files with manually specified mappings execute:

```
casn.exe -PREFIX=TopLayerPDUs:TopLayer -
PREFIX=MiddleLayerPDUs:MidLayer -PREFIX=BottomLayerPDUs:BottomLayer
-PREFIXLEN=11 TopLayer MidLayer BottomLayer
```

––

Option file that contains mappings between ASN.1 module references and file names may be automatically generated by using `mkprefil`:

```
mkprefil.exe -option-file=prefix.opt TopLayer MidLayer BottomLayer
```

`mkprefil` will generate `prefix.opt` file containing:

```
# Automatically generated by MKPREFIL ...
-PREFIX=TopLayerPDUs:TopLayer
-PREFIX=MiddleLayerPDUs:MidLayer
-PREFIX=BottomLayerPDUs:BottomLayer
-PREFIXLEN=11
```

Then ASN.1 files my be compiled by executing:

```
casn.exe +prefix.opt TopLayer MidLayer BottomLayer
```

The `casn` tool generates two files, one that contains compiler listing and another that contains binary representation of an ASN.1 module definition. If there are errors or warnings during the compilation then consult the listing file. The syntax tree file is intended only for back-end tools (`asn2c` and `asn2per`).

ASN.1 Encoding

The Compiler generates separate files containing BER and PER encoders variations for types that have an encoding attribute. All variations supported are generated if any of the encoding attributes is specified. The encoding rules and variations supported are:

- BER (definite length, indefinite length)
- PER (octet aligned, octet unaligned)

The BER definite length form, is chosen to be short if the length of the element is included in the range (0...127). Longer elements are encoded using the long form of length encoding. The decoders support both forms. Only basic PER is supported, not canonical PER.

Encoder usage from TTCN-3 attributes

The corresponding encoding attributes are:

- “BER:1997”, with variant: “length form 3"
- “PER-BASIC-ALIGNED:1997”
- “PER-BASIC-UNALIGNED:1997”

The BER attribute used without variant, defaults to definite length encoding. BER definite length encoding length field variations are selected automatically as stated above. Variants other than “length form 3” are not supported.

Encoding attributes are supported only per type. Using an encoding attribute when importing an ASN.1 type results in the corresponding encoder being used in run time. Field encoding attributes are not supported.

Encoder usage at run time

You can give the following command-line switches to the runtime system to override encoding to all ASN.1 types used in the executable modules:

- t3asn.<module_name>.BER.DEFINITE-LENGTH
- t3asn.<module_name>.BER.INDEFINITE-LENGTH
- t3asn.<module_name>.PER.BASIC-ALIGNED
- t3asn.<module_name>.PER.BASIC-UNALIGNED

They are boolean options and should be used as follows:

Example 16: ----------

To set types defined in a module called MESSAGES to use BER definite length encoding, type the following:

```
-t3rt "-confbool t3asn.MESSAGES.BER.DEFINITE-LENGTH true"
```

The option for BER indefinite length corresponds to the combination of attributes “BER:1997” and “length form 3”.

The command-line switches is further explained in “Command-Line Syntax” on page 122 in Chapter 6, *Execute Tests*.

Encoding open types

An encoder must be registered for an open type in run time initialization functions. The encoder must perform the following:

- Take the correct alternative from the TTCN-3 run time value.
- Get the encoder for the alternative type.
- Call the encoder function for the union alternative value and encoded data

The decoder performs the same sequence in reverse order, adding identification of the message:

- Create the open type value.
- Find out the type of the message.
- Get the decoder for the message from the run time type.
- Call the decoder.
- Set the union value to the alternative.

Using encoding attributes for a module other encoders are accessible from type descriptors.

Set Compiling Options

1. On the **Project** menu, click **Settings**.
2. Click the **TTCN-3** tab.
3. Set Verbosity Level.
4. Set Generation Type.
5. Set Output Directory.
6. If desired, specify TTCN-3 module specific settings.
7. Click the **ASN.1** tab.
8. Set the same settings as described for the TTCN-3 tab.
9. Select Use target code identifier prefix, when applicable.
10. Select Use Output Base Name for Files, when applicable.
11. If desired, specify ASN.1 module specific settings.

12. Click the **Build** tab.
13. Select a TRI Integration:
    - Rational Systems Tester TRI
    - Example
    - None
14. Specify Root module.
15. Select Generate makefile.
16. If you generate a new makefile, and there is already a generated makefile in the same directory, select whether it should overwritten or not.
17. Click OK.

Compile Test Suites

To compile test suites:

- On the Project menu, click Compiler

  or

  click the Compiler button in the toolbar.

To cancel compilation:

- Click the Stop action button in the toolbar

  or

  on the Project menu, click Stop action.

Industrial Standard Test Suites

SIP test suite

Compilation of ETSI Session Initiation Protocol (SIP) abstract test suites is supported. To create an SIP executable test suite one should:

- Create empty Rational Systems Tester project
- Add SIP TTCN-3 files to the project
- Provide TRI integration and encoders/decoders implemented either using TCI CD or proprietary IBM Rational codec interface.
- Compile and build project.

In order to compile a SIP test suite it is necessary to make the changes described below to the SIP ttcn-3 files.

The TTCN-3 analyzer will report three semantic errors. These errors concern the instantiation of templates that contain matching symbols. According to the TTCN-3 standard this is not allowed.

The erroneous operations are:

- `valueof(Route_l_4)` in the test case `SIP_CC_PR_MP_RQ_V_045`. An error is reported since the `Route_l_4` template contains an "`*`" symbol in its definition.
- `valueof(Contact_RD_Multi_s)` in `ptcRDMultiRegistration` function and `SIP_CC_RD_CE_V_004` testcase. Error is reported since `Contact_RD_Multi_s` template contains "`ifpresent`" operation in its definition.

To fix these errors you can create a copy of the templates for `Route_l_4` and `Contact_RD_Multi_s`, then replace any matching symbols with the appropriate value expressions and use the modified templates in the above mentioned "`valueof`" operations.

Assuming that `Route_l_4_value` is a copy of `Route_l_4` and `Contact_RD_Multi_s_value` is a copy of `Contact_RD_Multi_s, then the` "`*`" symbol in `Route_l_4_value` may be substituted with "`omit`" and "`ifpresent`" in `Contact_RD_Multi_s_value` may simply be deleted.

IPv6 test suite

The current Rational Systems Tester IPv6 test suite is based on the IPv6 draft version published September 9, 2005

To create an IPv6 executable test suite one should:

- Create empty Rational Systems Tester project
- Add IPv6 ttcn-3 files to the project
- Provide TRI integration and encoders/decoders implemented either using TCI CD or proprietary IBM Rational codec interface.
- Compile and build project.

Besides the IPv6 abstract test suite there exists a base TTCN-3 library that is included in the IPv6 test suite but also is available as a separate package.

It is necessary to declare a constant in the module `LibCommon_Sync`.

```
const charstring c_prSyncPoint := "T3 sync point";
```

This is due to the fact that the separate package contains one undefined constant (`c_prSyncPoint`) that may lead to some compile errors (the full IPv6 test suite compiles successfully)

TCI interface

IBM Rational Systems Tester supports standard TTCN-3 Test Control Interface (TCI). The TCI consists of several parts (sub-interfaces) providing control over different aspects of test execution and IBM Rational Systems Tester supports the majority of them. Using the TCI it is possible to:

TCI TM

Manage test execution using TCI TM (obtain information on the structure of test system, start/stop test cases and control parts, define module parameters).

TCI CD

Implement and plug-in encoders or decoders using TCI CD (access defined types in test system include “encode” attribute, create and manipulate with values).

TCI TL

Retrieve information about test execution using TCI TL logging (log exhaustive set of events generated during test execution). TCI TL has been introduced in TTCN-3 Version 3.0.

Test Executable build requirements

IBM Rational Systems Tester provides built-in test management. Besides driving test execution using RTS functions (in particular t3rt_run_test_suite()), it is possible to do similar things using either the TCI functions (`tciStartControl()` and `tciStartTestCase()`) or by means of the Rational Systems Tester GUI. This affects how a test executable is built.

When working with the Rational Systems Tester GUI it is sufficient to set the desired way of managing test execution in the project settings on Build tab:

- Static using RTS function
- Static using TCI functions
- Dynamic using Rational Systems Tester GUI

When building a test executable manually from the command line it is necessary to keep in mind the following:

The TCI requires that additional information is generated by the compiler. The TCI support in the compiler is switched on using the “-i” command line parameter. This will generate the additional files ts_modules.c and ts_modules.h, a special template main program and a special makefile. The template main program provides an example of how to select a root module and how to start the control part from it. The makefile with TCI support contains the additional rules to build the file ts_modules.c and link it with the TCI support library libt3tci.lib into a test executable.

Before using any of the TCI functions it is necessary to initialize the TCI interface using `tciInit()` function. The prototype is similar to the prototype of the main() function. `tciInit()` returns true if it succeeds and false otherwise. The TCI interface is finalized by applying the `tciFinalize()` function. For example the main() function for the test executable may start with the following lines:

```
main(int argc, char *argv[]) {
  if (tciInit(argc, argv)) {
    tciSetRoot("my_root_module");
    tciStartControl();
    ...
  }
  ...
  tciFinalize();
}
```

Note

It is recommended to initialize the TCI inside main() function, as the TCI should be initialized outside any calls to the runtime system functions (including t3rt_run_test_suite).

In order to control the test execution through the GUI it is necessary to follow some additional requirements for the compiler invocation. The compiler should be invoked with “-i -R” command line options. This will generate a special main program, that should not be changed, and a special makefile. The main function in this case consists of only one line:

```
main(int argc, char *argv[]) {
  tciStartTestsuiteServer(argc, argv);
}
```

Test management using the IBM Rational Systems Tester GUI requires two additional libraries are linked with the test executable. Besides libt3tci.lib that is the main TCI support library it is necessary to link also libt3tcite.lib and the system RPC (Remote Procedure Call) communication library, which is different for Windows and Linux/Solaris platforms. Windows RPC library is rpcrt4.lib whereas Linux and Solaris RPC library is librpcsvc.a. Under

Windows the test executable should be linked with libt3tci.lib, libt3tcite.lib and rpcrt4.lib libraries (and of course all other required TTCN-3 libraries like for example libt3rts.lib).

Functions to be implemented by the user

Similar to the TRI in the TCI interface you have to define a set of functions that provide the test executable ways to obtain information (e.g. a module parameter value) or notify the user about different events (runtime error, end of test execution etc.). The templates for the whole set of these functions are defined in t3tci_template.c file that is found in the installation directory.

It is not necessary to implement all (or even any) of these functions. For example, to manage test execution from IBM Rational Systems Tester GUI there is no need to make changes to the provided t3tci_template.c file at all. However, in order to plug standardized encoders and decoders, it is necessary to change the `tciEncode()` and the `tciDecode()` functions. To catch runtime errors it is necessary to change the implementation of the gatherer function.

The build rules file, t3tci_template.c, is automatically put into the generated makefile if the TCI/GUI test management is selected in the project settings or if the TCI support is switched on in the compiler. It is possible to define another file with the set of these functions by modifying the project configuration (`mcfg`) file.

Note

All files that use the TCI functions should include the header file "tci.h". This file is provided in the installation structure.

Using TCI encoders and decoders

IBM Rational Systems Tester provides a standard interface to implement and plug in encoders and decoders through the TCI CD interface. The implementation of encoders and decoders is based on Type and Value interfaces. These provide an extensive set of functions to manipulate with built-in and user-defined types and to create and change values. The Type and Value interfaces are described in the TTCN-3 TCI standard. A declaration with short comments may also be found in the header files tci_types.h and tci_values.h, located in the installation directory.

The top function for the encoder is `tciEncode()`. This function is called by the runtime system each time it is required to encode a value. This function returns encoded binary strings that later are sent through communication channels.

The top function for the decoder is `tciDecode()`. This function is called by the runtime system each time it is required to decode a binary string. The runtime system also passes to `tciDecode()` the decoding hypothesis (expected type reference), that may be used to determine an encoding rule, if contents of received message are not in it self sufficient to choose the decoding rule.

After implementing `tciEncode()` and `tciDecode()`, coders have to be plugged into the runtime system. TCI encoders and decoders are plugged similarly into encoders and decoders that are implemented using proprietary IBM Rational type and value interfaces. This allows for reuse of existing `codec` implementations while using the TCI TM interface to manage test execution.

TCI encoders/decoders are plugged in using wrapper functions provided in runtime system t3rt_tci_encode() and t3rt_tci_decode(). The following code describes how a TCI `codec` implementation may be plugged into the runtime system. This code should be linked into the test executable.

```
static void tci_codec_init_function(t3rt_context_t ctx) {
}
static void tci_codec_setup_function(t3rt_type_t type,
t3rt_context_t ctx) {
  t3rt_type_set_encoder (type, t3rt_tci_encode, ctx);
  t3rt_type_set_decoder (type, t3rt_tci_decode, ctx);
}
void t3ud_register_codecs (int argc, char * argv [], t3rt_context_t
ctx) {
  t3rt_codecs_register(&tci_codec_init_function,
&tci_codec_setup_function, ctx);
}
```

Note

Sometimes it is necessary to allocate for temporary memory in the `tciEncode()` *and* `tciDecode()` *functions (e.g. for the binary string returned by* `tciEncode()`*). This is done by the* `tciMemoryAllocate()` *function. The memory will be automatically de-allocated by the runtime system when it is not needed anymore.*

Providing values of module parameters

The TCI interface provides the ability to set the values of module parameters using the standardized `tciGetModulePar()` function. During module initialization the runtime system requests the value of each module parameter. In addition to existing ways of defining the module parameter values (using t3ud_read_module_param() function, in the module parameter file or with the use of command line parameters) IBM Rational Systems Tester supports `tciGetModulePar()`. This function has priority over all above-mentioned mechanisms. The runtime system will query module parameter value in the following chain:

- `tciGetModulePar`
- `t3ud_read_module_param`
- command line parameters
- module parameter file

This chain will be executed independently for each parameter thus it is possible to leave the `tciGetModulePar()` unimplemented for some parameters and define them in the module parameter file.

The TTCN-3 TCI standard does not provide any way of obtaining the type for a module parameter, thus it is not possible to build a module parameter value if it does not have default value. For this purpose IBM Rational Systems Tester provides an additional non-standard function, `tciGetModuleParameterType()`, that may be used.

Asynchronous behavior in TCI

The TCI TM interface contains two functions, `tciStartControl()` and `tciStartTestCase()`, whose behavior is different from the similar RTS function t3rt_run_test_suite(). t3rt_run_test_suite() will only after finishing execution return the whole control part. In turn `tciStartControl()` will start the control part and return immediately without waiting for the termination of the control part. The same is true for `tciStartTestCase()`.

The synchronous behavior of `tciStartControl()` may be implemented as following:

```
tciStartControlSync() {
  tciStartControl();
  <wait for tciControlTerminated() to be called>
  return;
}
```

The main program generated by the compiler will handle this.

Using logging capabilities of the TCI

IBM Rational Systems Tester fully supports the TCI TL logging (test logging) interface. This interface consists of an exhaustive set of functions that cover the various aspects of a test execution. Each function is invoked by the runtime system when the corresponding event occurs during a test execution. All information concerning this event is passed through parameters of these functions. Besides information specific for a certain event, each function passes the id of a component that generated this event, the timestamp of an event and a location inside the TTCN-3 source file.

Using the TCI TL it is for example possible to log:

- send and receive of a message
- procedure calls, replies and exceptions, in procedure communication
- creation and termination of components
- changes to the values of variables
- evaluation of alternatives in an alt statement

Note

The t3tci_template.c file contains the implementation with an empty body for all of the TCI TL functions.

Limitations - TTCN-3 Tools

This section describes the known limitations for tools used in the process of creating an ETS. The release notes for IBM Rational Systems Tester contains additional information on tool limitations.

Editing Limitations

TTCN-3 Editing Limitations

- Coloring of nested comments is not supported. First encounterd '*/' sequence in nested comment block changes "comment" color to "normal text" color.

ASN.1 Editing Limitations

- A structured view for ASN.1 modules is not available.
- Named bookmarks are not available in ASN.1 files.

Runtime System Limitations

Test Case Parameters

Passing of '`inout`' and '`out`' test case parameters will not keep information in partially initialized values. For these parameters, a value will be passed only if it is fully initialized – otherwise it is considered not to be initialized. The workaround is to make sure that values passed as '`inout`' or '`out`' parameters to test cases are fully initialized, even if parts of them are going to be overwritten.

Type Checking Limitations

The type checking implements a structural compatibility check that is slightly stricter than what ETSI ES 201 873-1 V2.2.1 specifies. Types are compatible when they are of the same "kind" of type. For example, two records with the same (recursive) type composition of the fields are compatible while an array of three integers is not compatible with a record containing three integer fields.

NULL ASCII Character in Character Strings

The ASCII character NULL (‘\0’) is not supported inside character strings. Even though it can be inserted into a string, operations on such strings will not be performed correctly (length calculation, for instance).

Limitation in implementation of TCI interface.

Implementation of TCI TL logging in IBM Rational Systems Tester has following limitations:

- duration parameter in `tliTStop` event always contains zero.
- Four functions (`tliMDetected_c`, `tliPrGetCallDetected_c`, `tliPrGetReplyDetected_c`, `tliPrCatchDetected_c`) that correspond to the detection of an arrived inter-component message, procedure call, reply or exception will not provide the detected values.
- `tciGetValueEncoding()` and `tciGetValueEncodingVariant()` functions are not supported. `tciGetTypeEncoding()` and `tciGetTypeEncodingVariant()` may be used instead.
- `tliMatch` event is not supported.
- `tciGetTestCaseTSI()` is not supported.

Compiler Limitations

Memory allocation exception

During the compilation of a TTCN-3 file the compiler consumes virtual memory. The larger TTCN-3 file the more memory is required by the compiler to generate the C code. In the situation when the TTCN-3 file is very large there may occur a memory allocation error exception during code generation. If this happens there are two possible actions that can remedy the situation:

- Increase the size of the swap file, thus increasing the size of available virtual memory.
- Split the TTCN-3 file into several smaller files.

Note

On Linux and Solaris the compiler will not notify you about a memory allocation error if the size of virtual memory is set to <unlimited>. Instead the program will stop execution with a "segmentation fault" message. To get a meaningful error message you must ensure that the maximum size of virtual memory is set to a limited value. This parameter is changed using the command "ulimit -v".

Stack overflow

A very large TTCN-3 file may result in a stack overflow exception during static analysis.

In this case it may become necessary to increase the size of the available stack for a compiler process. On Windows it is done using the `editbin.exe` utility supplied with Visual Studio .NET, on Linux and Solaris it may be done using the "`ulimit -s`" command.

Note

On Linux and Solaris the compiler will stop execution with a "segmentation fault" message in the case of a stack overflow exception.

Makefile Limitations

- In makefiles, the path to ASN.1 libraries (ASN1_LIB) may not be longer than 250 characters. If longer, the path will be truncated and as a consequence, it is not possible to build an executable.

Other Limitations

Statement terminator symbol (semicolon) is optional everywhere except two places:

- Altstep local declarations (variables, constants, timers or templates)
- Counter declaration inside the header of 'for' loop

Build limitations

Build intelligence doesn't support library projects that contain only ASN.1 files.

TTCN-3 Compiling Limitations

TTCN-3 Module Specific Settings and Command Line Compilation

If you use the `-d` flag (for specifying an output directory) in the **TTCN-3 module specific settings** in the **Settings** dialog box, or if you use the Compiler from the command-line, the following is not supported:

- More than one non-existing directory in a path.

  However, if you specify **one** non-existing directory, it will be created. For example, this is allowed:

  ```
  -d /home/output/doesnotexist/
  ```
  This is not allowed:

  ```
  -d /home/output/doesnotexist/doesnotexist/
  ```
  Compilation will fail.
- Current directory for Windows drives. For example, if you generate to a directory with the option

  ```
  -d C:
  ```
  the output will be generated to the root instead of the current directory.

ASN.1 Compiling Limitations

Module Names

- If module names do not differ in the first 5 characters of the identifier, name clashes will occur.
- The target code includes a number of header files. Their names may clash with names of header file in you generated source code. Especially on Windows, the file name clashes may result in compilation errors, such as duplicate definitions and/or missing definitions. You avoid these errors by renaming the modules in your source code.
- ASN.1 and TTCN-3 modules must not have identical names

Modules and Files

- A module file’s base name must not be more than 30 characters long.

Module Definitions

- A SEQUENCE OF type having an element type referenced from another module, will result in errors during compilation and/or linking.
- Import definitions are mapped to TTCN-3 to the ‘import all’ construct.

Module Related Errors and Warnings

- `FATAL ERROR: Can not open file for reading`

  This error message mean that you have not specified a file that is required for compilation, when running the Compiler from the command-line.

  The missing file is a file containing an ASN.1 module that is referenced in other modules, which were included in compilation.

Identifiers

Identifiers in ASN.1 may only be 128 characters. If longer, only the first 128 characters will be used.

Open Types

Open types are mapped to the corresponding ‘`anytype`’ without restrictions, so that the union type generated contains all the types visible in a module.

ASN.1 Module Specific Settings and Command Line Compilation

If you use the `-d` flag (for specifying an output directory) in the **ASN.1 module specific settings** in the **Settings** dialog box, or if you use the Compiler from the command-line, the following is not supported:

- More than one non-existing directory in a path.

  However, if you specify **one** non-existing directory, it will be created. For example, this is allowed (assuming that `/doesnotexist` are non-existing directories):

  `-d /home/output/doesnotexist/`
  This is not allowed:

  `-d /home/output/doesnotexist/doesnotexist/`
  Compilation will fail.
- Current directory for Windows drives. For example, if you generate to a directory with the option

  `-d C:`
  the output will be generated to the root instead of the current directory.

ASN.1 Constructs Supported in TTCN-3

Only the following ASN.1 constructs are supported in TTCN-3:

- CHOICE
- BOOLEAN
- CLASS
- ENUMERATED
- EXTERNAL
- IA5String
- INTEGER
- NULL
- NumericString
- OBJECT IDENTIFIER
- OCTET STRING
- Open type, with the limitation of no open type values
- Parameterization
- PrintableString
- SEQUENCE
- SEQUENCE OF
- SET
- SET OF
- VisibleString
- UTCTime
- Basic value definitions only

ASN.1 Constructs Not Supported

The following ASN.1 constructs (from X.680-683) are **not** supported:

- ABSTRACT-SYNTAX
- ASN1-CHARACTER-MODULE
- BMPString
- CHARACTER STRING
- EMBEDDED PDV

- Extension groups
  notation “[[““]]”
- GeneralString
- GraphicString
- INSTANCE OF
- ISO646String
- ObjectDescriptor
- Selection type
  notation “<“
- T61String
- TeletexString
- TYPE-IDENTIFIER
- UniversalString
- “ValueSetTypeAssignment”
  Notation “<type> <type::= <valueset>”
- VideotexString
- Absolute reference
  Notation “@”<module>.<element>
- C style comment notation, /* comment */

ASN.1 Constructs Partially Supported

The following ASN.1 constructs are supported, but with limitations:

ASN.1 Construct	**Comment**
ABSENT	Will pass analysis, but no code will be generated.
ALL	Will pass analysis, but no code will be generated.
APPLICATION TAGS	No PER support. See “TAGS” on page 109.

ASN.1 Construct	Comment
BIT STRING	PER Limitations: Maximum fixed constraint value is 64K-1. Bit strings of types defining named bits should be defined using the bits. Named bits are not imported into TTCN3. Define all bit string values by using named bits in ASN.1.
CHOICE	Value notation is not supported. PER limitations: Max 255 alternatives in extension root. Max 63 alternatives in extension addition.
CLASS	Will pass analysis, but no code will be generated. Information on usage is in “Open Types” on page 103. All types used in CLASS definitions must be defined types.
COMPONENTS OF	Value notation is not supported. Will pass analysis but no code will be generated. See “SEQUENCE” on page 108.
CONSTRAINED BY	Will pass analysis, but no code is generated for CONSTRAINED BY constraints and keywords used in those definitions so the value checking has to be implemented in application code.
ENUMERATED	Enumeration item values must fit into 32 bit integers. PER limitations: Enumerated type must be a defined type when it is a component of SEQUENCE, SEQUENCE OF, or SET OF type.
EXCEPT	Will pass analysis, but no code will be generated.
EXPLICIT TAGS	No PER support. See “TAGS” on page 109.
Extensibility (“...”)	Only one extension marker supported in structures. No extensibility of INTEGER, ENUMERATED or string types’ constraints for ASN.1 encoders. Will pass analysis, but no code will be generated.
FROM	Only list syntax is supported. Will pass analysis, but no code will be generated.

ASN.1 Construct	Comment
IA5String	PER general restrictions: • Unconstrained • Max 16 bits/character
IF PRESENT	No code is generated for simple WITH COMPONENT, WITH COMPONENTS, and CONSTRAINED BY constraints and keywords used in those definitions so the value checking has to be implemented in application code.
IMPLICIT TAGS	No PER support. See "TAGS" on page 109.
INTEGER	Supported value range is 64 bit signed integers. Note: Compiler specific integer type is usable for Visual C++ and GCC compilers, generated for INTEGER subtypes with corresponding constraint values Note: GCC option `-O` is not usable with large integers and ASN.1 encoders.
INTERSECTION	Will pass analysis, but no code will be generated.
MAX	Will pass analysis, but no code will be generated.
MIN	Will pass analysis, but no code will be generated.
MINUS-INFINITY	REAL value range constraint identifiers MINUS-INFINITY and PLUS-INFINITY are not supported.
NULL	Value notation is not supported in TTCN-3. Import for example: `asn1-NULL NULL ::= NULL`
OBJECT IDENTIFIER	Maximum number of elements is 32.
OPTIONAL	Open type field could not be optional in inline subtype definition. Use explicit subtype definition.
OCTET STRING	Maximum fixed constraint value is 64K-1.

ASN.1 Construct	Comment
Parameterization	Parameterization of types with types, object sets and values is supported. Parameterization of CLASS with CLASS or object sets is not supported. Parameterization with inline SEQUENCE values is not supported. Use explicit SEQUENCE value declaration.
PLUS-INFINITY	REAL value range constraint identifiers MINUS-INFINITY and PLUS-INFINITY are not supported.
PRIVATE TAGS	No PER support. See “TAGS” on page 109.
REAL	Will pass analysis, but no code will be generated. REAL values are not supported in CLASS definitions. No PER support.
SEQUENCE	PER limitations: Max 32 optional elements in extension root. Max 32 elements in extension addition. Value notation is not supported. Extensible constraints are not supported. Only one extension marker supported in structures. See “SEQUENCE OF” on page 108, “COMPONENTS OF” on page 106 and “TAGS” on page 109.
SEQUENCE OF	An imported element type of a SEQUENCE does not work. Value notation is not supported. PER limitations: Component type must be a defined type if SEQUENCE OF is defined without size constraints or with large upper bound constraint (more than 64K).
SIZE	Will pass analysis, but no code will be generated. In PER encoding, only range constraints are supported.

ASN.1 Construct	Comment
TAGS	AUTOMATIC TAGS default tagging mode is required for PER support. In general, tagged types are not supported in PER but they work if the tag does not have effect to resulting coding according of the PER encoding rules. If AUTOMATIC TAGS are not used, the encodings will be correct only for structures where the target happens to produce the same order of elements as with AUTOMATIC TAGS. See “APPLICATION TAGS” on page 105, “EXPLICIT TAGS” on page 106, “IMPLICIT TAGS” on page 107, and “PRIVATE TAGS” on page 108.
UNION	Will pass analysis, but no code will be generated.
UNIVERSAL	See “TAGS” on page 109.
WITH COMPONENT	No code is generated for simple WITH COMPONENT constraints and keywords used in those definitions so the value checking has to be implemented in application code.
WITH COMPONENTS	No code is generated for simple WITH COMPONENTS constraints and keywords used in those definitions so the value checking has to be implemented in application code.
“|”	Complex constraints are not supported in type and `codec` generation and a warning will be generated.
“!”	Exception specification is treated as comment, that is, the ETS must check and handle the exceptions.
“^”	Will pass analysis, but no code will be generated. No PER support.
“\”	Will pass analysis, but no code will be generated. No PER support.

5

Building

Before you can execute your test suite, the generated ANSI-C code must be compiled by using either the previously generated or user provided makefile.

Compiling generated code

Using GCC on Windows to compile C files

To use IBM Rational Systems Tester with GCC on Windows, the following should be done:

1. Locate `make_win_gcc.mcfg` file in the `include` subdirectory inside Rational Systems Tester installation and copy it to your project directory.
2. On **Build** tab in **Settings** dialog set target platform to “CygWin“.
3. When manual makefile configuration is used set mcfg file to `make_win_gcc.mcfg`. When automated makefile configuration is used open **Makefile Options** dialog and load `make_win_gcc.mcfg` file.
4. Change the **Build Settings** in the TTCN-3 project in the following way:
   – Remove the preamble `vsvars32.bat`
   – Change the make command from `nmake` to `make`

Rational Systems Tester GCC libraries

A working environment for GCC assumes a IBM Rational Systems Tester installation, and for Rational Systems Tester also a distribution of the specific win32-GCC files that are required for the operation of GCC from within the Rational Systems Tester environment.

This distribution contains the following libraries and configuration files, compiled with GCC:

```
libt3rts.a
librts.a
libcvrts.a
libasn2t3rts.a
libt3tri.a
libt3tci.a
libt3tcite.a
libt3dbg.a
make_win_gcc.mcfg
```

Restrictions using GCC for Rational Systems Tester

Error handling

Error handling where signals are caught using `sigaction` is not supported. As an example of this: should the application perform a division by zero, it will result in the application terminating unexpectedly, instead of catching the exception signal and printing an appropriate error message.

Using a Remote Host Compiler

If you use the Windows version of Rational Systems Tester, you are still able to use a Rational Systems Tester Compiler located on a UNIX server. Your project has to be located in your UNIX home directory, and the UNIX Compiler has to be reachable from that server. Another prerequisite is of course that the server has a valid license for the Rational Systems Tester Compiler.

In your local installation of Rational Systems Tester, you can find options for using a remote Compiler in the Rational Systems Tester/TTCN to C tab in the Options dialog box. When you have set the options to invoke a remote Compiler, that Compiler will be used the next time you analyze and generate code for a project that is located in your home directory, according to options in the Settings dialog box.

About the Rational Systems Tester/TTCN to C options

- Enable remote invocation

  Enables you to use a Compiler located on a UNIX server. If this is not selected, all other options are unavailable.

- Remote host

  Type the name of the server where the Compiler and your project are located.

Authorization options:

- User name

  Type a user name that is valid for the server.

- Password

  Type the password associated with the user name.

Path mapping options:

- Local path

  Enter the path to your UNIX home directory, as seen from your Windows computer, that is, the drive to where your UNIX home directory is mapped, for example z:.

  It is not necessary to map the network drive before you enter the local path, but it must be mapped before the remote invocation will work.

- Remote path

  Type the full path to your UNIX home directory, as seen from the UNIX server that you specified in the Remote host box. For example:

  `/home/your_name`.

- Remote tools path

  Type the full path to the remote Compiler, t3cg, as seen from the UNIX server that you specified in the Remote host box. For example /opt/Telelogic/Tester/bin.

SSH options:

- Reject connections to unknown hosts

  Rejects connections to the server if it is not listed in the known hosts file.

- Automatically memorize unknown hosts

  Adds the server to the known hosts file, if not listed already.

- Extra SSH parameters

  Optionally, type additional SSH parameters.

Bypassing the TRI external function mechanism

It is possible to bypass the RTS mechanism for calling external functions, for example using `triExternalCall` in the TRI case.

Example 17: ----------

1. In the makefile, define the symbol `EXT_FUNC_BYPASS_TRI_m_f`, for a function f in a module m.
2. At link time, provide an implementation of a function with the following prototype:

   ```
   extern t3rt_value_t m_f(long t3cg_argc, t3rt_value_t
   t3cg_argv[], t3rt_context_t t3cg_context).
   ```

   This function will then be called in the generated code wherever the function f is invoked.

Build Settings

The output from the code generation is:

- ANSI-C files
- Header files
- makefile (optionally)

All this must now be compiled and mapped to an integration (and possibly a TRI implementation), in order for the ETS to be able to be executed in the actual test environment.

Build Settings for Generating Code

To be able to generate appropriate code and makefile, you should change the following build settings before code generation:

- Specify which **TRI settings** to use (relevant only for manual makefile configuration). The alternatives are:
  - **Rational Systems Tester TRI**

    This is a complete TRI compatible library provided by IBM Rational. It can be found in the directory `/integrations/tri/<platform>` in the installation directory. Rational Systems Tester TRI is the default value.
  - **Example**

    This is an example non-TRI based example integration provided by IBM Rational.

    The complete example integration can be found in the directory /integrations/example in the installation directory.
  - **None**

    Select this if you have made an entire integration and wish to control the configuration of the build process.

    If this option is selected, only the generated C files are included in the makefile, according to what you have specified in the makefile generation configuration file. You can select to either edit this configuration file prior to code generation, or to edit the generated makefile.
- **Specify Root module**

  This is the root TTCN-3 module that defines the execution order. It is necessary to specify this, otherwise no makefile will be generated.

- **Select Generate makefile to generate a makefile**

  If selected, a makefile will be generated, based on the selected TRI setting and the makefile configuration file in your project.

  If you generate a makefile, you should also specify whether it should overwrite an existing makefile (with the same name and in the same directory), after a copy of it is saved; or if the new makefile should be saved with the extension `.new`, thus not overwriting an existing makefile.

You find all these settings in the Build tab in the Settings dialog box.

Settings for Building

To be able to build the ETS correctly, you have to specify the following:

- A **Make command preamble** (that is, the file that specifies the configuration of your build environment), depending on what operating system and third-party compiler vendor you use.
- A **Make command**, depending on what operating system and third-party compiler vendor you use.
- A **Makefile** to use when building. If you have generated a makefile, the name of the makefile is specified in the makefile configuration file

Specifying result file

It's necessary to specify the result file on the **Build** tab. Result file denotes the target file that is created during project build. It's used by Rational Systems Tester to track changes in the project and suggest recompiling project when source files or project settings changed.

Result file is automatically filled during the compilation if root module is specified.

Note

Result file may differ from the executable file.

Set Build Options

Note

Code generation must be performed before build.

1. On the **Project** menu, click **Settings**, and then click the **Build** tab.
2. Specify Make command preamble.

3. Specify a Make command.
4. Enter the location of a previously generated makefile.
5. Click OK.

The other settings in the Build tab are relevant for code generation.

See also

"Set Compiling Options" on page 89.

Build an ETS

To build an executable test suite:

- Click the Build button on the toolbar.

  or

  on the Project menu, click Build

  or

  click F7.

To cancel build:

- Click the Stop action button in the toolbar

  or

  on the Project menu, click Stop action.

Build intelligence

Rational Systems Tester tracks changes in the project to avoid inconsistent builds. The tracking includes:

- Changes in all source files attached to the project
- Changes in the dependent projects
- Addition of new files to the project
- Changes in the project settings

When you invoke build, execute or debug action Rational Systems Tester checks whether this action is valid for the current project state. If project is out of date then dialog window is opened asking to recompile or rebuild project. You may accept or reject Rational Systems Tester suggestion. The possible choices are: Yes, Yes to all, No, No to all and Cancel.

- **Yes** means that suggested action and only it should be performed. Further actions (if any) should be asked for confirmation again.
- **Yes to all** means that this actions and all necessary further action should be performed without asking confirmation.
- **No** means that this action should be skipped.
- **No to all** means that this action and all further actions should be skipped without asking confirmation.
- **Cancel** means that the action should be aborted.

Each option in project settings fall into one of three classes. Changing it requires project recompilation, project rebuild or nothing. For example changing root module forces project recompilation while changing executable command line parameters does nothing.

Note

Build intelligence takes into account compile and build errors and project sessions. It may not be reset by closing project and reopening it again.

Project configurations

You may specify arbitrary number of configurations for the every project in a workspace. All project settings including “Exclude from build“ option are stored separately for every configuration.

Note

Dependencies between projects are common for all configurations

Using configurations you may easily maintain **debug** and **release** versions of your test suite or versions that are compiled using difference target compilers (e.g. Microsoft VC++ and GCC).

To create new configuration open **Project** menu and choose **Configurations**. Press **Add** button, enter the name for the new configuration and configuration to copy settings from. Press **Ok** to finish operation.

Important!

It's strongly recommended that all projects in the workspace have one and the same set of configurations

Multiproject workspaces

Usually workspace contains only one project that is built into executable. Sometimes there is a need to divide project into several smaller ones moving common parts into libraries. Rational Systems Tester assists you greatly in maintaining such complex projects.

Specifying dependencies between projects

To create multiproject workspace do the following:

- Fill workspace with projects by creating new projects within current workspace or adding existing projects to the workspace.
- Define dependencies between projects.

Dependency tree of projects inside a workspace is defined by specifying all projects which current project depends on. Right click on a project and choose **Project Dependencies** in the popup window. Check all projects that should be built prior to this project. Rational Systems Tester prevents you from defining cyclic dependencies, which are not allowed. Every time you change dependencies you may switch to **Build Order** tab and observe the evaluated ordering of projects that will be used in all build operations.

After dependencies are defined Rational Systems Tester executes all build operations according in the automatically evaluated build order.

Batch build operations

There are four build operations in IBM Rational Systems Tester environment:

- Analyze
- Compile
- Build
- Clean

Each of these operations may be applied to either only the selected project or active project including all dependent projects or all projects in the workspace.

By default active project is built including all dependent projects. This type of build operation is invoked when you invoke build operations from **Project** menu, from toolbar or when you right click on a project and choose operation from popup menu. You may also open **Build** menu and invoke analyze, compile, build and clean operation from there.

Note

*Active project and current project configuration is specified using the **Project** toolbar or **Configurations** item in **Project** menu.*

You may want to perform operation only for the selected project without dependent ones. Right click on a project or select a project and open **Build** menu. Open **Project Only** submenu and choose required operation.

Finally you may decide to perform operation for the whole workspace. In this case operation will be performed for all projects in their dependency order. Open **Build** menu and choose **Analyze All**, **Compile All**, **Build All** or **Clean All**.

When the invoked build operation requires executing operations in the dependent projects Rational Systems Tester tries to use one and the same configuration for all projects. It's your task to have certain configuration defined for all depending projects. If no corresponding configuration found in the dependent project then Rational Systems Tester warns about it asking whether you would like to continue process using the default configuration.

Example 18

If you decided to build project **MyExe** when the current configuration is set to **MSVC** and this project depends on **MyLib** project then Rational Systems Tester tries to locate **MSVC** configuration in **MyLib** project and use it. Lack of the **MSVC** configuration in **MyLib** project may result in inconsistent build.

6

Execute Tests

When executing a test suite, you get reports, logs, of which test case in the test suite is executed. You also get the runtime reports of execution progress, as well as the verdict.

During execution, the log information can be saved both as an MSC trace, and as a text file that allows you to navigate from each event to the source in the abstract test suite. You may also specify your own log formats.

The log information, that is, runtime reports and the log file, are displayed in the Execution tab in the Output window. MSC files may be displayed in a separate viewer.

It is possible to reproduce test cases as well as to (re)configure connections to the underlying stack or SUT.

Command-Line Execution Overview

Command-line execution is fundamentally the way to execute the built ETS. The execution from the Rational Systems Tester user interface is a wrapper around the command-line execution, making it usable from the user interface with dialogs, log output, and navigability.

The RTS part of the built ETS can be given command-line arguments for controlling and setting up the environment for the ETS to execute in.

Command-Line Syntax

```
<ets> [user switches] -t3rt ["<switches>" [user
switches] ]
```

or

```
<ets> [user switches] -t3rttmpl <filename>
```

Hint

Just using -t3rt with no switches will display a help text on `stdout` *stream.*

- `[user switches]`

  Switches that are handled by the main function in general, the RTS does not regard them. The switches to the RTS is given as a space separated list of switches within the quoted string.

- `-t3rttmpl <filename>`

  This will generate a configuration file template, in a file with the given name to be used with the -file switch.

Example 19: ----------

```
etsbin -t3rt "-file myconfigfile +lrtconf -to 10.0"
```

Switches are parsed from left to right, and depth-first into any configuration files, that is, any switches given after a file directive will override the switches in the file.

Note

These switches can also be used when executing from the Rational Systems Tester user interface: On the Project menu, click Settings and then click the Execution tab. Enter the switches in the Additional execution switches text box.

Command-Line Switches

Switch	Description
`-confint <k> <n>`	General integer value. Sets the key <k> to the integer value <n>. Example: `-conffloat my.int 42`
`-confbool <k> true|false`	General boolean value. Sets the key <k> to the boolean value true or false. Example: `-conffloat my.flag false`
`-conffloat <k> <n>`	General floating point value. Sets the key <k> to the floating point value <n>[.<m>]. Example: `-conffloat my.float 41.9`
`-confcstr <k> [']<s>[']`	General character string value. Sets the key <k> to the character string value <s>. Example: `-confcstr my.str qwerty`
`-confbstr <k> <bstr>`	General binary string value. Sets the key <k> to the binary string value <bstr>. The binary string can be given on the form: `[']<str>[']\[h|H|o|O|b|B\].` Example: `-confbstr my.hex 'DEADBEEF'H`
`-file <filename>`	Use configuration file <filename> as input. See "Configuration Files" on page 132 for more details.
`-par <par_name> <value>`	Module parameter initialization. The named module parameter will be initialized with the given value definition. See "Module Parameter Syntax" on page 133 for the value definition format. Initializing a module parameter value with this switch will have the highest precedence (over both initialization file and default value). Example: `-par Module1.par1 12` Example: `-par par2 {{3,false},{2,true}}`

Switch	Description
`-parfile <filename>`	Module parameter initialization file. When module parameters are initialized and no explicit command-line definition is given (-par switch) a value definition will be searched for in the given initialization file(s). If this switch is used multiple times on the command-line (with different files) the last value definition found in any of the files will have precedence. The files will be processed in the order given on the command-line. See "Module Parameter Syntax" on page 133 for the value definition format. If a multi-process integration is used for concurrent component execution, the same filename as given on the command-line will be used in all processes, that is to say that the initialization file must be available to all processes in terms of file location. Example: `-parfile ModPars_01`
`-v/--verbosity <level>`	Verbosity level of the built-in textual event log. The value ranges from 0 to 4 with the following meaning: 0 - off (no log event are displayed) 1 - minimal (for example `testcase_started`, `test case_error`, etc.) 2 - normal (most TTCN-3 statement operations) 3 - extended (for example `message_detected`, `port_mapped`, etc.) See "t3rt.logging.builtin.verbosity" on page 129 for more details.
`-lrtconf/+lrtconf`	Disable the displaying of the runtime configuration contents. Disabled by default. See"t3rt.logging.rtconf_dump.enabled" on page 129 for more details.

Switch	Description
`-to <time-out value>`	Maximum time-out time for the execute statement. See “t3rt.behavior.default.testcase_timeout” on page 128 for more details.
`+tmpp/++temporary_memory_poison_pill` `-tmpp/--temporary_memory_poison_pill`	Enable/disable “poison pilling” of released temporary memory. See “t3rt.temporary_memory.poison_pill.enabled” on page 131 for more details.
`-tmru/--temporary_memory_release_unused`	Enable/disable immediate deallocation of released temporary memory. See “t3rt.temporary_memory.release_unused.enabled” on page 131 for more details.
`-tm <nbytes>/--block_size <nbytes>`	Defining the size (in bytes) of the blocks that are being allocated when the temporary memory area grows. See“t3rt.temporary_memory.block_size” on page 130 for more details.

User Interface Execution

Execution Settings

To start an execution of your test suite you click **Execution** on the **Project** menu.

The execution settings gives you a possibility to make choices for the execution. You find the settings in the **Execution** tab in the **Settings** dialog box.

You can select one or all of these options in a single execution:

- **Module parameters**

  Allows you to specify a module parameters initialization file. For more information, see “Module Parameter Syntax” on page 133.

- **Behavior**
  - **Default test case timeout**

    Allows you to specify a default time-out value in seconds that will be used if the execute statement is called with no time-out value.
  - **Use of built-in codec if non is registered for a type**

    Allows to use built-in codec for all types that do not have explicitly registered encoders and decoders. Note that use of this option is safe only for internal communication between test components.
  - **Continue template matching on fail**

    Allows to proceed with matching fields of the structured value after mismatch is detected. Using this option slows down performance but shows all differences between the template and matching value.
  - **Assume timers in TRI are implemented as "active"**

    Tells runtime system library that TRI timer implementation actively reports timeouts with triTimeout function. Otherwise RTS will try to evaluate timout basing on timer elapsed time.
  - **Pass template formal parameters by value**

    Tells RTS to translate static templates into template variables and pass them by value to called function.
- **Optional configuration file**

  Allows you to specify a configuration file, for re-use purposes. (That is, if you have an individual configuration file that you wish to use and re-use for several test executions.)
- **Additional execution switches**

  Allows you to set additional switches, that is exceptions or additions to the settings in the configuration file.

  All switches are listed in "Command-Line Switches" on page 123.
- **Executable test binary**

  Specify location of the executable test binary, the ETS, created during build.

Invoking an ETS on a Remote Host

An executable test suite may be located on a UNIX server, and you will still be able to invoke it from your computer. There is no specific license required for the UNIX server, but the ETS must be executable on that platform.

You find options for remote invocation of an ETS in the Options dialog box, Rational Systems Tester/TTCN to C tab. For a description of options, see "Using a Remote Host Compiler" on page 112 in Chapter 5, *Building*. The options are described from a Compiler perspective but are applicable to ETS invocation too.

Set Execution Options

1. On the **Project** menu, click **Settings**, then click the **Execution** tab.
2. Select:
   - Enable log event generation to allow logging.
   - Display Configuration settings, when applicable.
   - Display registered log mechanisms, when applicable.
   - Log also to file and specify a filename and destination, when applicable
3. Specify Module parameters.
4. Specify Default test case time-out in seconds, when applicable.
5. Select Optional configuration file, when applicable.
6. Set Additional execution settings, when applicable. All switches are listed in "Command-Line Switches" on page 123.
7. Specify Executable test binary.
8. Click **OK**.

Execute a Test Suite

To execute a test suite:

- On the **Project** menu, click **Execute**.
- Press **F5**
- On the toolbar, click the **Execute** button.

To cancel an execution:

- Click the **Stop action** button on the toolbar
- On the **Project** menu, click **Stop action**.

Predefined Configuration Keys

Keys starting with t3rt are reserved for the RTS. There are also some predefined keys that the RTS uses that can be set using the general `-conf...` switches.

t3rt.behavior.default.testcase_timeout

Setting this value will override the “wait forever” behavior of calling the execute statement with no timeout time. This will not apply to the execute statement that a timeout value has been given.

The value is a float-point value in seconds.

Useful to force a time-out even if a test case is hanging. This will make a control execute from start to end and generate a test case error for each timed out test cases.

No default timeout value is set by default.

t3rt.control.ack_timeout

The maximum time (in seconds) before timing out on pending (internal) acknowledgements.

The value is a float-point value in seconds.

t3rt.timers.assuming_all_active

Setting this boolean will make the RTS assume that all timers are active.

The default value is false, that is, the timers can be implemented using both a passive and/or an active scheme.

This means that if the RTS is not told explicitly (using “t3rt_timer_timed_out” on page 340 in Chapter 10, *Runtime System APIs* (non-TRI) or “triTimeout” on page 554 in Chapter 10, *Runtime System APIs* (TRI)) that a timer has timed out, it will never time out.

This should only be used if the timers' integration is using active timers only. See the Technical Integration Documentation for different timer implementations.

t3rt.logging.builtin.verbosity

This boolean value controls the verbosity level of the built-in textual log.

Level 0 turns off the logging.

Level 1 is a minimal set of event just reporting the progress of test cases, their verdicts and any test case errors. This is intended for a standard test run where the test result itself is interesting.

Level 2 (default) is intended for test suite writers who needs to trace the execution more closely to find out what goes wrong in the executed test. Most significant TTCN-3 operations are logged here.

Level 3 - This is intended for the integration/codecs system writers that needs to know all the details available. Here encoding/decoding failures and external stimuli detection (that is, when information arrives in the RTS) are reported.

Level 4 - All events. Information and debug messages included.

See "Execution Logs" on page 143 in Chapter 7, *Execution Logs* for more information.

t3rt.logging.rtconf_dump.enabled

This will display the actual configuration settings in the runtime configuration mechanism.

This is useful if you make a lot of configuration through configuration files or long switch lines, and want to verify the final configuration settings.

Disabled by default. See the Technical Integration Documentation for an explanation.

t3rt.matching.continue_on_fail

A template allows you to specify matching information for received data (only allowed for receive constraints). Normally the matching process stops when an error has been detected.

This key allows you to complete the matching process even though errors has been detected.

Disabled by default.

t3rt.values.limits.epsilon_double

This value is used to set up “epsilon”, used in comparisons of floating-point values.

Compare with “t3rt_type_is_equal” on page 251 in Chapter 10, *Runtime System APIs*.

The default value is 2*DBL_EPSILON (from 'limits.h').

Note

Setting this value to 0.0 will be equal to direct comparison of floats. Generally, this is not recommended.

t3rt.codecs.builtin_as_default.enabled

Setting this boolean value to true will make the RTS use the built-in codecs when no registered codecs has been registered for a type.

The transfer syntax for this is internal and never will be published. Enabling this should only be made in those cases where messages are never sent to the SUT (or TRI SA implementation).

It is disabled by default.

t3rt.temporary_memory.block_size

Set size (in bytes) of the blocks that are being allocated when the temporary memory area grows.

The default value of this is 100.000 bytes.

Setting this to a large number will have the effect that fewer calls to the system is made to allocate heap memory, which is a slow operation. The memory area will grow in larger increments which could possibly make the ETS consume more memory than necessary.

Setting this to a small number will cause more memory allocating calls to the system and the temporary memory area will consist of (possibly many) smaller blocks.

t3rt.temporary_memory.poison_pill.enabled

Enabling this boolean value will result in memory being overwritten with a known bit pattern when the memory is (logically) “freed”. This is only applicable to temporary memory. See the Technical Integration Documentation for information on temporary memory.

This is disabled by default and enabling it is only relevant in debug situations in an environment where memory can be displayed. Enabling poison pilling will decrease performance slightly.

t3rt.temporary_memory.release_unused.enabled

Enabling this boolean value will result in immediate freeing of unused temporary memory when leaving temporary memory scope block, i.e. when calling t3rt_memory_temp_end function. By default deallocation of temporary memory blocks is delayed until component termination. When execution leaves temporary memory scope all blocks allocated in that scope are marked as unused and reused later upon runtime system request. Such strategy allows to save malloc calls, but may result in unnecessarily big memory consumption of test suite executable.

Enable this boolean value to free temporary memory blocks as soon as they become unused. This is only applicable to temporary memory. See the Technical Integration Documentation for information on temporary memory.

Enabling this flag will decrease performance slightly.

t3rt.values.value2string.print_kinds.enable

Enabling output of value kinds in the value-to-string operations.

t3rt.values.value2string.print_types.enable

Enabling output of value types in the value-to-string operations.

t3rt.values.value2string.print_field_names.enable

Enabling output of field names in the value-to-string operations.

t3rt.logging.builtin.print_field_names

Enabling output of field names in the built-in log.

t3rt.logging.builtin.pretty_print

Pretty print values in the built-in log. When printing structured values each element will be printed on separate line maintaining indentation.

t3rt.logging.builtin.limit_size

Truncate structured and values to the specified number of elements. If length of a value exceeds specified limit then value is truncated and “<...>“ is appended to its end.

Configuration Files

Collecting a set of “standard” configuration settings in one or more configuration files can be useful. Especially when they can be overridden using subsequent command-line switches.

```
<key> [<type>] = <value>
```

Example: `mysetting.integration.version integer = 201`

The type for a predefined key is implicitly known. Forcing another type for predefined keys will cause undefined behavior.

The types that can be specified as key types are:

```
integer
float
boolean
charstring
bitstring
hexstring
octetstring
binary
```

The string values can be quoted but that is only required when the string value contains spaces.

The value for a binary typed key is:

```
[<quote>]<data>[<quote>][<format specifier>

<quote> ::= '|" (enclosing quotes must be the same)

<data>  ::= A sequence of characters 0-1 or A-F. The
sequence may have blank separators.

<format specifier> ::= H|h|B|b|O|o  (hex, bit, octet)
```

If no format specifier i given, hex format is assumed. If a format specifier is provided, the data has to be quoted.

See also

"-file <filename>" on page 123

Module Parameter Syntax

Value definition on command-line

As stated in the documentation for the -par switch, a module parameter is specified as:

```
-par <par_name> <value>
```

Value definition in initialization file

When specified in a file, the syntax for each parameter is:

```
<par_name> := <value>;
```

Lines (in a file) starting with '#' are considered to be comment lines and will be ignored.

In both cases, the name of the parameter should be fully qualified with module name (for example MyMod.par1). If an unqualified parameter name (for example par1) is used, it is assumed to be a parameter of the root module. If this is not the case, the definition will not be found during initialization.

The value syntax has intentionally been chosen as close to TTCN-3 as possible. Basically, constant literal expressions that require no evaluation is supported.

List of supported expressions:

- Literals of type:

```
integer
char
float
boolean
objid
verdicttype
bitstring
hexstring
octetstring
```

```
charstring
record
recordof
set
setof
enumerated
union
```

- References to test suite constants.
- Field references to constants of record or set types, (that is, the “dot” notation).

  Example: `M.par1 := M.rec_const1.field1;`
- Element indexing in constants of array type.

  The numeric expression given as index have to be a positive integer literal or a reference to a positive integer constant.

  Example: `M.par2 := M.arr_const1[4];`

  Example: `M.par2 := M.arr_const1[M.int_const1];`
- Field identifiers for record or set value definitions.

  Example: `{ field1 := 42 }`
- The omit symbol for optional record or set fields.
- Implicitly omitted fields are only supported if field identifiers are used, that is, not if the value list syntax is used.

Limitations

The following is a list of constructs that cannot be used in module parameter definitions:

- Any of the TTCN-3 Operators (§15.0 in ETSI ES 201 873-1 V2.2.1).
- References to other module parameters.
- References to variables.
- Expressions of type:

```
universal char
universal charstring
anytype
address
port
component
default
```

- Union values can only be given using the `r-value` notation.

  Example: `M.union_par := { int_field := 13 };`

  Example: `M.union_par.int_field := 13; // Wrong!`
- The “not used” symbol ‘-’.

Note

Any constructs that are not covered by these two lists are considered to be unsupported.

Table editor for module parameters

IBM Rational Systems Tester provides grid-like editor for module parameters files. Double click on module parameters file in the workspace window, Rational Systems Tester opens window containing grid with four columns:

- Module Name
- Parameter Name
- Parameter Value
- Comment (optional)

Note

Table editor is loaded only for files with “modparams” extension. If Rational Systems Tester fails to parse file structure it opens file in text editor.

Add module parameters

There are two ways of adding module parameters:

- Drag-and-drop module parameter from the “Structured view” window
- Choose parameter from the drop down list box in the table

Using drag-and-drop to add parameters

To add module parameters to the table using drag-and-drop:

- Open Structured View window and locate module parameter node
- Drag this parameter to the table and drop it on the last (free) line in the table. You need to drop it over second column (Parameter Name). Rational Systems Tester automatically fills Module Name column

Note

Dropping parameter over filled (not the last) line in the table overwrites existing data

Adding parameter from the drop down list

To choose the parameter from the list of all defined module parameters:

- Locate last (empty) line in the table
- Select second (Parameter Name) column
- Expand the list, scroll down to desired parameter and choose it. Rational Systems Tester automatically fills Module name column

Hint

You may want to start with choosing the module. Then Rational Systems Tester filters out all parameters defined in other modules.

Deleting module parameters

To delete one or several parameters from the table select desired parameters with the mouse by clicking on the line header containing ordinal line number and press `del` key.

Hint

You may use Ctrl and Shift keys to select blocks of parameters

Sorting and reordering parameters

You may reorder parameters in the table by selecting a line and drag-and-dropping it to another location in the table (upper or lower).

You may sort the entries of the table in the ascending or descending order by clicking on the column header.

You may change the width or height of a line by pointing the mouse over column or line header boundary and expanding it to the desired size.

Module parameter validation

Rational Systems Tester doesn't perform any kind of validating data entered in the table editor. Checking whether specified parameter is defined in the test suite and whether provided value conforms to the parameter type is performed at runtime during test suite initialization.

Test management in Rational Systems Tester GUI

IBM Rational Systems Tester provides the ability to manage test execution directly from GUI without hard coding the control part and rebuilding the test executable each time there is a need to execute a certain test case. A test case that does not have parameters may be started with only a mouse click. Besides executing single test cases it is also possible to define test plans - a sort of batch file that defines a sequence of test cases to be executed. Unlike changes to control parts, changes to the test plan does not require a rebuilding of the test executable. Each IBM Rational Systems Tester project may contain an arbitrary number of test plans.

In order to manage tests, execution from GUI project should be compiled with “Test management” project setting set to “Dynamic through Rational Systems Tester GUI”. This option is located on “Build” tab in the Project Settings dialog.

Note

This option should be set at start of test executable also.

When a test executable with enabled test management is started IBM Rational Systems Tester establishes communication with it and requests a list of modules and defined test cases. This information is displayed in the “Execution” tab in the “Output” window.

Note

“Execution” tab is normally hidden and activated only at start of test executable.

Modules and test cases are displayed in a tree-like view. The root of the tree is a name of a test executable. The next level is occupied by the modules defined in the test suite. Leafs of the tree represent the test cases defined in the test suite.

How to enable GUI test management for existing project

Several changes have to be done in the project configuration (`.mcfg`) file in order to use GUI test management for existing project:

- Add the following three libraries to the list of libraries linked to the test executable.
  - Windows: libt3tci.lib libt3tcite.lib rpcrt4.lib
  - Linux: libt3tci.a libt3tcite.a librpcsvc.a
  - Solaris: libt3tci.a libt3tcite.a librpcsvc.a libnsl.a

  Under Linux and Solaris due to peculiarities of a standard Unix linker (`ld`) it is necessary to enclose linked libraries with special switches. If gcc compiler is used to build test executable then above mentioned three libraries should be linked with the command:

  ```
  -Wl,--start-group -lt3tci -lt3tcite -Wl,--end-group -lrpcsvc
  ```

  rpcrt4.lib and librpcsvc.a are standard system libraries.
- Add t3tci_template.c file to the list of source files. This file is by default located in the directory:

  ```
  [SYSTEMS_TESTER_INSTALL_DIR]\integrations\tci
  ```
- Open the project settings dialog and on the “Build” tab set the “Test management” parameter to “Dynamic through Rational Systems Tester GUI”.
- Recompile and rebuild the project.
- Start the test executable and notice the “Execution” tab in “Output” window.

Starting and stopping control parts

Ensure that the test executable is running and that the “Execution” tab is displayed in the “Output” window.

To start a control part of a certain module, select the module in the tree. Then open the “Project” menu and select “Start Control Part”. It is also possible to start the control part if you right-click on a module and select “Start Control Part” from the shortcut menu.

To stop a control part you open the project menu and select “Stop Control Part” or right-click on the module and select “Stop Control Part” from the shortcut menu.

Starting and stopping test cases

Ensure that a test executable is running and that the “Execution” tab is displayed in “Workspace” window.

To start a certain test case, first select it in the tree. Then open the “Project” menu and select “Start Test Case”. It is also possible to right-click the test case and select “Start Test Case” from the shortcut menu. If the test case does not have any parameters it will start immediately.

If the test case has parameters a dialog window will open. This window lists all the test case parameters with name and type. It is required to give values to all of the parameters. The values are defined by string representations in the same way as they are defined in a source TTCN-3 file. When you have defined all values and pressed the “Run” button Rational Systems Tester will create parameter values from the string representation and start the test case.

Description of errors are found on the “TCI” tab of the “Output” window.

To stop a test case you open the Project menu and select “Stop Test Case” or right-click on a test case and select “Stop Test Case” from the shortcut menu.

Defining test plans

A test plan represents a batch of test cases that are executed one by one. Each test case may be executed several number of times. An example of test plan is:

- Execute test case tc_01() 1 time
- Execute test case tc_02(“Maoist”) 3 times
- Execute test case tc_03({1,2,3}) 6 times

IBM Rational Systems Tester projects may have an arbitrary number of test plans. Each test plan is defined in a separate file. In order to define a test plan you first create a new empty test plan using the File menu->New wizard (you may safely close the opened window).

Note

A test plan may be modified and executed only when a test executable is started.

Ensure that a test executable is running and that the “Execution” tab is displayed in “Output” window.

Switch to the “File view” tab, select the test plan file and in the “Project” menu choose “Modify Test Plan” item. It is possible to modify a test plan if you right-click on a test plan file name and choose “Modify Test Plan” in the shortcut menu. The “Test Plan Editing” dialog will appear.

The “Test Plan Editing” dialog consist of a list box with four columns:

- Module name of a test case
- Test case name
- Test case parameters
- Number of iterations

Each line represents a batch step. It may be described using the following TTCN-3 code where strings in “<>” are substituted with those defined by the user:

```
for (i := 0; i < <NUMBER_OF_ITERATIONS>; i := i + 1) {
  execute(<MODULE>.<TESTCASE>(<PARAMETER_LIST>));
}
```

Press the “Add Test Case” button to add an entry to the list. The module name and the test case name will be filled in from the prepared set of available modules and test cases in the test suite. Clicking on the first or second column will open a drop down list box with the available choices. Test case parameters (if any) should be defined in a comma delimited list. Finally the number of test case iterations is set in the last column.

Every entry in the test plan window may be deleted by selecting it and pressing the “Delete Test Case” button.

Press “Save” to confirm changes and close the “Test Plan Editing” dialog.

Press “Cancel” to discard all changes and close the “Test Plan Editing” dialog.

Press “Run” to confirm changes and execute test plan.

If any parameters are defined incorrectly (the length of the list is not correct or a defined string value is not compatible with the type of the formal parameter) then an error will be posted at the time of pressing “Run” or “Save” button

Running test plans

Ensure that the test executable is running and that the “Execution” tab is displayed in the “Workspace” window.

Switch to the “File view” tab, select the test plan file and in the “Project” menu choose “Run Test Plan”. It is possible to run a test plan if you right-click on the test plan file name and choose “Run Test Plan” in the shortcut menu.

Test Plan Internals

Test plans are not automatically synchronized with TTCN-3 files. Every error in a test plan specification (unknown test case name, wrong parameter type, etc.) is detected only when the test plan is saved or executed.

To check a test plan definition against a test suite specification you open the editing window and press the Save button.

The test plan file is a saved as a plain text file.

7

Execution Logs

Log Mechanism

IBM Rational Systems Tester contains several built-in log mechanisms that may be used to trace test suite execution. These are text (ASCII-based) simple and customizable log mechanisms, MSC file logging and graphical execution tracing. The example provided in the installation showing implementation of log mechanism using TTCN-3 standard TCI TL logging interface may be used instead of default text logging.

Simple Text Log

The log format of the built-in text log mechanism is the same whether you execute from the command-line or from the Rational Systems Tester user interface. When executed from the Rational Systems Tester, you are able to navigate to the source of the log messages from the Execution tab in the Output window.

It is also possible to save the execution output to file and open it later.

This log mechanism prints the values of all event parameters. You may use Customizable Text Log mechanism to select certain event parameters and specify your own event definition.

The simple built-in log mechanism logs to `stdout`. Each event has the following format:

```
<filename> (<line number>): [<component name>] <event name>
< <arg1>, <arg2>...<argN> > - <scope name>(<arg1>,
<arg2>...<argN>)
```

Example 20: ----------

```
bar.ttcn (42): [CompA] testcase_started < tc_1 > -
Module3()
```

- <filename> and <line number>

  This is the TTCN-3 source location where the event occurs.

- <component name>

  This is the running component generating the event.

- <event name> and its arguments <arg1>...<argN>

  This is the actual event. The scope information is the function, test case, test step, and so on, in which context you are executing.

When an error is logged, an error code is printed together with an error description. The description of these error codes can be found in “Error Codes Overview” on page 151.

Example 21: ----------

```
mymod.ttcn (42): [Comp1] ERROR GRL0006: Type 'YourType'
must be either integer or float. - testcase29()
```

Logging verbosity

The level of logging verbosity is decided by the predefined configuration key t3rt.logging.builtin.verbosity.

An integer value is assigned to all available logging events. By setting the configuration key t3rt.logging.builtin.verbosity you decide which events that should be logged. If you set the verbosity level to “2”, all logging events with integer values 2 or lower will be listed.

However, if you only want to display a subset of the events with verbosity level “2”, you must customize your logging. This means that you can change which events to log and which events that should not be logged. In the `include` directory in your installation directory, there is a file called `rtconf.cfg` available. This file includes all logging events and their corresponding integer values.

Follow the instructions below to customize you logging verbosity:

1. Copy the file `RTconf.cfg` and paste it at a location of your choice. Change the name of the file.
2. Open the file in a text editor and change the integer values of the events. Save the file.
3. From the **Project** menu, click **Settings**.
4. Select the **Execution** tab and in the **Optional configuration file** field, open your customized file.

See also

“t3rt.logging.builtin.verbosity” on page 129 in Chapter 6, *Execute Tests*

Logging Settings

The logging settings gives you a possibility to make choices for the text logging. You find the settings in the **Logging** tab in the **Settings** dialog box.

You can select one or all of these options in a single execution:

- **Logging settings**
  - **Enable log event generation**

    Allows you to enable or disable all log generation.
  - **Display configuration settings**

    Allows you to display current configuration settings in the Execution tab in the Output window.
  - **Pretty print values**

    Tells text log mechanism to print every field of a structured value on separate line maintaining indentation.
  - **Truncate string/vector values**

    Tells text log mechanism to truncate string (all kinds), recordof, setof and array values to the specified number of elements. If value is truncated it's appended with "...".
  - **Show timestamps**

    Turns on timestamping for every event. Timestamps appear at the beginning of the event log line.
- **Log target**
  - **Write log to**

    Choose where to write log data: to output window, to external file or both.
  - **External log file**

    Specify the name of the file to which log data will be written. Not applicable when log target is set to "Tester output window".

Enable Text-File Logging

1. On the **Project** menu, click **Settings**, then click the **Logging** tab.
2. Select **Enable log event generation**.
3. Select **Log target** and specify a filename if log is written to a file.
4. Click **OK**.

See also

"Set Execution Options" on page 127 in Chapter 6, *Execute Tests*.

Display Log From Text File

Specify which log file to view:

1. On the **Project** menu, click **Settings**, and then the **Misc** tab.
2. Specify the text log that you want to view in the **Post mortem log file** box.
3. Click **OK**.

View a text log file:

- On the **Project** menu, click **Post-mortem debugger**.
- Click the **Post-mortem debugger** button on the toolbar.

Navigate in Execution Log

1. In the Output window, select the **Execution** tab.
2. Press F4 or **SHIFT+F4** to browse log items.

Navigate from Execution Log

1. In the Output window, select the Execution tab.

Right-click an item and select Locate. The corresponding item is located in the abstract test suite.

Customizable Text Log

Customizable text log mechanism provide you with the ability to define log output by specifying format string for every event. Similar to “printf“ function format string contains free text and special parameters that are substituted with real values at runtime. Parameters are enclosed in “%“, e.g. %TimeStamp% denotes event timestamp and is substituted with something like “12:37:59.223“.

Each event has predefined set of parameters. Format string is checked during test suite initialization. Syntax errors in format string as well as use of illegal parameters are reported immediately and execution aborts.

Every event has default format string that is specified in RTConf.cfg file. Use this file as a template if you want to change event format. For example “message_sent“ event has following definition:

```
"%Location%%TimeStamp%[%CompName%] %Event%: %PortName%.send
(%MsgValue%) to '%ToCompName%'"
```

At runtime this definition is substituted with actual parameter values and printed. The resulted string may look like:

```
ts.ttcn (289): 12:37:59.223 [MTC] message_sent: P.send (1)
to 'Cint:1'
```

The behavior of customizable text log is the same as Simple Text Log mechanism. These mechanisms are interchangeable and differ only in the contents of the log messages.

By default customizable log mechanism is used but you may revert back to simple text log by patching t3rts_conditional.c file that is usually located in <Rational Systems Tester Installation Folder>/lib directory. In this case you need to disable registration of customizable log mechanism and enable simple log mechanism registration. This may be simple done by compiling t3rts_conditional.c file with T3RT_LEGACY_BUILTIN_LOG value defined (e.g. /DT3RT_LEGACY_BUILTIN_LOG).

Valid event parameters

Common parameters for all events

"Event"	Event name
"Location"	Source location of the event
"CompName"	Name of the component that produced event
"CompAddr"	Address of the component that produced event
"TimeStamp”	Local time at the moment of event generation

Timestamp is displayed only when RTConf key **t3rt.logging.timestamp.enabled** is set to true.

It’s possible to control how timestamp is presented in the log with the help of user-defined **t3ud_make_timestamp** function. The value returned by this function is put into the log without any changes. The default implementation prints timestamp as “HH:MM:SS.NNN“.

Parameters for “message_sent” event

"PortName"	Local port name
"MsgValue"	Value sent through port
"ToCompName"	Destination component name
"ToCompAddr"	Destination component address

Parameters for “message_sent_mc” event

"PortName"	Local port name
"MsgValue"	Value sent through port
"ToComp-NameList"	List of destination components names
"ToCompAddrList"	List of destination components addresses

Parameters for “message_sent_bc” event

"PortName"	Local port name
"MsgValue"	Value sent through port
"ToComp-NameList"	List of destination components names
"ToCompAddrList"	List of destination components addresses

Parameters for “sut_message_sent” event

"PortName"	Local port name
"MsgValue"	Value sent through port
"SutAddr"	SUT address (if specified)

Parameters for “sut_message_sent_mc” event

"PortName"	Local port name
"MsgValue"	Value sent through port
"SutAddrList"	List of SUT addresses (if specified)

Parameters for “sut_message_sent_bc” event

"PortName"	Local port name
"MsgValue"	Value sent through port

Parameters for “message_sent_failed”

"PortName"	Local port name
"MsgValue"	Value sent through port
"ToCompName"	Destination component name
"ToCompAddr"	Destination component address
"CodecStatus"	Boolean signaling status of encoding operation
"TransmitStatus"	Boolean signaling status of network transmission operation

Parameters for “message_sent_failed_mc”

"PortName"	Local port name
"MsgValue"	Value sent through port
"ToComp-NameList"	List of destination components names
"ToCompAddrList"	List of destination components addresses
"CodecStatus"	Boolean signaling status of encoding operation
"TransmitStatus"	Boolean signaling status of network transmission operation

Parameters for “message_sent_failed_bc”

"PortName"	Local port name
"MsgValue"	Value sent through port
"ToComp-NameList"	List of destination components names

"ToCompAddrList"	List of destination components addresses
"CodecStatus"	Boolean signaling status of encoding operation
"TransmitStatus"	Boolean signaling status of network transmission operation

Parameters for “sut_message_sent_failed” event

"PortName"	Local port name
"MsgValue"	Value sent through port
"SutAddr"	SUT address (if specified)
"CodecStatus"	Boolean signaling status of encoding operation
"TransmitStatus"	Boolean signaling status of network transmission operation

Parameters for “sut_message_sent_failed_mc” event

"PortName"	Local port name
"MsgValue"	Value sent through port
"SutAddrList"	List of SUT addresses (if specified)
"CodecStatus"	Boolean signaling status of encoding operation
"TransmitStatus"	Boolean signaling status of network transmission operation

Parameters for “sut_message_sent_failed_bc” event

"PortName"	Local port name
"MsgValue"	Value sent through port
"CodecStatus"	Boolean signaling status of encoding operation
"TransmitStatus"	Boolean signaling status of network transmission operation

Parameters for “message_detected” event

"PortName"	Local port name
“MsgData"	Binary string detected at port
"FromCompName"	Sending component name
"FromCompAddr"	Sending component address

Parameters for “sut_message_detected” event

"PortName"	Local port name
“MsgData"	Binary string detected at port
"SutAddr"	SUT address (if specified)

Parameters for “message_received” event

"PortName"	Local port name
"MsgValue"	Value received through port
"MsgTemplate"	Template specified in the receive operation (if specified)
"FromCompName"	Sending component name
"FromCompAddr"	Sending component address

Parameters for “message_found” event

"PortName"	Local port name
"MsgValue"	Value received through port
"MsgTemplate"	Template specified in the check(receive) operation (if specified)
"FromCompName"	Sending component name
"FromCompAddr"	Sending component address

Parameters for “message_discarded” event

"PortName"	Local port name
"MsgValue"	Value received through port
"MsgTemplate"	Template specified in the trigger operation (if specified)
"FromCompName"	Sending component name
"FromCompAddr"	Sending component address

Parameters for “sut_message_received” event

"PortName"	Local port name
"MsgValue"	Value received through port
"MsgTemplate"	Template specified in the receive operation (if specified)
"SutAddr"	SUT address (if specified)

Parameters for “sut_message_found” event

"PortName"	Local port name
"MsgValue"	Value received through port
"MsgTemplate"	Template specified in the check(receive) operation (if specified)
"SutAddr"	SUT address (if specified)

Parameters for “sut_message_discarded” event

"PortName"	Local port name
"MsgValue"	Value received through port
"MsgTemplate"	Template specified in the trigger operation (if specified)
"SutAddr"	SUT address (if specified)

Parameters for “sut_action_performed” event

"Action"	SUT action performed

Parameters for “call_initiated” event

"PortName"	Local port name
"ProcName"	Name of called procedure
"Signature"	Signature value (includes values of all parameters)
"ToCompName"	Destination component name
"ToCompAddr"	Destination component address

Parameters for “call_initiated_mc” event

"PortName"	Local port name
"ProcName"	Name of called procedure
"Signature"	Signature value (includes values of all parameters)
"ToComp-NameList"	List of destination components names
"ToCompAddrList"	List of destination components addresses

Parameters for “call_initiated_bc” event

"PortName"	Local port name
"ProcName"	Name of called procedure
"Signature"	Signature value (includes values of all parameters)
"ToComp-NameList"	List of destination components names
"ToCompAddrList"	List of destination components addresses

Parameters for “sut_call_initiated” event

"PortName"	Local port name
"ProcName"	Name of called procedure
"Signature"	Signature value (includes values of all parameters)
"SutAddr"	SUT address (if specified)

Parameters for “sut_call_initiated_mc” event

"PortName"	Local port name
"ProcName"	Name of called procedure
"Signature"	Signature value (includes values of all parameters)
"SutAddrList"	List of SUT addresses (if specified)

Parameters for “sut_call_initiated_bc” event

"PortName"	Local port name
"ProcName"	Name of called procedure
"Signature"	Signature value (includes values of all parameters)

Parameters for “call_failed” event

"PortName"	Local port name
"ProcName"	Name of called procedure
"Signature"	Signature value (includes values of all parameters)
"ToCompName"	Destination component name
"ToCompAddr"	Destination component address
"CodecStatus"	Boolean signaling status of encoding operation
"TransmitStatus"	Boolean signaling status of network transmission operation

Parameters for “call_failed_mc” event

"PortName"	Local port name
"ProcName"	Name of called procedure
"Signature"	Signature value (includes values of all parameters)
"ToComp-NameList"	List of destination components names
"ToCompAddrList"	List of destination components addresses
"CodecStatus"	Boolean signaling status of encoding operation
"TransmitStatus"	Boolean signaling status of network transmission operation

Parameters for “call_failed_bc” event

"PortName"	Local port name
"ProcName"	Name of called procedure
"Signature"	Signature value (includes values of all parameters)

"ToComp-NameList"	Liast of destination components names
"ToCompAddrList"	List of destination components addresses
"CodecStatus"	Boolean signaling status of encoding operation
"TransmitStatus"	Boolean signaling status of network transmission operation

Parameters for “sut_call_failed” event

"PortName"	Local port name
"ProcName"	Name of called procedure
"Signature"	Signature value (includes values of all parameters)
"SutAddr"	SUT address (if specified)
"CodecStatus"	Boolean signaling status of encoding operation
"TransmitStatus"	Boolean signaling status of network transmission operation

Parameters for “sut_call_failed_mc” event

"PortName"	Local port name
"ProcName"	Name of called procedure
"Signature"	Signature value (includes values of all parameters)
"SutAddrList"	List of SUT addresses (if specified)
"CodecStatus"	Boolean signaling status of encoding operation
"TransmitStatus"	Boolean signaling status of network transmission operation

Parameters for “sut_call_failed_bc” event

"PortName"	Local port name
"ProcName"	Name of called procedure
"Signature"	Signature value (includes values of all parameters)
"CodecStatus"	Boolean signaling status of encoding operation
"TransmitStatus"	Boolean signaling status of network transmission operation

Parameters for “call_timed_out” event

"PortName"	Local port name

Parameters for “sut_call_timed_out” event

"PortName"	Local port name

Parameters for “call_detected” event

"PortName"	Local port name
"ProcName"	Name of called procedure
"SignatureData"	Encoded signature value (includes encoded actual parameters)
"FromCompName"	Sending component name
"FromCompAddr"	Sending component address

Parameters for “sut_call_detected” event

"PortName"	Local port name
"ProcName"	Name of called procedure
"SignatureData"	Encoded signature value (includes encoded actual parameters)
"SutAddr"	SUT address (if specified)

Parameters for “call_received” event

"PortName"	Local port name
"ProcName"	Name of called procedure
"Signature"	Signature value (includes values of all parameters)
"SignatureTemplate"	Template specified in the getcall operation (if specified)
"FromCompName"	Sending component name
"FromCompAddr"	Sending component address

Parameters for “call_found” event

"PortName"	Local port name
"ProcName"	Name of called procedure
"Signature"	Signature value (includes values of all parameters)
"SignatureTemplate"	Template specified in the check(getcall) operation (if specified)
"FromCompName"	Sending component name
"FromCompAddr"	Sending component address

Parameters for “sut_call_received” event

"PortName"	Local port name
"ProcName"	Name of called procedure
"Signature"	Signature value (includes values of all parameters)
"SignatureTemplate"	Template specified in the getcall operation (if specified)
"SutAddr"	SUT address (if specified)

Parameters for “sut_call_found” event

"PortName"	Local port name
"ProcName"	Name of called procedure
"Signature"	Signature value (includes values of all parameters)
"SignatureTemplate"	Template specified in the check(getcall) operation (if specified)
"SutAddr"	SUT address (if specified)

Parameters for “reply_sent” event

"PortName"	Local port name
"ProcName"	Name of the procedure for which the reply is sent
"Signature"	Signature value (includes values of all parameters and the return value)

"ReturnValue"	The procedure return value (if any)
"ToCompName"	Destination component name
"ToCompAddr"	Destination component address

Parameters for “reply_sent_mc” event

"PortName"	Local port name
"ProcName"	Name of the procedure for which the reply is sent
"Signature"	Signature value (includes values of all parameters and the return value)
"ReturnValue"	The procedure return value (if any)
"ToComp-NameList"	List of destination components names
"ToCompAddrList"	List of destination components addresses

Parameters for “reply_sent_bc” event

"PortName"	Local port name
"ProcName"	Name of the procedure for which the reply is sent
"Signature"	Signature value (includes values of all parameters and the return value)
"ReturnValue"	The procedure return value (if any)
"ToComp-NameList"	List of destination components names
"ToCompAddrList"	List of destination components addresses

Parameters for “sut_reply_sent” event

"PortName"	Local port name
"ProcName"	Name of the procedure for which the reply is sent
"Signature"	Signature value (includes values of all parameters and the return value)
"ReturnValue"	The procedure return value (if any)
"SutAddr"	SUT address (if specified)

Parameters for “sut_reply_sent_mc” event

"PortName"	Local port name
"ProcName"	Name of the procedure for which the reply is sent
"Signature"	Signature value (includes values of all parameters and the return value)
"ReturnValue"	The procedure return value (if any)
"SutAddrList"	List of SUT addresses (if specified)

Parameters for “sut_reply_sent_bc” event

"PortName"	Local port name
"ProcName"	Name of the procedure for which the reply is sent
"Signature"	Signature value (includes values of all parameters and the return value)
"ReturnValue"	The procedure return value (if any)

Parameters for “reply_failed” event

"PortName"	Local port name
"ProcName"	Name of the procedure for which the reply is sent
"Signature"	Signature value (includes values of all parameters and the return value)
"ReturnValue"	The procedure return value (if any)
"ToCompName"	Destination component name
"ToCompAddr"	Destination component address
"CodecStatus"	Boolean signaling status of encoding operation
"TransmitStatus"	Boolean signaling status of network transmission operation

Parameters for “reply_failed_mc” event

"PortName"	Local port name
"ProcName"	Name of the procedure for which the reply is sent
"Signature"	Signature value (includes values of all parameters and the return value)
"ReturnValue"	The procedure return value (if any)
"ToComp-NameList"	List of destination components names
"ToCompAddrList"	List of destination components addresses
"CodecStatus"	Boolean signaling status of encoding operation
"TransmitStatus"	Boolean signaling status of network transmission operation

Parameters for “reply_failed_bc” event

"PortName"	Local port name
"ProcName"	Name of the procedure for which the reply is sent
"Signature"	Signature value (includes values of all parameters and the return value)
"ReturnValue"	The procedure return value (if any)
"ToComp-NameList"	List of destination components names
"ToCompAddrList"	List of destination components addresses
"CodecStatus"	Boolean signaling status of encoding operation
"TransmitStatus"	Boolean signaling status of network transmission operation

Parameters for “sut_reply_failed” event

"PortName"	Local port name
"ProcName"	Name of the procedure for which the reply is sent
"Signature"	Signature value (includes values of all parameters and the return value)

"ReturnValue"	The procedure return value (if any)
"SutAddr"	SUT address (if specified)
"CodecStatus"	Boolean signaling status of encoding operation
"TransmitStatus"	Boolean signaling status of network transmission operation

Parameters for “sut_reply_failed_mc” event

"PortName"	Local port name
"ProcName"	Name of the procedure for which the reply is sent
"Signature"	Signature value (includes values of all parameters and the return value)
"ReturnValue"	The procedure return value (if any)
"SutAddrList"	List of SUT addresses (if specified)
"CodecStatus"	Boolean signaling status of encoding operation
"TransmitStatus"	Boolean signaling status of network transmission operation

Parameters for “sut_reply_failed_bc” event

"PortName"	Local port name
"ProcName"	Name of the procedure for which the reply is sent
"Signature"	Signature value (includes values of all parameters and the return value)
"ReturnValue"	The procedure return value (if any)
"CodecStatus"	Boolean signaling status of encoding operation
"TransmitStatus"	Boolean signaling status of network transmission operation

Parameters for “reply_detected” event

"PortName"	Local port name
"ProcName"	Name of the procedure for which the reply is detected
"SignatureData"	Encoded signature value (includes encoded actual parameters)
"ReturnValueData"	Encoded return value (if any)
"FromCompName"	Sending component name
"FromCompAddr"	Sending component address

Parameters for “sut_reply_detected” event

"PortName"	Local port name
"ProcName"	Signature name
"SignatureData"	Encoded signature value (includes encoded actual parameters)
"ReturnValueData"	Encoded return value (if any)
"SutAddr"	SUT address (if specified)

Parameters for “reply_received” event

"PortName"	Local port name
"ProcName"	Name of procedure for which the reply is received
"Signature"	Signature value (includes values for all parameters and the return value)
"SignatureTemplate"	Template specified in the getreply operation (if specified)
"ReturnValue"	Return value (if any)
"ReturnValueTemplate"	Template for return value specified in the getreply operation (if specified)
"FromCompName"	Sending component name
"FromCompAddr"	Sending component address

Parameters for “reply_found” event

"PortName"	Local port name
"ProcName"	Name of procedure for which the reply is received
"Signature"	Signature value (includes values for all parameters and the return value)
"SignatureTemplate"	Template specified in the check(getreply) operation (if specified)
"ReturnValue"	Return value (if any)

"ReturnValueTemplate"	Template for return value specified in the getreply operation (if specified)
"FromCompName"	Sending component name
"FromCompAddr"	Sending component address

Parameters for “sut_reply_received” event

"PortName"	Local port name
"ProcName"	Name of procedure for which the reply is received
"Signature"	Signature value (includes values for all parameters and the return value)
"SignatureTemplate"	Template specified in the getreply operation (if specified)
"ReturnValue"	Return value (if any)
"ReturnValueTemplate"	Template for return value specified in the getreply operation (if specified)
"SutAddr"	SUT address (if specified)

Parameters for “sut_reply_found” event

"PortName"	Local port name
"ProcName"	Name of procedure for which the reply is received
"Signature"	Signature value (includes values for all parameters and the return value)
"SignatureTemplate"	Template specified in the getreply operation (if specified)
"ReturnValue"	Return value (if any)
"ReturnValueTemplate"	Template for return value specified in the getreply operation (if specified)
"SutAddr"	SUT address (if specified)

Parameters for “exception_raised” event

"PortName"	Local port name
"SignatureName"	Name of the procedure for which the exception is raised
"Exception"	Exception value
"ToCompName"	Destination component name
"ToCompAddr"	Destination component address

Parameters for “exception_raised_mc” event

"PortName"	Local port name
"SignatureName"	Name of the procedure for which the exception is raised
"Exception"	Exception value
"ToComp-NameList"	List of destination components names
"ToCompAddrList"	List of destination components addresses

Parameters for “exception_raised_bc” event

"PortName"	Local port name
"SignatureName"	Name of the procedure for which the exception is raised
"Exception"	Exception value
"ToComp-NameList"	List of destination components names
"ToCompAddrList"	List of destination components addresses

Parameters for “sut_exception_raised” event

"PortName"	Local port name
"SignatureName"	Name of the procedure for which the exception is raised
"Exception"	Exception value
"SutAddr"	SUT address (if specified)

Parameters for “sut_exception_raised_mc” event

"PortName"	Local port name
"SignatureName"	Name of the procedure for which the exception is raised
"Exception"	Exception value
"SutAddrList"	List of SUT addresses (if specified)

Parameters for “sut_exception_raised_bc” event

"PortName"	Local port name
"SignatureName"	Name of the procedure for which the exception is raised
"Exception"	Exception value

Parameters for “raise_failed” event

"PortName"	Local port name
"SignatureName"	Name of the procedure for which the exception is raised
"Exception"	Exception value
"ToCompName"	Destination component name
"ToCompAddr"	Destination component address
"CodecStatus"	Boolean signaling status of encoding operation
"TransmitStatus"	Boolean signaling status of network transmission operation

Parameters for “raise_failed_mc” event

"PortName"	Local port name
"SignatureName"	Name of the procedure for which the exception is raised
"Exception"	Exception value
"ToComp-NameList"	List of destination components names
"ToCompAddrList"	List of destination components addresses
"CodecStatus"	Boolean signaling status of encoding operation
"TransmitStatus"	Boolean signaling status of network transmission operation

Parameters for “raise_failed_bc” event

"PortName"	Local port name
"SignatureName"	Name of the procedure for which the exception is raised
"Exception"	Exception value
"ToComp-NameList"	List of destination components names
"ToCompAddrList"	List of destination components addresses
"CodecStatus"	Boolean signaling status of encoding operation
"TransmitStatus"	Boolean signaling status of network transmission operation

Parameters for “sut_raise_failed” event

"PortName"	Local port name
"SignatureName"	Name of the procedure for which the exception is raised
"Exception"	Exception value
"SutAddr"	SUT address (if specified)
"CodecStatus"	Boolean signaling status of encoding operation
"TransmitStatus"	Boolean signaling status of network transmission operation

Parameters for “sut_raise_failed_mc” event

"PortName"	Local port name
"SignatureName"	Name of the procedure for which the exception is raised
"Exception"	Exception value
"SutAddrList"	List of SUT addresses (if specified)
"CodecStatus"	Boolean signaling status of encoding operation
"TransmitStatus"	Boolean signaling status of network transmission operation

Parameters for “sut_raise_failed_bc” event

"PortName"	Local port name
"SignatureName"	Name of the procedure for which the exception is raised
"Exception"	Exception value
"CodecStatus"	Boolean signaling status of encoding operation
"TransmitStatus"	Boolean signaling status of network transmission operation

Parameters for “exception_detected” event

"PortName"	Local port name
"SignatureName"	Name of the procedure for which the exception is detected
"ExceptionData"	Encoded exception value
"FromCompName"	Sending component name
"FromCompAddr"	Sending component address

Parameters for “sut_exception_detected” event

"PortName"	Local port name
"SignatureName"	Name of the procedure for which the exception is detected
"ExceptionData"	Encoded exception value
"SutAddr"	SUT address (if specified)

Parameters for “timeout_exception_detected” event

"PortName"	Local port name
"SignatureName"	Name of the procedure for which the timeout exception is detected

Parameters for “sut_timeout_exception_detected” event

"PortName"	Local port name
"SignatureName"	Name of the procedure for which the timeout exception is detected

Parameters for “exception_caught” event

"PortName"	Local port name
"SignatureName"	Name of the procedure for which the exception is received
"Exception"	Exception value
"ExceptionTemplate"	Template specified in the catch operation (if specified)
"FromCompName"	Sending component name
"FromCompAddr"	Sending component address

Parameters for “exception_found” event

"PortName"	Local port name
"SignatureName"	Name of the procedure for which the exception is received
"Exception"	Exception value
"ExceptionTemplate"	Template specified in the check(catch) operation (if specified)
"FromCompName"	Sending component name
"FromCompAddr"	Sending component address

Parameters for “sut_exception_caught” event

"PortName"	Local port name
"SignatureName"	Name of the procedure for which the exception is received

"Exception"	Exception value
"ExceptionTemplate"	Template specified in the catch operation (if specified)
"SutAddr"	SUT address (if specified)

Parameters for “sut_exception_found” event

"PortName"	Local port name
"SignatureName"	Signature name
"Exception"	Exception value
"ExceptionTemplate"	Template specified in the check(catch) operation (if specified)
"SutAddr"	SUT address (if specified)

Parameters for “timeout_exception_caught” event

"PortName"	Local port name
"SignatureName"	Name of the procedure for which the timeout exception is received

Parameters for “timeout_exception_found” event

"PortName"	Local port name
"SignatureName"	Name of the procedure for which the timeout exception is received

Parameters for “sut_timeout_exception_caught” event

"PortName"	Local port name
"SignatureName"	Name of the procedure for which the timeout exception is detected

Parameters for “sut_timeout_exception_found” event

"PortName"	Local port name
"SignatureName"	Name of the procedure for which the timeout exception is detected

Parameters for “timer_started” event

"TimerName"	Name of the started timer
"Duration"	Timer duration

Parameters for “timer_stopped” event

"TimerName"	Name of the stopped timer

Parameters for “timer_read” event

"TimerName"	Name of the timer read
"Duration"	Overall timer duration
"Elapsed"	Time elapsed since timer start
"State"	Current timer state

Parameters for “timer_is_running” event

"TimerName"	Name of the timer
"IsRunning"	Boolean signaling whether timer is running
"Duration"	Overall timer duration
"Elapsed"	Time elapsed since timer start
"State"	Current timer state

Parameters for “timeout_detected” event

"TimerName"	Name of the timer for which timeout is detected

Parameters for “timeout_received” event

"TimerName"	Name of the timer for which timeout is received
"Duration"	Overall timer duration
"TimerTemplate"	Template specified in the timeout operation

Parameters for “timeout_mismatch” event

"TimerName"	Name of the timer for which timeout is checked
"TimerTemplate"	Template specified in the timeout operation

Parameters for “component_created” event

"PtcType"	Name of the created component type
"PtcName"	Name of the created component
"PtcAddr"	Address of the created component

Main test component is assigned with “MTC“ name. Parallel test component by default are assigned with “TypeName:<Ordinal Number>“. However if user specified component name in the ‘create’ operation then user-defined name is shown.

Parameters for “component_started” event

"PtcName"	Name of the started component
"PtcAddr"	Address of the started component
"FunctionName"	Name of the invoked function
"Signature"	Signature value of the invoked function (includes values for all actual parameters)

Parameters for “component_is_running” event

"PtcName"	Name of the checked component
"PtcAddr"	Address of the checked component
"IsRunning"	Boolean signaling whether component is running

Parameters for “component_is_alive” event

"PtcName"	Name of the checked component
"PtcAddr"	Address of the checked component
"IsAlive"	Boolean signaling whether component is alive

Parameters for “component_stopped” event

"PtcName"	Name of the stopped component
"PtcAddr"	Address of the stopped component

Parameters for “component_killed” event

"PtcName"	Name of the killed component
"PtcAddr"	Address of the killed component

Parameters for “component_terminated” event

None

Parameters for “done_check_failed” event

"PtcName"	Name of the checked component
"PtcAddr"	Address of the checked component
"PtcTemplate"	Template specified in the done operation

Parameters for “done_check_succeeded” event

"PtcName"	Name of the checked component
"PtcAddr"	Address of the checked component
"PtcTemplate"	Template specified in the done operation

Parameters for “kill_check_failed” event

"PtcName"	Name of the checked component
"PtcAddr"	Address of the checked component
"PtcTemplate"	Template specified in the killed operation

Parameters for “kill_check_succeeded” event

"PtcName"	Name of the checked component
"PtcAddr"	Address of the checked component
"PtcTemplate"	Template specified in the killed operation

Parameters for “port_connected” event

"PtcName1"	Name of the component specified in the first parameter of connect operation
"PtcAddr1"	Address of the component specified in the first parameter of connect operation
"PortName1"	Name of the port specified in the first parameter of connect operation

"PtcName2"	Name of the component specified in the second parameter of connect operation
"PtcAddr2"	Address of the component specified in the second parameter of connect operation
"PortName2"	Name of the port specified in the second parameter of connect operation

Parameters for “port_disconnected” event

"PtcName1"	Name of the component specified in the first parameter of disconnect operation
"PtcAddr1"	Address of the component specified in the first parameter of disconnect operation
"PortName1"	Name of the port specified in the first parameter of disconnect operation
"PtcName2"	Name of the component specified in the second parameter of disconnect operation
"PtcAddr2"	Address of the component specified in the second parameter of disconnect operation
"PortName2"	Name of the port specified in the second parameter of disconnect operation

Parameters for “port_mapped” event

"PtcName"	Name of the component specified in the map operation
"PtcAddr"	Address of the component specified in the map operation
"PortName"	Name of the port specified in the map operation
"SysPortName"	Name of the TSI port specified in the map operation

Parameters for “port_unmapped” event

"PtcName"	Name of the component specified in the unmap operation
"PtcAddr"	Address of the component specified in the unmap operation
"PortName"	Name of the port specified in the unmap operation
"SysPortName"	Name of the TSI port specified in the unmap operation

Parameters for “port_enabled” event

"PortName"	Name of the started port

Parameters for “port_disabled” event

"PortName"	Name of the stopped port

Parameters for “port_halted” event

"PortName"	Name of the halted port

Parameters for “port_cleared” event

"PortName"	Name of the cleared port

Parameters for “scope_entered” event

"ScopeName"	Name of the entered scope (function, altstep, testcase, etc)
"ScopeKind"	Scope kind
"Signature"	Signature value of the entered scope (includes values of all formal parameters)

Parameters for “scope_left” event

"ScopeName"	Name of the scope left (function, altstep, testcase, etc)
"ScopeKind"	Scope kind
"Signature"	Signature value of the scope left (includes values of all formal parameters and the return value)

Parameters for “scope_changed” event

None

Parameters for “alternative_activated” event

"AltstepName"	Name of the activated altstep
"Signature"	Signature value of the activated altstep (includes values of all actual parameters)
"DefaultRef"	Default reference value of the activated alstep

Parameters for “alternative_deactivated” event

"DefaultRef"	Default reference value of the deactivated alstep

Parameters for “local_verdict_changed” event

"OldVerdict"	Previous local component verdict
"NewVerdict"	Verdict attempted to set on the component

Value of “NewVerdict” may be better than than the value of “OldVerdict“ but this doesn’t mean that verdict changes to better one (e.g. fail->pass).

Parameters for “local_verdict_queried” event

"Verdict"	Local component verdict

Parameters for “variable_modified” event

"VarName"	Name of the modified variable
"NewValue"	New value of the variable

“VarName“ denotes the name of the whole (base) variable. It means that even if certain field of the compound variable has been modified then the whole variable name and value apear in this event.

Parameters for “template_match_begin” event

"Value"	Value (field) matched against the template
"ValueName"	Value (field) name that is matched
"Template"	Template used

Parameters for “template_match_end” event

"Status"	Boolean signaling the result of matching value (field) against template

Parameters for “template_match_failed” event

"Value"	Value (field) matched against the template
"ValueName"	Value (field) name that is matched
"Template"	Template used

Parameters for “template_mismatch” event

"Value"	Value (field) matched against the template
"ValueName"	Value (field) name that is matched
"Template"	Template used
"Reason"	Reason of mismatch

Parameters for “sender_mismatch” event

"FromCompName"	Actual sending component name (for intercomponent operation)
"ActualSenderAddr"	Actual sender address (SUT address for SUT operation)
"ExpectedSenderAddr"	Expected sender address (SUT address for SUT operations)

Parameters for “testcase_started” event

"TestcaseName"	Name of the test case started
"Signature"	Signature value of the test case started (includes values of all formal parameters)
"Timeout"	Test case timeout

Parameters for “testcase_ended” event

"TestcaseName"	Name of the test case started
"Signature"	Signature value of the test case started (includes values of all formal parameters)
"Verdict"	Test case verdict

Parameters for “testcase_timed_out” event

"TestcaseName"	Name of the timed out test case
"Timeout"	Test case timeout

Parameters for “testcase_error” event

"TestcaseName"	Name of the test case
"ErrorMsg"	Error description

Parameters for “testcase_verdict” event

"TestcaseName"	Name of the test case
"Verdict"	Test case verdict

Parameters for “message_encoded” event

"Value"	Value that is encoded
"Data"	Encoded data (binary string)
"Strategy"	Strategy used for encoding operation (built-in or user-defined encoder)

Parameters for “message_encode_failed” event

"Value"	Value failed to be encoded
"Strategy"	Strategy used for encoding operation (built-in or user-defined encoder)

Parameters for “message_decoded” event

"Value"	Value that is decoded
"Data"	Decoded data (binary string)
"Strategy"	Strategy used for encoding operation (built-in or user-defined encoder)

Parameters for “message_decode_failed” event

"Data"	Data failed to be decoded
"Strategy"	Strategy used for encoding operation (built-in or user-defined encoder)

Parameters for “info_message” event

"Message"	Information message
"Kind"	Message kind

Parameters for “warning_message” event

"Message"	Warning message
"Kind"	Message kind

Parameters for “error_message” event

"Message"	Error message
"Kind"	Message kind

Parameters for “debug_message” event

"Message"	Debug message
"Kind"	Message kind

Parameters for “ttcn3_message” event

"Message"	TTCN-3 log message
"Kind"	Message kind

Parameters for “alt_entered” event

None

Parameters for “alt_left” event

None

Parameters for “alt_rejected” event

None

Parameters for “alt_else” event

None

Parameters for “alt_defaults” event

None

Parameters for “alt_repeat” event

None

Parameters for “alt_wait” event

None

Parameters for “function_call” event

"FunctionName"	Name of the invoked function
"Signature"	Signature value of the invoked function (includes values of all actual parameters)

Parameters for “external_function_call” event

"FunctionName"	Name of the invoked external function
"Signature"	Signature value of the invoked external function (includes values of all actual parameters)

Parameters for “altstep_call” event

"AltstepName"	Name of the invoked altstep
"Signature"	Signature value of the invoked altstep (includes values of all actual parameters)

MSC File Log

A generated MSC file log mainly contains dynamic information. This way, the execution trace may be made available in a graphical format so that you easily can see what messages were sent from where to where, what happened when, and where the execution stopped and why, and so on.

There is one MSC-file created for each executed test case. It is named like this:

```
[name of the TTCN-3-module]_[name of the test case]_[number
of execution times].mpr
```

For example, execution of TTCN-3 Tutorial will produce 3 files:

```
TutorialModule_TC1a_0.mpr
TutorialModule_TC2s_1.mpr
TutorialModule_TCTimer_2.mpr
```

The MSC format conforms to the MSC96 standard (ITU_T Z.120) in the MSC-PR-format.

Enable MSC Logging

To enable MSC logging, you have two options depending on how your project is managed. If you are using mcfg file to configure project then set MSC_LOGGING option to “yes” (MSC_LOGGING=yes) in mcfg file. This will add T3RT_MSC96_EVENT_LOG to the list of defined compiler switches in the generated makefile.

On Windows generated makefile will look smth like:

- Change the switch to:

  ```
  CFLAGS = /c /W2 /MT /Zi /DT3RT_MSC96_EVENT_LOG
  ```

On UNIX generated makefile will look smth like:

- Change the switch to:

  ```
  CFLAGS = -c -W2 -MT -Zi -DT3RT_MSC96_EVENT_LOG
  ```

If you are configuring project through makefile options then you need to set “Generate MSC logging“ check on “General“ tab.

In both cases project has to be recompiled and rebuilt.

Convert MSC files into sequence diagrams

With MSC run-time tracing enabled in TTCN-3 project settings the execution of an ETS produces one trace for every executed test case. Each MSC trace is stored in a separate text file.

The `.mpr` files are produced only when graphical execution tracing is disabled. If an ETS is started from the Rational Systems Tester GUI with graphical tracing enabled there will be sequence diagrams generated in real-time during the execution.

It is possible to open an `.mpr` file in a text editor if you select the file and in the **File** menu point to **Open**.

There are two ways for converting `mpr` files into sequence diagrams. The typical approach is to click **MSC File Trace** on the **Project** menu, select a set of mpr files and click **OK**. All selected files will be automatically converted. Alternative approach is to add mpr files to the project and activate them one by one by double-clicking on them:

- Open TTCN-3 project
- Add the `.mpr` file to the project. This can be done from the **Project** menu. Point to **Add to Project** and then choose **Files**.
- Double click on the `.mpr` filename in the workspace window

The `.mpr` file will be converted to a sequence diagrams which will be opened in the desktop area.

Display Log Files

Specify which log file to view:

1. On the **Project** menu, click **Settings**, and then the **Misc** tab.
2. Specify the MSC file that you want to view in the **Input MSC file** box.
3. Click **OK**.

View an MSC log file:

- On the **Project** menu, click **MSC File Trace**.
- Click the **MSC File Trace** button on the toolbar.
- Add MSC files to your project. Then double-click them in the File View to open them.

Graphical Execution Tracing

Graphical execution tracing is based on MSC File Log. It processes MSC statements in real-time, converts them into sequence diagrams and displays on screen. No mpr files are generated if graphical tracing is enabled.

For each test case and test case run separate diagram is produced.

If you closed diagram window you may always restore it through “Model View“ tab in the workspace window. Open “Model“ subtree and locate “Debug Trace“ package. All diagrams are placed in this package.

You have to manually save diagrams into external file if you wish to work with them later. Right click on “Debug Trace“ and choose “Save in New File“. Rational Systems Tester creates U2 file for all diagrams an puts it into the project.

TTCN-3 specific diagram attributes

Each event on the diagram has additoinal attributes that represent event timestamp and event location in the TTCN-3 code. You may view these attributes by right clicking on the diagram element and choosing “Properties“ item in the popup window. All available attributes are displayed in the “Comments” list box.

Note

Rational Systems Tester extracts attributes from the specific information that is generated into mpr files. Files generated with previous Rational Systems Tester versions may lack information and some attributes may be missing. Consider regenerating mpr files using latest Rational Systems Tester version.

Value of top attribute from the “Comments” list box is displayed in the tooltop window when you hover mouse over diagram element corresponding to event. You may force displaying certain attribute by reordering elements in the list box.

Enable Graphical Execution Tracing

To enable graphical execution tracing you have to enable MSC logging first. See Enable MSC Logging for instructions.

After that go to project settings and open “Execution“ tab. Set “Enable run-time MSC tracing“ check to enable graphical tracing and start execution of test suite.

TCI TL logging

TCI TL logging is enabled by default. This means that whether a project is build with the test management option set to “Dynamic through Rational Systems Tester GUI” or “Static using TCI/TM functions in ETS” all of the TCI TL functions (prefixed with “`tli`”) will be called as long as the corresponding event happens inside the runtime system.

Since the calls to the TCI TL functions may affect the overall performance of a test suite it is possible to disable the TCI TL logging.

TCI TL log mechanism may be disabled at runtime using check box on the Logging tab in the project settings or by providing the following command line parameter to the test executable:

```
-t3rt "-confbool t3rt.tci.tl.enabled false" .
```

This parameter may be specified in “Additional execution switches” project option (in the Project Settings dialog, “Execution” tab).

XML Logging

IBM Rational Systems Tester supports XML logging as defined in Part 6 of TTCN-3 standard “TTCN-3 Control Interfaces (TCI)“. One of the major advantages of the XML logging over generic ASCII logging is that it provides information about mismatches between a message and a template in very convenient way. XML logging is implemented in the dedicated log mechanism that should be registered in the runtime system. XML data generated during test suite exeuction is passed to the user-provided event handling functions that may write it to an external file or zip and pass to another machine in a network. Amount of generated data may be controlled by assigning a severity level (integer number) to every event and then filtering out all events that have severity less than specified during test suite execution.

Enable XML logging

XML logging may be enabled either using project setting in Rational Systems Tester or manually in project mcfg file.

Enable XML logging in project settings

To enable XML logging in Rational Systems Tester:

- Open project settings, switch to Build tab, enter Makefile Options dialog and check “Generate XML logging”
- Leave Makefile Options dialog and switch to Logging tab. Check “Enable XML Logging” and provide the name for the target file. If you leave this field empty log will be written to ExecutionLog.xml file.

Enable XML logging in project configuration file

To enable XML logging using project configuration file:

- Open mcfg file for your project (e.g. make_win.mcfg)
- Add XML_LOGGING=yes to the file header (before SECTION part)

Pretty print XML data

XML log mechanism may format output data by indenting each tag. This makes it possible to review XML data in generic text editor but increases size of the output file. Note that major internet browsers reformat XML data therfore there is no need to pretty print XML data if viewing it in the browser.

Open project settings, switch to Logging tab and check “Pretty print XML data“ to get XML data in the indented format.

Control amount of XML data

You may control amount of XML data by enabling/disabling certain events at runtime in the similar way as it’s done for built-in ASCII logging. See Logging verbosity section for details. However event levels as well as verbosity level may be specified for XML log mechanism independently from ASCII logging.

Note

Generation of XML data for mismatch events requires having “template_match_failed” event enabled. Levels assigned to “template_match_begin”, “template_match_end” and “template_mismatch” events are ignored and implicitly set to the level of “template_match_failed” event.

Redirect XML data stream

When registering XML log mechanism in the runtime system one has to provide XML data handlers (function pointers) that will receive portions of XML data and consume it appropriately (write to a file, send to network machine, display in the window).

Three valid (not NULL) function pointers must be provided to XML log registration function:

- `pInitFunc` will be called once during initialization and will provide XML data header. This function receives reference to context therefore RTConf data may be queried at this moment.

- `pEventFunc` will be called for every event providing complete XML data block representing exactly one event. XML data describing one event is never splitted. Two events never merged. This function must be thread safe since calls to it may overlap. This function receives reference to context.
- `pFinalizeFunc` will be called once during test suite finalization and will provide XML data footer. This function doesn't provide context.

Below is the example implementation of these three functions taken from t3rts_conditional.c file (error handling was removed to shorten example):

Example 22

```
void
t3rt_register_provided_logging (int argc, char * argv [])
{
  // register XML log mechanism in the runtime system
  t3rt_register_xml_log_mechanism(&xmlLogInit,
                                  &xmlLogEvent,
                                  &xmlLogFinalize);
}

//
// XML data handlers
//

void xmlLogInit(const char *xmlLogHeader,
                unsigned long headerSize,
                t3rt_context_t ctx)
{
  gpXmlLogOutfile = fopen(gpXmlLogOutname, "w")
  fwrite(xmlLogHeader, 1, headerSize, gpXmlLogOutfile);
}

void xmlLogEvent(const char *xmlEventData,
                 unsigned long dataSize,
                 t3rt_context_t ctx)
{
  fwrite(xmlEventData, 1, dataSize, gpXmlLogOutfile);
}

void xmlLogFinalize(const char *xmlLogFooter,
                    unsigned long footerSize)
{
  fwrite(xmlLogFooter, 1, footerSize, gpXmlLogOutfile);
  fclose(gpXmlLogOutfile);
}
```

XML log schema

XML log is generated according to the standard schema defined in Part 6 of TTCN-3 standard Version 3.3.1. XSD files containing schema are delivered with Rational Systems Tester and located in [SYSTEMS_TESTER_INSTALLATION_DIR]/etc directory. The files are:

- SimpleTypes.xsd
- Types.xsd
- Values.xsd
- Templates.xsd
- Events.xsd
- TLI.xsd

Control XML log from command line

You may use the following command line switches to control XML log when running test suite from the command line:

- -t3rt “-xmllog” enables XML log.
- -t3rt "-xmlv <verbosity>" sets verbosity level for XML log mechanism
- -t3rt "-xmldebug" pretty prints XML data
- -t3rtxmlout "<filename>" sets output filename for XML data. Note that this switch is specific to default implementation of XML data handlers in the t3rts_conditional.c file

Note

Using the above mentioned runtime switches without having XML log mechanism registered results in the runtime errors during test suite initialization

Error Codes

Group Prefixes

The runtime system error codes can be grouped into the following:

Error Codes	Descriptions
GRL*	“General Error Codes” on page 200
FMT*	“Wide String Related Error Codes” on page 203
VAL*	“Values Related Error Codes” on page 204
TYP*	“Types Related Error Codes” on page 207
PRT*	“Port Operations Error Codes” on page 208
CMP*	“Component Operations Error Codes” on page 210
TMR*	“Timer Operations Error Codes” on page 215
ACL*	“Activation Lists Error Codes” on page 215
CNF*	“Runtime Configuration Error Codes” on page 216

General Error Codes

Error Code	Description
GRL0001	The value was not of the expected kind. The cause of the error is probably that a function was called with a value for which the function was not defined to handle.
GRL0002	The value was expected to be an instantiated value of some type. The cause of the error is probably that a function was called with a pseudo value that this function does not accept.
GRL0003	The value was expected to be properly initialized before the intended operation could be performed.

Error Code	Description
GRL0004	The named (pointer) parameter to a function was unexpectedly NULL.
GRL0005	An attempt to index into an entity resulted in an "out of bounds" error. The cause of the error is probably that an incorrect index is being used in a function call. The named entity, the actual index, and the lower and upper bound should be sufficient to localize the problem.
GRL0006	A type was used in an operation where another type was expected.
GRL0007	A type was used in an operation where that particular type was not allowed.
GRL0008	An operation failed because an unusable type structure was applied. It is the given (unusable) type that is displayed. The error can probably be found in the usage of the operation.
GRL0009	The named parameter to a function was unexpectedly negative where it is not permitted ('int2bit', for instance).
GRL0010	"Out of bounds" error. The cause of the error is probably an integer value that is out of range for a particular operation. The named entity, the actual value, and the lower and upper bound should be sufficient to localize the problem.
GRL0011	"Division by zero" error. The second operand to the `rem` operation was zero for which the operation is not defined. The actual call to '`rem`' can be the result of a '`mod`' operation since this is defined in terms of the '`rem`' operation.

Error Code	Description
GRL0012	General “division by zero” error. The execution of an integer division with zero (0) as the divisor was detected. Make proper adjustment to the test suite to eliminate this problem.
GRL0013	File could not be found. The named file could not be located when the RTS attempted to open it. It is probably a filename given as command-line argument. Check the file path and verify that the name is correct.
GRL0014	Object not found in symbol table. The named object (e.g. type instance) could not be located in the symbol table of a module. This error is generated by t3rt_find_element function. The propable cause for it is that some TTCN-3 modules in the project were not regenerated after TTCN-3 files changed.
GRL0015	Misspelling in the regexp pattern. This is a warning, not an error. It’s reported by runtime system when it parses regexp pattern and encounters propable syntax error (incorrectly specified metacharacter).
GRL0016	Compilation of pattern failed. This error is reported by runtime system when it fails to parse regexp pattern due to bad structure of a pattern.
GRL0017	Bad type reference. This error is reported when \N{typereference} metacharacter inside regexp pattern doesn’t represent valid type reference for specifying alphabet.

Wide String Related Error Codes

Error Code	Description
FMT001	The parameter number (after "%") in the formatting string is incorrect. The cause of the error is probably an attempt to use 0 as parameter number, or no number at all (no digits).
FMT002	The wide string unexpectedly ends after "%" and parameter number, where a valid parameter type character is expected. The cause of the error is probably an incorrect wide format string.
FMT003	The format type character is not recognized. The cause of the error is either that a type character is a non-ASCII character, or that the type character is not valid.
FMT004	The wide string unexpectedly ends after "%" character, where a valid parameter number or another "%" character is expected. The cause of the error is probably an incorrect wide format string.
FMT005	The format specifiers for the parameter are not consistent. This error will occur if the same parameter is referred in the wide format string with different type specifiers.
FMT006	One of the parameters numbers was skipped in the format wide string. If MAXNUM is the maximum parameter number used in the format wide string, then there should be also present all the parameters in the 1...MAXNUM range.

Values Related Error Codes

Error Code	Description
VAL0001	Values of this kind cannot be copied. The cause of the error is probably that a copy function was called with a value of port/component record or timer.
VAL0002	Value is read-only and cannot be changed. The cause of this error is an attempt to delete or assign to a value that was allocated using static allocation strategy.
VAL0003	Value failed consistency check against its own type. The cause of this error is probably an attempt to set or use a value that is not allowed by the type of this value.
VAL0004	Values of this kind cannot be assigned to (used as l-value). The cause of the error is probably that an assign function was called with a value of port/component record or timer.
VAL0005	Length, provided as parameter to one of the '`int2{bit, hex, oct}`' functions, is not sufficient for conversion.
VAL0006	String was unexpectedly empty. The cause of the error is probably an attempt to convert empty string into an integer or retrieve element by index from empty string.
VAL0007	String, provided as parameter to one of the '`{str,bit, hex, oct}2int`' functions, cannot be converted into an integer due to overflow.
VAL0008	Vector was unexpectedly empty. The cause of the error is probably an attempt to manipulate with an element of the empty vector.

Error Code	Description
VAL0009	Object identifier was unexpectedly empty. The cause of the error is probably an attempt to manipulate with an element of the empty object identifier.
VAL0010	The union value alternative was expected to be selected before the intended operation could be performed.
VAL0011	`Rvalue` cannot be assigned to the `lvalue` due to type check failure. The cause of this error is probably an attempt to assign values of incompatible types, or `rvalue` that violates type restrictions of the `lvalue` type.
VAL0012	Field of the record or set value cannot be marked as omitted since it is mandatory. The cause of this error is call to the 't3rt_value_set_omit' with a field that is not specified as optional.
VAL0013	The element value accessed is not used. Either implicitly by leaving out the value at initialization or by using the explicit '-' (`NotUsedSymbol`). Check the initialization of this element in the value or remove the access to it.
VAL0014	The attempted selection of union alternative is not a valid one according to the union type. Check that the alternative is in the type and change the usage to a valid one.
VAL0015	Value initialization failed during parsing (for example module parameter initialization). Check that the value definition is correct and that it is of correct type.

Error Code	Description
VAL0016	An unexpected character was encountered during value parsing (for example module parameter initialization). The string displayed shows the position where the parsing halted. Check that the value definition is correct and that it is of correct type.
VAL0017	An unexpected syntax was encountered during value parsing. The string displayed shows the position where the parsing halted. Check that the value definition is correct and that it is of correct type.
VAL0018	An attempt to set a non-zero sized array to empty was detected. This is not allowed.
VAL0019	When converting an octet string to a character string, a symbol outside the convertible range was encountered.
VAL0020	Casting of a value to another type failed.
VAL0021	Failed to pass omit as a pass-by-value parameter to a function.
VAL0022	Failed to pass static template by reference (as inout parameter) to a function.
VAL0023	‘valueof’ operation failed because template contained matching symbols (e.g. ‘*’ or ‘?’).
VAL0024	Operation over template variables could not be performed for template variables of this type. This error is reported for example when trying to dereference string element in a string template variable using subscription operation (‘str[i]’).
VAL0025	Operation over template variables could not be performed for template variables with such structure. This error is reported for example when trying to dereference element in a record template variable, to which ‘*’ is assigned.
VAL0026	Provided values have to be of the same type although they were not.

Error Code	Description
VAL0027	When converting character string to an octet string, a symbol that doesn't represent valid octet character (A-F, 0-9) was encountered.
VAL0028	When processing regexp pattern runtime system failed to locate value with the name specified in value reference metacharatcer \{value}.
VAL0029	String value specified in \N{value} alphabet metacharacter should have length of 1 element.
VAL0030	char2int and unichar2int functions accept string only with one element length.

Types Related Error Codes

Error Code	Description
TYP0001	Type has no fields. The cause of the error is an attempt to access a field of the type that does not have any.
TYP0002	No applicable codecs found when encoding or decoding a value of the reported type. The cause of the error is that no codec function was registered for this type and the built-in codecs was disabled.
TYP0003	sizeoftype function failed because provided type doesn't correspond to length restricted subtype or upper length boudary has infinite limit.

Port Operations Error Codes

Error Code	Description
PRT0001	Port without valid assigned address detected. The cause of the error is probably that the port value was not correctly initialized with the appropriate address.
PRT0002	Found multiple handled calls of same type from same client. The cause of the error is probably an error in the test suite.
PRT0003	Ports are incompatible. The cause of the error is an attempt to connect or map ports that are not compatible. All `out/inout` types of one port must be acceptable as `in/inout` types for another one.
PRT0004	Port is inactive. Operation on unmapped/unconnected port attempted.
PRT0005	Value can not be sent or received on the port. The type of a value being communicated could not be found in the list of allowed message declared in the port type.
PRT0006	Communication was attempted on a port that was not mapped/connected. The only situation where ports are mapped implicitly are when a system component type is specified for a test case. In all other cases, map operations have to be done explicitly.
PRT0007	Communication is attempted on a connected port that has more than one destination connected. A component destination must be stated explicitly on the communication operation.

Error Code	Description
PRT0008	Address unavailable. During the process of extracting the sender address of an operation by using the “sender redirect” syntax (`-> sender addr_var`) the address was not available in the port input queue. The source of the problem is in the integration (for example the SA implementation in the case of TRI).
PRT0009	Wrong type of destination reference. When stating destination with the to directive, the destination must be a component reference for a connected port and an address for a mapped port. The analyzer should prevent this from happening but in some cases it can not and improper usage is only detectable at runtime.
PRT0010	An address value had, unexpectedly, the value null. This can occur if the to address argument for a communication operation has the value null.

Component Operations Error Codes

Error Code	Description
CMP0001	Component without valid assigned address detected. The cause of the error is probably that a PTC was created without a valid control port address.
CMP0002	Component terminated with running timers. The cause of the error is probably that a component was terminated (abruptly or not) and the test suite did not wait for or cancel the relevant timer.
CMP0003	Component terminated with unconsumed messages. The cause of the error is probably that a component was terminated (abruptly or not) and the test suite is written such that it did not receive these messages before termination.
CMP0004	May not connect a system port. Use map instead. The cause of the error is that one of the components specified to a connect operation is identical to the system component. This is not allowed. Use the map operation for this purpose.
CMP0005	Problem with operation detected at this position. This is a generic error message indicating that some error condition was detected. Look elsewhere in the information log for additional information on what the actual problem is. This message is generated primarily to give the correct source location for the other error message.
CMP0006	Attempt to map a port more than once. The present implementation of TTCN-3 limits port mappings to at most one mapping per port.

Error Code	Description
CMP0007	Attempt to map a port that is already connected. It is not allowed to have a port that is simultaneously connected and mapped according to ETSI ES 201 873-1 V2.2.1.
CMP0008	Attempt to map without specifying a system port. It is not allowed to map without specifying one system port according to ETSI ES 201 873-1 V2.2.1.
CMP0009	Disconnect of unconnected port attempted. There is no use in disconnecting a port when it is not yet connected.
CMP0010	Unmap of unmapped port attempted. There is no use in unmapping a port when it is not yet mapped.
CMP0011	Disconnect of a mapped port attempted. The cause of the error is that one of the components specified to a disconnect operation is identical to the system component. This is not allowed. Use the `unmap` operation for this purpose.
CMP0012	Unmap of a connected port attempted. The cause of the error is that none of the components specified to an `unmap` operation is identical to the system component. This is not allowed. Use the disconnect operation for this purpose.
CMP0013	Start on component not in newly initialized state The cause of the error is that the components specified to a start operation is either already running or has been running. This is not allowed.

Error Code	Description
CMP0014	Map to multiple system ports not allowed The cause of the error is that the map operation is applied on an already mapped port. This is not allowed.
CMP0015	Connect to multiple ports in the same component not allowed The cause of the error is that the connect operation is applied on a port that is already connected to a port on the same component. This is not allowed (section 8.2 in ETSI ES 201 873-1 V2.2.1).
CMP0016	Mapping two system ports are not allowed The cause of the error is that the map operation is applied to two system ports. This is not allowed (section 8.2 in ETSI ES 201 873-1 V2.2.1).
CMP0017	The execution of the test case took longer than the specified time. The cause of the error is either that the execution ended up in a dead-lock or the execution just has not finished.
CMP0018	Unexpected null component reference. The cause of the error is that one of the component operations is performed on a null component reference, where it is not applicable.
CMP0019	Unexpected null component reference constant. The cause of the error is that one of the component operations is performed on a null component reference symbol, where it is not applicable.
CMP0020	Attempt to connect an already mapped port. ETSI ES 201 873-1 V2.2.1 prohibits a port from being both mapped to a system port and connected to a component port at the same time.

Error Code	Description
CMP0021	Operation not allowed to be applied to all/any except in the MTC. ETSI ES 201 873-1 V2.2.1 prohibits applying operations to all/any components except from within the MTC.
CMP0022	Waiting for acknowledgement timed out. In the internal communication of the running components, an acknowledgement response took too long. Either this is an effect of slow execution in parts of the code or a hanging component did not respond. Try to increase this value substantially to see if the problem disappears. ("t3rt.control.ack_timeout" on page 128 in Chapter 6, *Execute Tests*)
CMP0023	Attempt to map a port to an occupied system port. It is not allowed to map two ports of the same component to the same system port.
CMP0024	Attempt to execute a test case that cannot be found.
CMP0025	Operation not allowed to be applied except in the control part. The TTCN-3 standard, ETSI ES 201 873-1 V2.2.1, prohibits applying certain operations except from within the control part of the test suite.
CMP0026	Deadlock in `alt` construct detected. The TTCN-3 standard, ETSI ES 201 873-1 V2.2.1, explicitly require testing for situations where an `alt` would be unable to proceed due to all ports being filled with non-matching data (messages, procedures or exceptions).
CMP0027	Detected a spurious timeout that was discarded. The cause of this error is probably that a timer from a previous test case delivered a timeout "too late" so that the following test case had already started when it was delivered.

Error Code	Description
CMP0028	Detected a lookup of a named object that is not available in the current component. The cause of this error is probably that the actual component used in the TTCN-3 code is of an invalid type.
CMP0029	Detected a behavior that is unsupported in the current situation. The cause of this error is probably that the behavior specified for a component type initialization includes an operation that is unsupported at that time.The solution is to move the initialization statement in question to early in the testcase/function-that contain the behavior associated with the component type.
CMP0030	Premature component termination detected. The cause of this warning is probably that the behavior specified for the test case do not take into account the possibilityfor components to run completely independently which in turn may cause other components to not be finished when the MTC terminates and hence terminates the test case. The solution to this is most likely to insert a "all component.done" statement near the end of the test case.
CMP0031	Map to unexistent port. The cause of this error is the attempt to map a component's port to a non-existent TSI port.
CMP0032	Unmap from unexistent port. The cause of this error is the attempt to unmap a component's port from a non-existent TSI port.
CMP0033	Creation of component for test case without TSI. The cause of this error is the attempt to create parallel component for a test case that doesn’t explicitly declare system (TSI) component. Such test cases may have only MTC component.

Timer Operations Error Codes

Error Code	Description
TMR0001	Timer with invalid duration The cause of the error is probably that a timer was either specified with a negative default duration or started with a negative duration.
TMR0002	Timer has no defined duration. An operation was made where a duration was needed for a timer. This happens, for example, when a 'start' operation is made with no explicit duration and no default duration was stated for the timer declaration.

Activation Lists Error Codes

Error Code	Description
ACL0001	Invalid default reference. The cause of the error is an attempt to use deactivate statement with an invalid default reference (that is, already deactivated, for instance).
ACL0002	Test step is not found. The cause of the error is an attempt to use activate statement with a combination of modulename/teststepname that cannot be found.
ACL0003	Local timer passed to activated altstep. The cause of the error is an attempt to use activate statement with an altstep passing to it local timer (only component timers are allowed).

Runtime Configuration Error Codes

Error Code	Description
CNF0001	Unknown switch. `RTconf` encountered unknown switch that will be discarded.
CNF0002	During parsing of a configuration file, no proper type could be found for the given key. Either the type was not stated or it was misspelled. The key is ignored.
CNF0003	During parsing of a configuration file, the value of the key could not be parsed in the context of the stated type. Check that the type and the value correspond. The key is ignored.
CNF0004	Syntax error found when parsing the command-line switches. The string found does not necessarily point to the exact position of the error, it is very likely that the error occurred earlier.
CNF0005	An error occurred when setting a value to a configuration key of another type than the expected. For example: `'-confbool this.confkey 10'` The problem may also occur if a switch is supplied with the wrong type of argument, for example '-to true' which would set the overall test case timeout value to true when it is supposed to be a float value.
CNF0006	A module parameter given on command-line was not a valid module parameter. Module parameters must start with an alpha symbol, contain only alphanumeric symbols and have a maximum of one dot symbol. If a dot symbol is found, the following identifier must start with an alpha symbol.

Debugging Test Suite

IBM Rational Systems Tester contains two debuggers: real-time and post-mortem.

Real-time debugger may be used to debug ETS while test suite is executing. That is you use real-time debugger to control and analyze test suite execution.

Post-mortem debugging means that the debugger gets all the input after ETS finished its execution, to achieve this the debugger works with a trace (log) output of the ETS.

TTCN-3 Real-Timer Debugger

TTCN-3 Real-Time debugger may be used to control and investigate test suite execution. Debugger may be used with any form of test management including GUI test management. In order to perform real-time debugging test suite has to be compiled and built with debugger support enabled.

Working with the debugger

General debugger workflow includes building project for debug, starting the debugger, setting necessary breakpoints, executing test suite up to breakpoint, tracing certain parts of test suite, investigating state of variables and possible runtime errors.

Building project for debugging

In order to debug test suite you may have to rebuild project with debugger support turned on. To do it go to project setting, open “Build” tab and select “Build for debug” check box. Then recompile and rebuild project.

Note

You may normally execute your test suite while debugger support is enabled without any degrade of performance.

Existing projects require manual patching of project configuration file (mcfg file) and probably t3rts_conditional.c file if you are not using this file from the Rational Systems Tester installation.

Locate make_win.mcfg file (or other according to platform used) in the installation directory and copy line started with “DEBUGGER_LIBS” into your project mcfg file.

If you use custom makefile then you have consider following requirements. Project has to be compiled with “-G” option and libt3dbg library (and several system libraries) has to be linked to the test suite executable. The value of DEBUGGER_LIBS variable denotes the list of libraries that have to be linked.

If you use custom t3rts_conditional.c file (not from Rational Systems Tester installation) please consider that real-time debugger has to be registered in the runtime system. Usually (at least for new projects) this is done by in-

voking `t3rt_register_debugger` function. Ensure that this function is invoked from the `t3rt_register_provided_logging` function, which is defined in t3rts_conditional.c file.

Starting and stopping debugger

Real-time debugger session is started by pressing “Real-time debugger” button on the tool bar or choosing menu item with the same name from the “Project” menu. You will receive error message if debugger for some reason cannot start (see Building project for debugging for notes on enabling debugger support in existing projects).

Depending on selected test management you may observe two situations. If GUI test management is used then Rational Systems Tester fills “Execution” tab in the workspace window with available test cases and awaits your command. You may set necessary breakpoints and start execution or tracing of a test case or control part.

If you are using other forms of test management then debugger breaks execution at the beginning of control part providing you with the ability to set necessary breakpoints and choose whether to continue execution or start tracing.

Debugger may be stopped by pressing “Stop debugger” or “Stop action” buttons or choosing “Stop action” menu item from “Project” menu. Debugger also stops when test suite executable terminates.

All user-defined breakpoints are permanently saved when debugger stops. They will be restored at next debugger session.

Setting and removing breakpoints

You may set breakpoint by pressing “Set/Remove breakpoint” button or choosing menu item with the same name from “Debugger” menu. You may also perform this action by right clicking on certain line and choosing corresponding item from the popup window.

When breakpoint is set red bullet appears on the left border of the window.

Breakpoint may be removed by pressing the same button. Red bullet disappears when breakpoint is removed.

Breakpoints may be temporarily disabled. Read Administering breakpoints section on how to disable and enable breakpoints.

Administering breakpoints

Breakpoint administration is performed using the special window that may be opened by choosing “Edit breakpoints” item from “Edit” menu. Breakpoint administration window contains list of currently set breakpoints. Each line contains a check that denotes whether breakpoint is enabled or disabled and a position of the breakpoint.

You may remove check thus disabling the breakpoint. Read bullet at the breakpoints line becomes grey in this case. Debugger doesn’t break execution at disabled breakpoints.

Double clicking on any line in the breakpoint window locates the breakpoint in the text editor.

Pressing “del” button in the breakpoint window removes selected breakpoint.

Breaking execution

You may break execution of the test suite at any moment by pressing “Break” button or choosing “Break” menu item from the debugger menu. Debugger pauses all executing components and locates source position of MTC component in the text editor. Current position is highlighted with yellow arrow at the left border of the window.

Debugger also highlights the event description in the execution log for the last event of MTC component.

Press “Go” button or choose “Go” menu item from the debugger menu to continue execution.

Tracing execution

Debugger support three tracing operations:

- Step Into. This operation continues execution up to next statement and breaks execution upon reaching it. This operation steps into user defined functions, `altsteps` and test cases.
- Step Over. Similar to Step Into but doesn’t step into user defined functions, `altsteps` and test cases.
- Step Out. This operations continues execution up to the end of current scope (function, `altstep`, test case or control part).

You may invoke any of these tracing operations by pressing corresponding buttons or choosing corresponding menu items from the “Debugger” menu.

Note

When debugger steps into scope (e.g. function) it always make a step on the scope header (e.g. function name). When debugger leaves the scope it always make a step on the closing “}” of the scope body.

Step Into command when invoked on “execute” or “start” TTCN-3 operation steps into test case or a function started on parallel component.

If you are using GUI test management when you may start tracing of a test case or control part by right clicking on the certain item of the “Execution” tab in the workspace window and choosing “Trace Test Case” or “Trace Control Part”. You may also use toolbar or “Project” menu for this action.

Tracing TTCN-3 “alt” statements

You may perform tracing of “alt” blocks. In this case Step Into and Step Over actions have specific semantics. Pressing Step Over on an “alt” block header or an alternative continues execution until one of the alternatives matches and execution flow enters body of the alternative. Debugger breaks at the beginning of the alternative body. If alternative doesn’t have body then debugger breaks at first statement after “alt” block.

Pressing StepIn on an “alt” block header or an alternative allows to step through matching of all alternatives as well as guard expression. In this case debugger breaks on every alternative and steps into defaults and any functions that may be used in guard expressions. You may set breakpoints in the activated `altsteps` and in functions that are used in guard expressions. Debugger breaks test suite execution when it reaches these breakpoints.

Note

When execution breaks in a function invoked from guard expression always use StepIn to trace function body.

Handling runtime errors

Debugger traps runtime errors and notifies user about them. When runtime error occurs debugger breaks execution at error position and shows dialog window with error information. Information includes source position, component and scope name and error description. Pressing “Ok” button in this window locates error position in the text editor.

If you choose to continue execution then runtime system correctly shutdowns test case execution.

Inspecting Call Stack

When debugger breaks execution (e.g. execution reaches breakpoint) you may inspect (trace) call stack of currently selected component. Open “Debugger View” tab in the workspace window. You will notice “CallStack“ node in the displayed tree. The name in brackets denotes the name of the current component. Child nodes represent stack frames of the component. Top child node denotes topmost stack frame.

Double click on an arbitrary entry. Debugger locates the source position in the text editor and updates watches and execution log. Watched variables receive values that they have inside the selected stack frame. Variables that are not visible there are shown as undefined. In execution log debugger locates and selects event description that was generated for the current stack statement (usually function call).

Observe that position for inner stack frames is marked with green triangle while position of topmost frame is marked with yellow arrow. Whatever frame is selected you may always invoke “Show next statement“ debugger command to locate the current position in the topmost stack frame.

Debugging parallel components

IBM Rational Systems Tester real-time debugger supports debugging of test suites with multiple parallel test components. While debugging such test suite you need to keep in mind several specifics.

Debugger has the notion of “active” component. Only one component may be active at the same moment. Active component differs from other components in two things:

- Position in the text editor (with yellow arrow) is shown for active component.
- Tracing operations are performed for an active component.

When debugger breaks execution you may manually select arbitrary component as active. Open “Debugger View” tab in the workspace window. You will notice “Components“ node in the displayed tree. Child nodes represent all components executing in the test suite. This may include control component (CPC) if test suite execution started from the control part.

Double click on arbitrary component to set is as active. Debugger updates watches, execution log, call stack and according the selected component and locates its current position in the text editor. Subsequent tracing operations will be related to this component.

You may always refer to the “CallStack” node in the “Debugger View“ to determine which component is active. This name of the active component is displayed in brackets, e.g. “CallStack (MyPTC)“.

Note

When one of the components breaks execution (e.g. reaches breakpoint) debugger stops all other components. No components are executing in the background.

Semantics of becoming “active” for a component are following:

- Initially component executing control part (CPC component) is active if control part is started to execute or MTC – if test case is started to execute directly using GUI test management.
- Invoking “Step Into” operation on “execute” statement makes MTC active.
- Invoking “Step Into” operation on “start” statement makes corresponding PTC active.
- When user breaks execution with “Break” command MTC becomes active (or CPC if no test case is running).
- When component reaches breakpoint it becomes active.
- When active PTC terminates MTC becomes active
- When MTC terminates CPC becomes active.
- When runtime error occurs in the component it becomes active.

Example 23(Example debugging scenario) ––––––––––––––––––––––––

User starts test case from the “Execution“ tab thus MTC is active. At some moment in time PTC:5 reaches breakpoint and becomes active. Debugger shows source position for this component and highlights last event of PTC:5 in the execution trace window. Note that this may not be last entry in the trace window. PTC:5 is traced with StepInto/StepOver/StepOut commands.

While tracing operation is performed PTC:1 reaches breakpoint thus PTC:1 becomes active and debugger forgets about PTC:5. Debugger pauses all executing components and locates source position of the PTC:1.

PTC:1 is traced up to its end. When PTC:1 terminates debugger makes MTC active, pauses all components and locates MTC source position in the text editor.

Watching variables

IBM Rational Systems Tester real-time debugger provides the ability to watch the values of variables and simple expressions. Values of structured variables (records, unions, arrays) are displayed in tree-like view allowing to expand and collapse subfields thus providing convenient way for inspecting test suite execution state.

Debugger supports watching of almost every object in the test suite. It's possible to watch values of constants, variables, templates, component and default references, the state of ports and timers.

Expressions that may be watched are restricted to record field names (e.g. recordVar.field1.subField2), array elements (e.g. arrayVar[7]) and its mixes (e.g. arrayVar[7].field1[8].subField2). It's allowed to use references to other variables (e.g. loop counter) to select certain array element (e.g. arrayVar[i]). Variable names may be given in the qualified form, i.e. may be prefixed with module names (e.g. MyModule.myConst). It's not allowed to use calls to functions (including TTCN-3 predefined functions).

Note

Variables may be watched only when test suite execution is paused (broken).

Quick inspection of variables

You may quickly inspect the value of a variable or expression by hovering mouse over it in the text editor. Value is displayed in the tooltip window in the "pretty-printed" form. Curly braces are not displayed in order to minimize the size of the tooltip window. This should not harm the presentation of the value since every field is printed on separate line.

Debugger tries to guess the expression which has to be watched. It means that if mouse is hovered over a variable name then the value of the whole variable is displayed. And if cursor is hovered over a field name (i.e. over a "myField" part in "recordVar.myField" string) then only the value of the field under cursor is displayed.

If tooltip window is not shown then it means that either name under cursor doesn't denote valid expression or debugger failed to guess right the inspected expression. The later situation may occur when expression contain spaces, e.g. "arrayVar [7]". In this case you may select the expression that you are interested in and hover mouse over selection, the value of the selected expression will be displayed in tooltip.

Adding variables to the Watch window

There exist two ways of adding variables to the watch window: using Debug menu and using context menu in the text editor.

To add variable through menu choose "Add Watch" item in the "Debug" menu, type watched expression in the opened window and press "Ok" button.

To add variable through text editor context menu right click on the variable or field name and choose "Add Watch" in the popup window. Accept or change suggested watch expression and press "Ok". If you chose "Add Watch" after right clicking on the selection then this selection will be used as an expression to be watched.

Names of all watched expressions are displayed in the "Watches" folder on the "Debugger View" in the workspace window. Only unique names are displayed. If you added one and the same variable for several times then "Watches" folder will contain exactly one element for all of these entries.

Note

Watches are saved between debugger sessions even if you closed project and restarted Rational Systems Tester.

Removing variables from Watch window

There are several ways of removing watched expressions from watch window. You may right click on a watched value and choose "UnWatch" or press "del" button on it.

In order to delete all instances of this or that watched expression locate it in "Watches" folder on the "Debugger View", right click mouse on it and choose "UnWatchAll". Debugger will remove instances of this expression from all watched windows and "Watches" folder as well.

Working with Watch windows

You may open arbitrary number of watch windows in the debugger. Choose “Open new watch window“ in the “Debug“ menu to open new window. Each time when you add new watch it’s put into the currently selected watch window. If no watch windows opened then debugger opens new window.

You may duplicate watch by drag-and-dropping watched value between watch windows. Right clicking on a watched value and choosing “Watch“ creates a duplicate entry in the same watch window.

You may quickly create new watch for a record field. Right click on the node representing the field value in a compound value and choose “Watch“. Debugger will create new watch instance for a selected field.

Note

Closing watch window doesn’t remove watches from it. You have to manually remove them by choosing “UnWatchAll“ on the certain entry in the “Watches” folder on the “Debugger View“ tab.

Printing watched expression value

You may display the value of a watched expression or any of its fields in the “Output” window. Value is displayed according to TTCN-3 syntax rules so you may copy-paste outputed string into TTCN-3 text editor and use it in test specification.

To print value of watched expression right click on a watched value and choose “Display“. Value of the selected object will be displayed in the “Debugger“ of the output window.

Note

Values of objects that do not have standard representation (e.g. timers and ports) and undefined variables are displayed in angle brackets.

Debugger commands

Following debugger commands are supported in real-time debugger:

- Restart
- Stop
- Go
- Insert/Remove breakpoint

- Step into
- Step over
- Step out
- Show next statement

Restart

This commands restarts debugged test suite.

If you need to start system under test prior to starting test suite (e.g. if connection between ETS and SUT is established in triSAReset function) then this command may not function properly. This is because debugger doesn't restart SUT and it may not execute when triSAReset is called. In such case first stop debugger, then restart SUT and finally start debugger.

Key accelerator is CTRL + SHIFT + F5.

Step Into

This command continues test suite execution up to the next statement of "active" component (see Administering breakpoints about "active" components). This command steps into user-defined functions, `altsteps` and test cases. StepInto may be used to trace alternatives, guard expressions and defaults in "alt" blocks.

Note

If another component reaches breakpoint then StepInto command will not be completed.

Key accelerator is F11.

Step Over

This command continues test suite execution up to the next statement of "active" component (see Administering breakpoints about "active" components). This command doesn't steps into user-defined functions, `altsteps` and test cases. StepOver may not be used to trace "alt" blocks.

Note

If another component reaches breakpoint then StepOver command will not be completed.

Key accelerator is F10.

Step Out

When this command is executed debugger stops at the source position immediately outside the current scope (i.e. function, test case, `altstep` or control part). StepOut may not be used to trace “alt“ blocks.

Note

If another component reaches breakpoint then StepOut command will not be completed.

Key accelerator is SHIFT + F11.

Show next statement

This command is used to locate current source position of “active“ component. Editor window with current source line is brought to the front, and centered on this line.

Key accelerator is ALT + * (‘*’ on the numeric keypad).

Breakpoints

Breakpoints can be used to break test suite execution at certain position.

Breakpoints in Rational Systems Tester work the same way as breakpoints in Microsoft Visual Studio with respect to threads: as soon as breakpoint (source line) is encountered in any of the running components, execution breaks and text editor window showing reached source position opens.

Source lines with breakpoints are marked specially in the editor windows.

Insert/Remove breakpoint

This operation sets a breakpoint on the current source line if there was no breakpoint; it removes existing breakpoint if there is one.

Key accelerator is F9.

Go

This command is used together with breakpoints; it resumes test suite execution until one of the components reaches breakpoint or runtime error is generated or test suite terminates.

TTCN-3 Post-Mortem Debugger

Post-mortem debugging means that the debugger gets all the input after ETS finished its execution, to achieve this the debugger works with a trace (log) output of the ETS.

Post-Mortem debugger may be useful in debugging time-critical test suites where stopping at breakpoint while ETS is executing may impact the overall test case verdict.

General workflow

In order for the post-mortem debugger to work (and its commands activated) it is necessary to perform the following steps.

- Enable the Post-mortem debugger.
  From the Tools menu select the Customize command. Go to the Add-Ins tab and select the **TTCN3PostMortemDebugger**.
- Run ETS, so Execution output tab contains a log of the ETS execution. Save the log (by default this will be in a file with extension `.spm`). Another option is to reuse any previous log files, saved from a previous run. To do this you can save the log from the output tab into a text file after ETS has finished. It is also possible to set this up for your project, from the **Project** menu choose **Settings** and then go to the **Execution** tab. Check the **Log also to file** option. Enter the filename of your choice.
- Provide the name of the log file. From the **Project** menu choose **Settings** and then go to the **Misc** tab. Browse to the file in the **Post-mortem log file** field.
- Run the Post-mortem debugger option in the **Project** menu.

Debugger commands

Functionality for reconstruction of ETS control flow is similar to functionality provided by Microsoft Visual Studio debugger. MSVS debugger is real-time, so all features do not make sense for the post-mortem debugger.

Following debugger commands are supported in Post-mortem debugger:

- Restart
- Stop
- Go

- Insert/Remove breakpoint
- Step into
- Step over
- Step out
- Show next statement

All these commands are described in detail below.

Restart

This command allows to revert debugger state to the one corresponding to the beginning of ETS execution, so traversing can be started once again.

This operation is performed automatically after ETS is executed once again, so there is no need to perform any specific “start traversing” operation.

Key accelerator is CTRL + SHIFT + F5.

Step Into

This command selects and focuses on source line that has been executed after the one that is currently selected. For calls to named scopes (like functions, test cases, `altstep` and control part), next statement inside the scope is selected.

Key accelerator is F11.

Step Over

This command selects and focuses on source line that has been executed after the one that is currently selected. This is very similar to Step Into operation; the difference is in how it treats named scopes (like functions, test cases, `altstep` and control part). These scopes are considered as single statements by this operation. So, if, for example, currently selected statement is a function call, then Step Over will select the statement after the function call, not the first statement inside the function body, as it will be for Step Into.

Key accelerator is F10.

Step Out

When this command is executed, debugger selects the source line immediately outside the current named scope (like function, test case, `altstep` or control part).

Key accelerator is SHIFT + F11.

Show next statement

This command is used to focus on the next statement to be executed (as seen in the execution log). Editor window with current source line is brought to the front, and centered on this line.

Key accelerator is ALT + * (‘*’ on the numeric keypad).

Breakpoints

Breakpoints can be used with post-mortem debugger to simplify traversing. They work similar to breakpoints in any other real-time debugger. In the post-mortem debugger they are used primarily to skip directly to the interesting parts of execution traces.

Source lines with breakpoints are marked specially in the editor windows.

When it comes to concurrent execution of TTCN-3 components, breakpoints in Rational Systems Tester work the same way as breakpoints in Microsoft Visual Studio with respect to threads: as soon as breakpoint (source line) is encountered in any of the running components, traversing terminates and control returns to the user.

Insert/Remove breakpoint

This operation sets a breakpoint on the current source line if there was no breakpoint; it removes existing breakpoint if there is one.

Key accelerator is F9.

Go

This command is used together with breakpoints; it means “traverse execution log until the end, or until a source line with breakpoint is encountered; then stop”. The line with a breakpoint becomes current line, and it is possible

to use other commands (like Step Into) starting from this position. If no breakpoints are encountered, debugger reports that end of execution log is reached, and automatically performs “Restart traversing” command.

9

Using the Script Wizard

IBM Rational Systems Tester includes a Script Wizard, enabling you to navigate in a UML representation of a TTCN-3 project – presented as HTML files – and to create Tcl scripts to extract information from this model.

With the Script Wizard, you can filter information in your projects and only display the parts that are essential to you.

You can store the UML navigation history, and the resulting model query in a Tcl script by clicking Create script code, always displayed at the top of the HTML page. When your script is done, just click on the Execute script button in the toolbar. This creates a Tcl interpreter that executes the script and subsequently displays HTML pages or information in an output tab.

Hint

By clicking the + or - signs in the HTML page, different UML elements, that is, Associations, Classes and so on, are collapsed or expanded.

You reach the Script Wizard through the Script Wizard button in the toolbar, or by selecting Script Wizard in the Tools menu.

Note

The interpreter is different for each execution of the script, and that you cannot share a variable among different executions.

About Tcl

Tcl (Tool Command Language) has a simple and programmable syntax and can be either used as a standalone application or embedded in application programs.

Tcl Semantics

All Tcl command names begin with a capital letter, followed by lower-case letters, unless it is a new word:

```
command -> Command
addmenu -> AddMenu
```

All parameters are written with small letters, without any '-' or '_'. With one exception, some parameters can begin with a '-':

```
Selection add <object>, ReportInit <tabname> [-check]
```

Tcl Commands

Tclscripts are made up of commands separated by newline characters or semicolons. All commands have the same basic form, but for most commands the word structure is important, with each word used for a distinct purpose.

All Tcl commands return results. If a command has no meaningful result, an empty string will be returned.

Tcl API Commands

MapRole

```
MapRole <sourceobject> <rolename> [-filter <filterscript>]
[<script>]
```

GetRole

```
GetRole <sourceobject> <rolename>
```

Class

```
Class <object>
```

Get
```
Get <sourceobject> <attributename>
```

RolesOf
```
RolesOf [<sourceobject>|<classname>]
```

AttributesOf
```
AttributesOf [<sourceobject>|<classname>]
```

Classes
```
Classes
```

Call
```
Call <sourceobject> <methodname> [<args>]
```

Create
```
Create <mainobject> <classname>
```

SetAtt
```
SetAtt <object> <attribute> <value>
```

SetRole
```
SetRole <sourceobject> <rolename> <destobject>
```

AddRole
```
AddRole <sourceobject> <rolename> <destobject>
```

Output Commands

Output
```
Output <text>
```

This command allows Tcl script to print a message in the Script output tab.

OutputLog
```
OutputLog [on <tabname> <pathname> <separator>|off]
```

This command starts/ends the log of the Output window, which means that the content of tabs are flushed in files.

<pathname> must contain the path of the directory when the different log files will be created (file name are the name of the tabs).

<separator> is the character (or string) used to separate the column values of a same line.

Report Commands

ReportInit

```
ReportInit <tabname> [-check] [-cb <filename>]
[<columnlabel> <columnwidth> <columnalignment>]+
```

This command is used to create a new report tab in the Output window.

If the -check is used, a check box is displayed before each result line. These check boxes allow a more powerful navigation: only the checked lines will be selected when you navigate by using the F4 key, others are ignored.

If the -cb <filename> is present, then the associated Tcl script file will be evaluated when you double-click on a line object, and the OnDoubleClick procedure of this Tcl script will be called with double-clicked object as parameter.

For each <columnlabel> <columnwidth> <columnalignment> item, a new column is created with the corresponding label, width and alignment. An alignment of 0 means left align and an alignment of 1 means right. For sorting, columns right aligned are considered as number and columns left aligned are considered as text.

Report

```
Report <sourceobject> [<label_string>]*
```

This command is used for adding new lines in a report tab.

The <sourceobject> will be used to fill the first column (icon + text) automatically.

The <label_string> values are labels, used to fill the remaining columns.

BrowserReportInit

```
BrowserReportInit <tabname> [<iconfile>] [-keep] [-cb
<filename>]
```

This command allows you to activate a tree tab inside the Browser view. If needed, the tab is created.

If <iconfile> is present, then the icon will be used as the tab image.

If -keep is present then the content of the tab (if any) is kept, otherwise the tab is re-set.

If the -cb <filename> is present, then the associated Tcl script file will be evaluated when you double-click on a tree object, and the OnDoubleClick procedure of this Tcl script will be called with the double-clicked object as parameter.

BrowserReport

```
BrowserReport [-expanded] [-userdata <userdata>]
<childobject> [<parentobject>]
```

This command is used to add new object called <childobject> in the tree.

If -expanded is present and if the object has children, then the node will be expanded, otherwise it will be collapsed.

If <parentobject> is present the node is created as a child of the node corresponding to <parentobject>.

If -userdata is present, then <userdata> will be associated to this node on the tree.

HtmlReport

```
HtmlReport [-delete] [-use <ident>] [<filename>]+
```

Command to load an HTML file inside an HTML view.

The first <filename> is the URL to load.

Any following <filename> will be file(s) to delete when not used anymore, if -delete is present.

If the -use option is used, then the window with the title <ident> must be used again if it already exists.

TextReport

```
TextReport <filename> [<line>] [<column>]
```

This command opens the file <filename>, and locates (puts the position of the cursor) to a <line> if specified, and to a <column> if specified. You may specify a line without specifying any column.

Activate Commands

Activate

```
activate <object>
```

This command simulates a double-click on the occurrence of <object>.

ActivateBrowser

```
ActivateBrowser [<tabname>]
```

The browser view becomes active. If a `<tabname>` is present, then this command activates this tab.

ActivateProject

```
ActivateProject <projectname>
```

This command changes the active project into the workspace.

ActivateConf

```
ActivateConf <confname>
```

This command changes the current configuration of the active project into the workspace.

ActivateTool

```
ActivateTool <toolname>
```

This command changes the current tool.

OpenDocument

```
OpenDocument <filename>
```

This command opens the file <filename>.

This command is related to the SaveDocument command.

Important!

Be careful, this command applies to the active view (call Activate or Activate-Browser before calling it)

Command activate

```
Command activate <extensionident> <commandid>
```

This command simulates the call of the command, as if you click on the menu or the toolbar.

`commandid` is a hexadecimal number

Command get

```
Command get [state|check|menu|toolbar] <extensionident>
<commandid>
```

This command returns a value indicating the status of the command.

If state is used, it returns if the command is greyed or not (possible return values are Unknown, Disabled and Normal).

If check is used, it returns if the command is checked or not (possible return values are Unknown, Disabled or Normal).

If toolbar is used, it returns if this command can be found in a toolbar (possible re-turn values are Disabled or Normal).

If menu is used, it returns if this command can be found in a menu (possible return values are Disabled or Normal).

Selection Commands

The selection commands modify a special selection only known by the Tcl script. If the Tcl command Command activate is called, and the Tcl selection is not empty, the command takes into account the Tcl selection. If the Tcl selection is empty, it takes into account the selection in Rational Systems Tester.

Selection add

```
Selection add <object>
```

Add an object to the Tcl selection.

Selection remove

```
Selection remove <object>
```

Remove an object from the Tcl selection

Selection set

```
selection set <object>
```

Empty the Tcl selection, then add the object to it.

Selection reset

```
selection reset
```

Empty the Tcl selection

Commands from the Edit Menu

Undo

```
Undo
```

Call Undo on the Edit menu.

Redo

```
Redo
```

Call Redo on the Edit menu.

Cut

```
Cut <object>
```

Call Cut on the Edit menu, after having selected <object>. The selection then returns to its previous value.

Copy

```
Copy <object>
```

Call Copy on the Edit menu, after having selected <object>. The selection then returns to its previous value.

Paste

```
Paste <objectcontainer>
```

Call Paste on the Edit menu, after having selected `<objectcontainer>`. The selection then returns to its previous value.

Delete

```
Delete <object>
```

Call Delete on the Edit menu, after having selected <object>. The selection then returns to its previous value.

CanCut

```
CanCut <object>
```

Evaluates if Cut on the Edit menu would be available with <object> selected.

Possible return values are Unknown, Disabled and Normal.

CanCopy

```
CanCopy <object>
```

Evaluates if Copy on the Edit menu would be available with <object> selected.

Possible return values are Unknown, Disabled and Normal.

CanPaste

```
CanPaste < objectcontainer>
```

Evaluates if Paste on the Edit menu would be available with <object> selected.

Possible return values are Unknown, Disabled and Normal.

CanDelete

```
CanDelete <object>
```

Evaluates if Delete on the Edit menu would be available with <object> selected.

Possible return values are Unknown, Disabled and Normal.

Miscellaneous commands

Locate

```
Locate <locatestring>
```

This function locates an object described by `<locatestring>`. This string must be understood by a DataServer.

SaveDocument

```
SaveDocument <filename>
```

This command saves the file <filename>. This file must be opened in Rational Systems Tester.

This command is related to the OpenDocument command.

SaveAll

```
SaveAll [<filename>]
```

This command saves the whole content of a project if the <filename> of the project is given (this project must be opened in Rational Systems Tester), or saves everything if there is no parameter.

The <filename> parameter can be only the name of a project file, or the full path of a project file.

FlushEvents

```
FlushEvents
```

This command refreshes all Rational Systems Tester windows.

View Data in the Script Wizard

Simply navigate in the UML meta-model and click on Eval script code (always displayed at the top of the page). Data is displayed in the Browse tab of the Output window or in the Script tab for textual information.

Create Tcl Script Code

1. Start the Script Wizard by clicking Script Wizard in the Tools menu. You cam also click the Script Wizard button in the toolbar.
2. Click project or TTCN-3.
3. Click Create script code. The script is displayed.
4. Click the Execute Script button in the toolbar.
5. In the File menu, click Save.

Copy Tcl Script Code

1. Start the Script Wizard by clicking Script Wizard in the Tools menu. You can also click the Script Wizard button in the toolbar.
2. Click project or TTCN-3.
3. Browse to the code you wish to copy.
4. Click Copy script code.
5. Paste the copied code into another script or document.

Customize the Rational Systems Tester Views

1. Start the Script Wizard by clicking Script Wizard in the Tools menu. You can also click the Script Wizard button in the toolbar.
2. Click project or TTCN-3.
3. Click Create script code. The script is displayed.
4. Use one of the following functions:

   ```
   fwreportInit tabName [iconFile] [-keep] [-cb tclFile]
   ```

5. Click the Execute Script button in the menu bar. You can also press CTRL+F5.

   This activates a new tab in the workspace window, when needed.

   – If `iconFile` is present, then this file (that is, the icon) will be used as the tab image.

   – If -keep is present then the content of the tab (if any) is kept, otherwise the tab is reset.

   – If the `-cb tclFile` is present, then the associated Tcl script will be evaluated when you double-click a tree object, and the `OnDoubleClick` procedure of this Tcl script will be called with this object as parameter.

```
fwreport [-expanded] identObj [parentObj]:
```

Adds a new object in the tree.

– If -expanded is present and if the object has children, the node will be expanded, if not, it will be collapsed.

– If `parentObj` is present the node is created as a child of the node corresponding to `parentObj`.

```
reportInit [-check] [-cb tclfile] [column-label column-
width column-alignment]+:
```

Creates a new report tab in the Output window.

- If the -check is used, a check box is displayed before each result line. These check boxes allow more powerful navigation: only checked lines will be selected when you navigate by using the F4 key, others are ignored.
- If the `-cb tclFile` is present the associated Tcl script will be evaluated when you double-click on a line object, and the `OnDoubleClick` procedure of this Tcl script will be called with the double-clicked object as parameter. For each column-label column-width column-alignment a new column is created with the corresponding label, width and alignment.

Note

An alignment of 0 means left align and an alignment of 1 means right.

For sorting, columns right aligned are considered as number and columns left aligned are considered as text.

```
report sourceObject [label]*:
```

Adds new lines in a report tab.

- The `sourceObject` will be used to fill the first column (icon + text) automatically.
- The labels are used to fill the remaining columns.

```
output msg:
```

Allows the Tcl script to print a message in the Tcl output tab.

10

Runtime System APIs

Runtime Layer API

This interface contains all the services that the RTS provides. It is both used by the generated code from the IBM Rational Systems Tester Compiler and from user implementation such as a non-TRI integration, codecs systems and log mechanisms.

Although most functions in the RTL interface are public and can be used by anyone, a number of them are only intended to be from the code generated by the Rational Systems Tester Compiler.

See also

"RTL Function for Generated Code Only" on page 500 for a complete listing.

RTL Type Definitions

Descriptions of the C types that are used in the Runtime Layer API are spread out to the functions where they are most relevant. Most of these types are accessed through functions, their actual, underlying, representation is not meant to be public. However, a few types (for example `t3rt_symbol_entry_t`) have a public representation which will be detailed where appropriate. Type definitions of function prototypes are described where they are used (functions used in codecs registration for instance).

t3rt_context_t

Throughout the execution an object of the type `t3rt_context_t` is passed around to all RTS functions. It contains information about the current component and a lot of other data.

Example 24: Function with t3rt_context_t parameter ----------

t3rt_value_delete

Deletes the value and (recursively) all the value elements.

```
void t3rt_value_delete (t3rt_value_t *value, t3rt_context_t
context);
```

RTL Type Functions

These are the type related functions that instantiates values and accesses type information. Types are generated statically, so the functions have read-only access, no dynamic type construction exists.

Note

The type representation is visible from a C perspective but should never be directly de-referenced since that representation can change without notice. Always use the type access functions.

Each user-defined type have generated type descriptors and the built-in types of the TTCN-3 language has the following `t3rt_type_t` type descriptor constants:

```
t3rt_integer_type
t3rt_float_type
t3rt_boolean_type
t3rt_verdicttype_type
t3rt_default_type
t3rt_charstring_type
t3rt_bitstring_type
t3rt_octetstring_type
t3rt_hexstring_type
t3rt_universal_charstring_type
t3rt_char_type
t3rt_universal_char_type
t3rt_address_type
t3rt_timer_type
t3rt_objectidentifier_type
t3rt_binary_string_type (added type, not TTCN-3)
```

There are also two special type descriptor constants that represent an illegal type and a non-existing type. Many RTL type-related functions are not applicable for these two type descriptors:

```
t3rt_illegal_type
t3rt_undefined_type
```

RTL Type Related Type Definition

The following are the C types used in the type related functions:

t3rt_type_t

A type descriptor representing a TTCN-3 type. It is a very central entity of the RTS. The type descriptors are all static and are either statically defined by the RTS or generated by the Rational Systems Tester Com-

piler. The structural definition of this descriptor happens to be public but should never be accessed in any other way than with the type access functions.

t3rt_type_kind_t

Represents the type kind of the TTCN-3 type, integer and record, and so on. The difference between a “type” and the “type kind” is that the record type in TTCN-3 is not a type on its own but have to be defined further, while an integer is both a type and a type kind. So, the type descriptor for the user-defined TTCN-3 type MyRecord will have the type kind record.

t3rt_encoding_attr_t

Contains information about the “with encode”, “with variant”, “with display“ and “with extension“ attribute if one is available for the type. All attributes have this representation but are retrieved by four different access functions t3rt_type_encode_attribute, t3rt_type_variant_attribute, t3rt_type_display_attribute and t3rt_type_extension_attribute. Additional functions may be used to query attributes of type fields using either their names or indexes.

t3rt_field_properties_t

A numeric type that reflects the properties of a field in a structured type. Retrieved through t3rt_type_field_properties.

t3rt_long_integer_t

The signed 64-bit integer type representation. Used, for example, in t3rt_value_set_integer.

t3rt_unsigned_long_integer_t

The unsigned 64-bit integer type representation.

t3rt_type_instantiate_value

Creates a new value instance of a type.

```
t3rt_value_t t3rt_type_instantiate_value
        (t3rt_type_t type,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

`type`	A TTCN-3 built-in or a generated type descriptor.
`strategy`	Memory allocation strategy.

Description

This function creates an uninitialized value instance of the given type.

In the majority of cases, the type descriptor is directly available when a value has to be instantiated (for example in decoders). In those cases it is straightforward to make the instantiation.

In cases where values of TTCN-3 built-in types must be created, there are constant type descriptors to use (for example the t3rt_integer_type type descriptor to instantiate a pure integer value).

In rare cases, if a value has to be instantiated when no type descriptor is present, the type descriptor can be located (by name) through the symbol table. See RTL Symbol Table Functions for accessing type descriptors from user-defined types.

Memory for the new instance is allocated according to the specified strategy.

After the value has been allocated, the post-processing function of the type, if present, is called.

It is not possible to instantiate signature values from signature types.

Example Usage

```
/* Instantiate a value of a given type descriptor 'type'*/
t3rt_value_t builtin_val =
t3rt_type_instantiate_value(type, t3rt_temp_alloc_c, ctx);
...
/* Instantiate an integer value */
t3rt_value_t builtin_val =
t3rt_type_instantiate_value(t3rt_integer_type,
t3rt_temp_alloc_c, ctx);
...
/* Instantiate a value of type MyType. */
t3rt_type_t type = t3rt_find_element("MyType", ctx);
t3rt_value_t my_val = t3rt_type_instantiate_value(type-
>type_descriptor, t3rt_temp_alloc_c, ctx);
```

Return Values

The newly created value. This never returns any special value constants (t3rt_no_value_c, for example), a test case error will be generated if a value can not be instantiated and the current test case will be terminated.

See also

"RTL Symbol Table Functions" on page 490

"t3rt_type_field_type" on page 257

"t3rt_type_template_base_type" on page 269

"t3rt_type_array_contained_type" on page 269

"t3rt_value_kind" on page 279

t3rt_type_instantiate_named_value

Creates a new value instance of the type with a given name.

```
t3rt_value_t t3rt_type_instantiate_named_value
        (t3rt_type_t type,
         const char* name,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

`type`	The type to instantiate.
`name`	Name of the value.
`strategy`	Memory allocation strategy.

Description

Same as t3rt_type_instantiate_value but with a `charstring` typed label attached to it.

t3rt_type_check

Verifies that a value corresponds to a type.

```
bool t3rt_type_check
```

```
        (t3rt_type_t type,
         t3rt_value_t value,
         t3rt_context_t ctx);
```

Parameters

type	The type to check against.
value	The value to check.

Description

Checks if a value is compatible to a type. Built-in checks are made and a call is made to the generated type check function which means that, for example, type restrictions are checked.

Return Values

True if the value is compatible with the type, otherwise false.

See also

"t3rt_type_is_equal" on page 251

t3rt_type_is_equal

Checks if two type descriptors represent the same type.

```
bool t3rt_type_is_equal
        (t3rt_type_t type1,
         t3rt_type_t type2,
         t3rt_context_t ctx);
```

Parameters

type1	A valid type descriptor.
type2	A valid type descriptor.

Description

Checks if two type descriptors are equal. Since it is not sufficient to compare the type descriptors by the equality operator in C, this predicate should be used instead of direct t3rt_type_t instance comparison.

Return Values

True if the types are the same, otherwise false.

See also

"t3rt_type_check" on page 250

t3rt_type_kind

Retrieves the type kind from the type.

```
t3rt_type_kind_t t3rt_type_kind
        (t3rt_type_t type,
         t3rt_context_t ctx);
```

Parameters

type	A valid type descriptor.

Description

All type descriptors are of a specific type kind. The difference of a type and a type kind is, for example, that integer is a type but record is a type kind. This is because you can not instantiate a value of type record, it can only be instantiated from a user-defined type based on record.

Return Value

The kind of the type, see t3rt_type_kind_t for applicable values.

t3rt_type_parent

Retrieves the parent type descriptor from the type.

```
t3rt_type_t t3rt_type_parent
        (t3rt_type_t type,
         t3rt_context_t ctx);
```

Parameters

type	A valid type descriptor.

Description

While t3rt_type_field_type function obtains field type for a given structured type descriptor this function behaves vice versa. For a given type that should represent field type of some structured type (e.g. record) it returns type descriptor of that parent type.

Return Value

Valid type descriptor if given type descriptor represents field type, NULL otherwise.

t3rt_type_name, t3rt_type_definition_name

Returns the name of the type.

```
const char *t3rt_type_name
        (t3rt_type_t type,
         t3rt_context_t ctx);

const char *t3rt_type_definition_name
        (t3rt_type_t type,
         t3rt_context_t ctx);
```

Parameters

type	The type to access.

Description

Accesses the name of the type. The two functions only differ when the type is imported.

Return Values

If the type is local (that is, not imported) the returned name will be the plain type name.

If the type is imported, t3rt_type_name will return “<m>.<t>” where <m> is the imported module and <t> the type name, and t3rt_type_definition_name will just return the unqualified type name.

See also

"t3rt_type_module, t3rt_type_definition_module" on page 254

t3rt_type_module, t3rt_type_definition_module

Returns the module name of a type.

```
const char *t3rt_type_module
        (t3rt_type_t type,
         t3rt_context_t ctx);

const char *t3rt_type_definition_module
        (t3rt_type_t type,
         t3rt_context_t ctx);
```

Parameters

type	The type to access.

Description

Accesses the module of the type. The two functions only differ if the type is imported.

Return Values

If the type is local (that is, not imported) both functions return the same, obvious, module name.

If the type is imported, t3rt_type_module returns the importing module name and t3rt_type_definition_module returns the module name from where the type was imported.

See also

"t3rt_type_name, t3rt_type_definition_name" on page 253

t3rt_type_qualified_name

Returns the name of a type qualified with the name of a module name.

```
const char *t3rt_type_qualified_name
        (t3rt_type_t type,
         t3rt_context_t ctx);
```

Parameters

`type`	The type to access.

Description

Returns a type name which is qualified with a module name, separated by a '.' (PERIOD) character, for example "Mod1.MyType".

The module name is always the module in which the type is defined.

Return Values

Returns a qualified name, allocated in temporary memory.

See also

"t3rt_type_name, t3rt_type_definition_name" on page 253

"t3rt_type_module, t3rt_type_definition_module" on page 254

t3rt_type_field_count

Returns number of sub-fields of the specified type.

```
unsigned long t3rt_type_field_count
        (t3rt_type_t type,
         t3rt_context_t ctx);
```

Parameters

`type`	A valid type descriptor of a structured type.

Description

This is applicable to structured types (for example records and sets) where the number returned in the number of fields in the type declaration.

It is also applicable to template types (number of formal parameters), signature types (number of formal parameters) and enumerated types (number of enumerated elements).

Zero field in the signature type represents type of the returned value.

Return Values

The number of “fields” in the type, zero if not applicable or no fields are found.

t3rt_type_field_name

Returns the name associated with indicated sub-field of the specified type.

```
const char * t3rt_type_field_name
        (t3rt_type_t type,
         unsigned long field_index,
         t3rt_context_t ctx);
```

Parameters

type	A valid type descriptor.
field_index	A valid field index of the type descriptor. Indexes always starts from zero.

Description

This is applicable to structured types (for example records and sets) where the name is the name of the field in the type declaration.

It is also applicable to template types (name of formal parameter at index), signature types (name of formal parameter at index) and enumerated types (name of enumerated element at index).

Indices always starts from zero in all field access by index. If the index is not valid, a test case error will be generated and the test case will terminate.

Zero field in the signature type represents type of the returned value.

Return Values

The name of the “field”.

t3rt_type_field_index

Returns the index associated with indicated sub-field of the specified type.

```
unsigned long t3rt_type_field_index
        (t3rt_type_t type,
```

```
        const char *field_name,
        t3rt_context_t ctx);
```

Parameters

value	A valid type descriptor.
field_name	A valid field name of the type descriptor.

Description

This is applicable to structured types that have declared fields with names (for example records and sets).

It is also applicable to template types (index of named formal parameter), signature types (index of names formal parameter) and enumerated types (index of named enumerated element).

Indices always starts from zero in all field access by index.

Return Values

The index of the named “field”. If no field could be found, t3rt_no_field_index_c is returned.

t3rt_type_field_type

Returns the type associated with indicated sub-field of the specified type.

```
t3rt_type_t t3rt_type_field_type
        (t3rt_type_t type,
         unsigned long field_index,
         t3rt_context_t ctx);
```

Parameters

type	A valid type descriptor
field_index	A valid field index of the type descriptor. Indexes always starts from zero.

Description

This is applicable to structured types (for example records and sets) with typed fields.

It is also applicable to template types (type of formal parameter at index), signature types (type of formal parameter at index) and enumerated types (type of enumerated element at index).

Indices always starts from zero in all field access by index. If the index is not valid, a test case error will be generated and the test case will terminate.

Zero field in the signature type represents type of the returned value.

Return Values

The type of the indicated field.

t3rt_type_field_properties

Returns the type associated with indicated sub-field of the specified type.

```
t3rt_field_properties_t t3rt_type_field_properties
        (t3rt_type_t type,
         unsigned long field_index,
         t3rt_context_t ctx);
```

Parameters

`type`	A valid type descriptor representing a structured type, for example a record, a template or a signature type.
`field_index`	A valid index with respect to the number of fields in the type. Indexes always starts from zero.

Description

Retrieves the field properties of the given field in this structured type.

Zero field in the signature type represents type of the returned value.

Return Values

A field can have one of the following values:

Field Property	Description
t3rt_field_property_no_property_c	No properties
t3rt_field_property_mandatory_c	This is the default attribute when the field is not optional.
t3rt_field_property_optional_c	The field is optional according to the type definition (which is either a record or a set type).
t3rt_field_property_in_value_c	The field represents an in parameter of a type with formal parameters (for example a template or a signature).
t3rt_field_property_out_value_c	The field represents an out parameter of a type with formal parameters (for example a template or a signature).
t3rt_field_property_inout_value_c	The field represents an inout parameter of a type with formal parameters (for example a template or a signature).
t3rt_field_property_return_value_c	The field represents the return type of a type that has a return type (for example a signature).

t3rt_type_enum_named_values_count

Returns number of named values of the specified enumerated type.

```
unsigned long t3rt_type_enum_named_values_count
        (t3rt_type_t type,
         t3rt_context_t ctx);
```

Parameters

`type`	A valid type descriptor for an enumerated type.

Return Values

The number of enumerated elements in the type.

t3rt_type_enum_name_by_index

Returns the name associated with the indicated position of the specified enumerated type.

```
const char * t3rt_type_enum_name_by_index
        (t3rt_type_t type,
         unsigned long index,
         t3rt_context_t ctx);
```

Parameters

type	A valid type descriptor for an enumerated type.
index	A valid index within the range from 0 to one less than the number of enumerated elements. A test case will be generated otherwise.

Return Values

Returns the name of the enumerated element at the given index.

t3rt_type_enum_number_by_index

Returns the number associated with the indicated position of the specified enumerated type.

```
t3rt_long_integer_t t3rt_type_enum_number_by_index
        (t3rt_type_t type,
         unsigned long index,
         t3rt_context_t ctx);
```

Parameters

type	A valid type descriptor for an enumerated type.
index	A valid index within the range from 0 to one less than the number of enumerated elements. A test case will be generated otherwise.

Return Values

The enumeration number of the named element.

t3rt_type_enum_name_by_number

Returns the name associated with the indicated number of the specified enumerated type.

```
const char * t3rt_type_enum_name_by_number
        (t3rt_type_t type,
         t3rt_long_integer_t named_value_number,
         t3rt_context_t ctx);
```

Parameters

type	A valid type descriptor for an enumerated type.
named_value_number	A valid number in the enumerated type.

Return Values

The name of the enumerated element for the given enumerated (integer) value or NULL if the number can not be found.

t3rt_type_enum_number_by_name

Returns the number associated with the indicated name of the specified enumerated type.

```
t3rt_long_integer_t t3rt_type_enum_number_by_name
        (t3rt_type_t type,
         const char *named_value_name,
         t3rt_context_t ctx);
```

Parameters

type	A valid type descriptor for an enumerated type.
named_value_name	A valid enumerated element in the enumerated type.

Return Values

The enumeration number of the named element or t3rt_no_enum_number_c if no such named element can be found.

t3rt_type_field_encode_attribute_by_name

Returns encoding attribute associated with a type's field.

```
t3rt_encoding_attr_t
t3rt_type_field_encode_attribute_by_name
        (t3rt_type_t type,
         const char *fieldname,
         t3rt_context_t ctx);
```

Parameters

type	A valid type descriptor.
field_name	Type field name.

Return Values

Returns “with encode” attribute descriptor for the given type field identified using its field name.

t3rt_type_field_encode_attribute_by_index

Returns encoding attribute associated with a type's field.

```
t3rt_encoding_attr_t
t3rt_type_field_encode_attribute_by_index
        (t3rt_type_t type,
         unsigned long index,
         t3rt_context_t ctx);
```

Parameters

type	A valid type descriptor.
index	Type field index.

Return Values

Returns “with encode” attribute descriptor for the given type field identified using its field index.

t3rt_type_field_variant_attribute_by_name

Returns variant attribute associated with a type's field.

```
t3rt_encoding_attr_t
```

```
t3rt_type_field_variant_attribute_by_name
        (t3rt_type_t type,
         const char *fieldname,
         t3rt_context_t ctx);
```

Parameters

type	A valid type descriptor.
field_name	Type field name.

Return Values

Returns “with variant” attribute descriptor for the given type field identified using its field name.

t3rt_type_field_variant_attribute_by_index

Returns variant attribute associated with a type's field.

```
t3rt_encoding_attr_t
t3rt_type_field_variant_attribute_by_index
        (t3rt_type_t type,
         unsigned long index,
         t3rt_context_t ctx);
```

Parameters

type	A valid type descriptor.
index	Type field index.

Return Values

Returns “with variant” attribute descriptor for the given type field identified using its field index.

t3rt_type_field_display_attribute_by_name

Returns display attribute associated with a type's field.

```
t3rt_encoding_attr_t
t3rt_type_field_display_attribute_by_name
        (t3rt_type_t type,
         const char *fieldname,
         t3rt_context_t ctx);
```

Parameters

type	A valid type descriptor.
field_name	Type field name.

Return Values

Returns “with display” attribute descriptor for the given type field identified using its field name.

t3rt_type_field_display_attribute_by_index

Returns display attribute associated with a type's field.

```
t3rt_encoding_attr_t
t3rt_type_field_display_attribute_by_index
        (t3rt_type_t type,
         unsigned long index,
         t3rt_context_t ctx);
```

Parameters

type	A valid type descriptor.
index	Type field index.

Return Values

Returns “with display” attribute descriptor for the given type field identified using its field index.

t3rt_type_field_extension_attribute_by_name

Returns extension attribute associated with a type's field.

```
t3rt_encoding_attr_t
t3rt_type_field_extension_attribute_by_name
        (t3rt_type_t type,
         const char *fieldname,
         t3rt_context_t ctx);
```

Parameters

type	A valid type descriptor.
field_name	Type field name.

Return Values

Returns “with extension” attribute descriptor for the given type field identified using its field name.

t3rt_type_field_extension_attribute_by_index

Returns extension attribute associated with a type's field.

```
t3rt_encoding_attr_t
t3rt_type_field_extension_attribute_by_index
        (t3rt_type_t type,
         unsigned long index,
         t3rt_context_t ctx);
```

Parameters

type	A valid type descriptor.
index	Type field index.

Return Values

Returns “with extension” attribute descriptor for the given type field identified using its field index.

t3rt_type_encode_attribute

Returns the encode attribute associated with the type.

```
t3rt_encoding_attr_t t3rt_type_encode_attribute
        (t3rt_type_t type,
         t3rt_context_t ctx);
```

Parameters

type	A valid type descriptor.

Return Values

Returns “with encode” attribute descriptor for the given type.

t3rt_type_variant_attribute

Returns the variant attribute associated with the type.

```
t3rt_encoding_attr_t t3rt_type_variant_attribute
```

```
        (t3rt_type_t type,
         t3rt_context_t ctx);
```

Parameters

type	A valid type descriptor.

Return Values

Returns “with variant” attribute descriptor for the given type.

t3rt_type_display_attribute

Returns the display attribute associated with the type.

```
t3rt_encoding_attr_t t3rt_type_display_attribute
        (t3rt_type_t type,
         t3rt_context_t ctx);
```

Parameters

type	A valid type descriptor.

Return Values

Returns “with display” attribute descriptor for the given type.

t3rt_type_extension_attribute

Returns the extension attribute associated with the type.

```
t3rt_encoding_attr_t t3rt_type_extension_attribute
        (t3rt_type_t type,
         t3rt_context_t ctx);
```

Parameters

type	A valid type descriptor.

Return Values

Returns “with extension” attribute descriptor for the given type.

t3rt_encoding_attr_get_specifier

Returns the free text string specifier of the encode, variant, display or extension attribute.

```
const char* t3rt_encoding_attr_get_specifier
        (t3rt_encoding_attr_t attr,
         t3rt_context_t ctx);
```

Parameters

attr	A valid attribute descriptor.

Description

This function returns character string that represents attribute specifier as defined in the TTCN-3 module. ‘attr’ parameter should be a valid attribute descriptor previously obtained by one of the attribute extraction functions (e.g. t3rt_type_encode_attribute).

Return Values

Returns character string attribute specifier as defined in the TTCN-3 module.

t3rt_encoding_attr_is_override

Predicates telling if the attribute specification has been overridden by enveloping TTCN-3 element.

```
bool t3rt_encoding_attr_is_override
        (t3rt_encoding_attr_t attr,
         t3rt_context_t ctx);
```

Parameters

attr	A valid attribute descriptor.

Description

This function tests whether given attribute has been overridden by the attribute specification of enveloping TTCN-3 element, which declares attribute with “override“ statement. This function only informs about the happened overriding. There is no way of accessing original attribute specifier.

Return Values

Returns true if attribute specifier represents attribute of enveloping TTCN-3 element declared with “override“ statement, false otherwise.

t3rt_type_array_size

Retrieves the size of the array type.

```
unsigned long t3rt_type_array_size
        (t3rt_type_t array_type,
         t3rt_context_t ctx);
```

Parameters

array_type	The array type itself.

Description

Retrieves the size in elements of the given array type.

Return Values

The size of the array type.

t3rt_type_array_base_index

Retrieves lower subscription index of the array type.

```
unsigned long t3rt_type_array_base_index
        (t3rt_type_t array_type,
         t3rt_context_t ctx);
```

Parameters

array_type	The array type itself.

Description

Retrieves the lower subscription index of the given array type.

Return Values

For the majority of the array types this function returns zero. This is the case for the below mentioned ‘my_array’ type.

```
type integer my_array[10];
```

‘my_array’ type has size of 10 with base index equal to 0.

When array type defines lower boundary then base index may be greater than zero.

```
type integer another_array[5..10];
```

‘another_array’ type has size of 6 with base index equal to 5.

t3rt_type_array_contained_type

Retrieves the type of the elements of this array type.

```
t3rt_type_t t3rt_type_array_contained_type
        (t3rt_type_t array_type,
         t3rt_context_t ctx);
```

Parameters

array_type	The array type itself.

Description

Retrieves the type descriptor that is the type of the value elements of this array type.

Return Values

The valid type descriptor for the array elements.

t3rt_type_template_base_type

Retrieves the base type of this template type.

```
t3rt_type_t t3rt_type_template_base_type
        (t3rt_type_t type,
         t3rt_context_t ctx);
```

Parameters

`type`	The template type itself.

Description

Retrieves the base type of the given template type. In the template definition

```
template integer my_template := 7
```

calling this function against ‘my_template’ type descriptor returns type descriptor for the ‘integer’ data type.

Return Values

The valid type descriptor for the base template type.

t3rt_template_description

Retrieves unparsed template description as defined in the TTCN-3 module

```
const char * t3rt_template_description
        (t3rt_type_t type,
         t3rt_context_t ctx);
```

Parameters

`type`	The template type itself.

Description

Retrieves the unparsed template constraint as defined in the TTCN-3 module. This function relies on the information generated by the compiler. Usually template constraint is truncated to 100 characters to make generated code smaller however you may control the maximum size of generated template description using “-l <max_length>“ compiler option. If template constraint has been truncated then compiler appends “...“ string to the end of it.

Return Values

Returns character string that represents template constraint as defined in the TTCN-3 module.

t3rt_type_set_encoder

Sets the encoder function of the type descriptor.

```
void t3rt_type_set_encoder
        (t3rt_type_t type,
         t3rt_encoder_function_t encoder_function,
         t3rt_context_t ctx);
```

Parameters

type	The type to set encoder function for.
encoder_function	Function to set.

Description

This will set the encoder function of the type descriptor. There can be only one encoder function for each type.

The encoder function will be called prior to sending a value on a port.

This function is typically called from a registered codecs system's setup function.

If you are using coders implemented with TCI CD interface then you need to set t3rt_tci_encode as an encoder function.

See also

"RTL Codecs Functions" on page 473

t3rt_type_set_decoder

Sets the decoder function of the type descriptor.

```
void t3rt_type_set_decoder
        (t3rt_type_t type,
         t3rt_decoder_function_t decoder_function,
         t3rt_context_t ctx);
```

Parameters

type	The type to set decoder for.
decoder_function	Function to set.

Description

This will set the decoder function of the type descriptor. There can be only one decoder function for each type.

The decoder function will be used to decode an incoming value before any TTCN-3 operations (for example match) are performed.

This function is typically called from a registered codecs system's setup function.

If you are using coders implemented with TCI CD interface then you need to set t3rt_tci_decode as a decoder function.

See also

"RTL Codecs Functions" on page 473

t3rt_type_get_encoder

Retrieve the encoder function of the type descriptor.

```
t3rt_encoder_function_t t3rt_type_get_encoder
        (t3rt_type_t type,
         t3rt_context_t ctx);
```

Parameters

type	Type descriptor from which to retrieve the encoder function.

Description

This will retrieve the encoder function of the type descriptor. There can be only one encoder function for each type.

The encoder function will be called prior to sending a value on a port.

Return Values

The set function pointer or NULL if none is present.

See also

"RTL Codecs Functions" on page 473

t3rt_type_get_decoder

Retrieve the decoder function of the type descriptor.

```
t3rt_decoder_function_t t3rt_type_get_decoder
        (t3rt_type_t type,
         t3rt_context_t ctx);
```

Parameters

`type`	Type descriptor from which to retrieve the decoder function.

Description

This will retrieve the registered decoder function of the type descriptor. There can be only one decoder function for each type.

The decoder function will be used to decode an incoming value before any TTCN-3 operations (for example match) are performed.

Return Values

The set function pointer or NULL if none is present.

See also

"RTL Codecs Functions" on page 473

RTL Value Functions

These are the value related functions that handles the variables, timers, ports, and component references in TTCN-3.

The actual value representation is internal and the values can only be manipulated through functions.

The following value constants (of type t3rt_value_t) are defined, and where and when they are used is described among the individual API functions.

```
t3rt_illegal_value_c
t3rt_omit_value_c
t3rt_anyone_value_c
t3rt_anyornone_value_c
t3rt_no_value_c
t3rt_not_used_value_c
t3rt_value_any_c
t3rt_value_all_c
t3rt_value_no_return_c
t3rt_timeout_exception_c
t3rt_value_null_address_c
t3rt_value_true_c
t3rt_value_false_c
t3rt_value_verdict_pass_c
t3rt_value_verdict_fail_c
t3rt_value_verdict_inconc_c
t3rt_value_verdict_none_c
t3rt_value_verdict_error_c
t3rt_value_null_default_reference_c
t3rt_value_null_component_reference_c
```

RTL Value Related Type Definitions

t3rt_value_t

The representation of a variable, timer, component (reference), port (reference), and so on. Basically all entities that can be passed as parameters to functions, test step, test cases, and so on, or declared within component types.

t3rt_verdict_t

An enumeration of all the possible verdicts in TTCN-3.

t3rt_value_copy

Returns a newly created “deep” copy of the value.

```
t3rt_value_t t3rt_value_copy
        (const t3rt_type_t value,
         const t3rt_alloc_strategy_t strategy,
         t3rt_context_t context);
```

Parameters

value	The value to copy.
alt_index	Memory allocation strategy for the created copy of the original value.

Description

Some kinds of values (timers, port and component records) cannot be copied. Copied value has to be fully initialized. Use t3rt_value_is_initialized to check whether given value is initialized. Calling this function for not-initialized values results in test case error.

Return Values

Copy of given value allocated according to specified memory allocation strategy.

t3rt_value_parent

Returns parent (enveloping) value of the given value

```
t3rt_value_t t3rt_value_parent
        (const t3rt_value_t value,
         t3rt_context_t context);
```

Parameters

value	Element of the structured value.

Description

This function is applicable to the various types of values: records, sets, arrays, unions, signatures, etc. Given an element of a structured value it returns reference to the compound value that contains given element. If “value“ is not an element of a compound value then this function returns t3rt_no_value_c special value.

Example Usage

```
/* Suppose 'record_value' is a value of record type */
/* 'element_value' is a first field in this record value */

t3rt_value_t element_value =
```

```
t3rt_value_field_by_index(record_value, 0, ctx);
...
/* Get reference to record value using field value */
t3rt_value_t parent_value =
t3rt_value_parent(element_value, ctx);
...
/* Parent value for 'element_value' is a 'record_value'*/
assert(parent_value == record_value);
```

Return Values

Parent value for the given element value.

t3rt_value_is_dynamic_template

Checks whether underlying value is a generic value or a dynamic template.

```
bool t3rt_value_is_dynamic_template
        (const t3rt_value_t value,
         t3rt_context_t context);
```

Parameters

value	Value to be checked.

Description

This function may be used to distinguish dynamic templates from generic values.

Return Values

Returns true if provided value represents template, false if it's a generic value.

t3rt_value_set_union_alternative_by_index

Returns the newly created uninitiated value for the alternative.

```
t3rt_value_t t3rt_value_set_union_alternative_by_index
        (t3rt_value_t union_value,
         unsigned long alt_index,
         t3rt_context_t context);
```

Parameters

union_value	The union value to initialize.
alt_index	The index of the alternative to select. An index that does not correspond to a valid alternative according to the type will give a test case error.

Description

Initializes the union value by calling t3rt_type_instantiate_value for the type, corresponding to alternative. The initialization uses the same allocation strategy as the union value. Returns the newly created uninitiated value for the alternative.

If the new alternative is different from the current alternative, the allocated value will be de-allocated and replaced by a newly instantiated (uninitiated) value. If the new alternative is the same as the current one, this function will keep the existing value.

Return Values

The union value given as input parameter.

t3rt_value_set_union_alternative_by_name

Returns the newly created uninitiated value for the alternative.

```
t3rt_value_t t3rt_value_set_union_alternative_by_name
        (t3rt_value_t union_value,
         const char* alt_name,
         t3rt_context_t context);
```

Parameters

union_value	The union value to initialize.
alt_name	The name of the alternative to select. A name that does not correspond to a valid alternative according to the type will give a test case error.

Description

Initializes the union value by calling t3rt_type_instantiate_value for the type, corresponding to alternative. The initialization uses the same allocation strategy as the union value. Returns the newly created uninitiated value for the alternative.

If the new alternative is different from the current alternative, the allocated value will be de-allocated and replaced by a newly instantiated (uninitiated) value. If the new alternative is the same as the current one, this function will keep the existing value.

Return Values

The union value given as input parameter.

t3rt_value_delete

Deletes the value and (recursively) all the value elements.

```
void t3rt_value_delete
        (t3rt_value_t *value,
         t3rt_context_t context);
```

Parameters

value	Address of a value to be deleted.

Description

It's not necessary to delete values allocated in the temporary memory (i.e. values allocated with t3rt_temp_alloc_c strategy) since deallocation of objects in temporary memory is performed automatically by the runtime system.

t3rt_value_is_initialized

Checks if value and all its elements (recursively) are initialized.

```
bool t3rt_value_is_initialized
        (t3rt_value_t value,
         t3rt_context_t context);
```

Parameters

value	Value to be checked.

Description

Some of value operations (e.g. copy, assign, encode) requires used value to be fully initialized. Arrays (including “record of” and “set of”) are treated as not initialized if they have undefined elements. Omitted optional fields in records and sets should be explicitly assigned with “omit“ value using t3rt_value_set_omit otherwise they also are treated as not-initialized.

Return Values

Returns “true“ if value is initialized, false otherwise.

t3rt_value_kind

Calls the t3rt_type_kind function.

```
t3rt_type_kind_t t3rt_value_kind
        (t3rt_value_t value,
         t3rt_context_t context);
```

Parameters

value	Valid value descriptor.

Description

This function simply calls t3rt_type_kind for the value type, i.e. it’s the same as calling t3rt_type_kind(t3rt_value_type(value, context), context).

Return Values

Returns type kind of the value type.

t3rt_value_type

Returns the type descriptor for the type of the value.

```
t3rt_type_t t3rt_value_type
        (const t3rt_value_t value,
         t3rt_context_t context);
```

Parameters

value	Valid value descriptor.

Return Values

Returns the type descriptor for the type of the value.

t3rt_value_set_label

Sets a label (for example name) of a value.

```
void t3rt_value_set_label
        (t3rt_value_t value,
         t3rt_value_t label,
         t3rt_context_t context);
```

Parameters

value	The value to label.
label	The label of the value.

Description

This applies a label on a value. The label is also any kind of value to keep this general. It is most widely use with a `charstring` value signifying a variable name for instance.

It is applied automatically for instantiated ports and timers that are declared inside a component type definition when the component is instantiated.

The set label is retrieved by calling the t3rt_value_label function.

See also

"t3rt_type_instantiate_named_value" on page 250

t3rt_value_label

Retrieves the label of the value.

```
t3rt_value_t t3rt_value_label
        (const t3rt_value_t value,
         t3rt_context_t context);
```

Parameters

`value`	Valid value descriptor.

Description

The label is an arbitrary value that serves as a “name” for the value.

For generated entities, the name is a `charstring` typed value with the name of the declared entity (for example variables, timers, ports, and so on).

Return Values

The label value. If no label has been set the t3rt_no_value_c constant is returned.

t3rt_value_allocation_strategy

Returns the memory allocation strategy used for allocation of the value.

```
t3rt_alloc_strategy_t t3rt_value_allocation_strategy
        (const t3rt_value_t value,
         t3rt_context_t context);
```

Parameters

`value`	Valid value descriptor.

Description

All elements of a compound value are allocated always using one and the same strategy. It’s not possible to have some elements of a value to be allocated in permanent memory and other elements - in temporary memory.

Return Values

Returns the memory allocation strategy used for allocation of the value.

t3rt_value_string_length

Returns the length of the string value.

```
unsigned long t3rt_value_string_length
```

```
        (t3rt_value_t string_value,
         t3rt_context_t context);
```

Parameters

string_value	Value of one of the string types.

Description

This function operates on charstring, bitstring, octetstring, hexstring, universal charstring and binary string values. The length is measured in elements of a certain string type. Length of octetstring is measured in octets while each octet is represented by two hex symbols. Length of binary string is measured in bits.

Return Values

Length of a string value measured in elements.

t3rt_value_vector_size

Returns size of the vector.

```
unsigned long t3rt_value_vector_size
        (t3rt_value_t vector_value,
         t3rt_context_t context);
```

Parameters

vector_value	Value of one of the vector types.

Description

This function operates on setof, recordof, array, set, record, objectidentifier, signature, and template values. Undefined and omitted elements are also counted.

Return Values

The number of element in the vector.

t3rt_value_set_vector_size

Resizes a recordof or setof value by adding or removing elements when necessary.

```
void t3rt_value_set_vector_size
        (t3rt_value_t vector_value,
         const unsigned long new_size,
         t3rt_context_t context);
```

Parameters

vector_value	The vector value to set.
new_size	The desired size of the vector.

Description

This function operates only on setof and recordof.

When called, the size of the vector will be modified. If the vector is shortened, the truncated elements are de-allocated normally and if the vector is lengthened, the special value constant `t3rt_not_used_value_c` is added as placeholder for each new element.

To set fields of a vector, use “t3rt_value_assign_vector_element” on page 295.

t3rt_value_set_vector_empty

Initialize a vector value to being empty.

```
void t3rt_value_set_vector_empty
        (t3rt_value_t vector_value,
         t3rt_context_t context);
```

Parameters

vector_value	The vector value to set.

Description

This function operates on setof, recordof, array, record, set and signature values.When called, the value will be initialized to empty.

For arrays, the size of the array must be zero or a test case error will be generated. For recordof and setof if the value contains elements it will behave like t3rt_value_set_vector_size passing it zero length. For record, set and signature values this function will assign t3rt_not_used_value_c to all fields.

When implementing decoder for a vector type it's necessary to call this function for a record or set value even if record type definition doesn't contain fields.

To set fields of a vector, use "t3rt_value_assign_vector_element" on page 295.

t3rt_value_field_by_index

Returns the value of the indicated field.

```
t3rt_value_t t3rt_value_field_by_index
        (t3rt_value_t value,
         unsigned long field_index,
         t3rt_context_t context);
```

Parameters

value	Valid vector value.
field_index	Zero based position of the requested field.

Description

This function operates on setof, recordof, array, record, set, signature and template values. For all kinds of values except recordof and setof specified field index should not exceed ordinal index of last field value.

When applied to recordof and setof value while "field_index" is greater than value size this function expands recordof/setof assigning t3rt_not_used_value_c to all new elements.

If requested field value has not been defined then this function instantiates specified field value and returns uninitialized value instance. Returned value is lvalue (that is, the returned element can be used in assignment). There is no need to use t3rt_value_assign_vector_element after filling returned field value.

Return Values

Returns previously set or fresh instance of (if field has not been defined) field value.

t3rt_value_field_by_name

Returns the value of the indicated field.

```
t3rt_value_t t3rt_value_field_by_name
        (t3rt_value_t value,
         const char *field_name,
         t3rt_context_t context);
```

Parameters

`value`	Valid vector value.
`field_name`	Name of the requested field.

Description

This function is similar to t3rt_value_field_by_index but queries field value using given field name. This function operates only on records, sets, signatures and templates.

The field with the provided name should exist in the underlying type. The name of field may be obtained by its ordinal index inside structured type with the help of “t3rt_type_field_name” on page 256 function.

Return Values

Returns previously set or fresh instance of (if field has not been defined) field value.

t3rt_value_vector_element

Returns the value of the vector’s indicated element.

```
t3rt_value_t t3rt_value_vector_element
        (t3rt_value_t value,
         unsigned long element_index,
         t3rt_context_t context);
```

Parameters

value	Valid vector value.
element_index	Zero based position of the requested element.

Description

This function behaves similar to t3rt_value_field_by_index.

Return Values

Returns the value of the specified element.

t3rt_value_string_element

Returns the newly created string value containing the indicated element of the given string.

```
t3rt_value_t t3rt_value_string_element
        (t3rt_value_t string_value,
         unsigned long element_index,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t context);
```

Parameters

value	Valid string value.
element_index	Zero based position of the requested element.

Description

This function operates on charstring, octetstring, bitstring, hexstring and universal charstring type values. The type of the returned value depends on the type of the given string. For charstring and universal charstring values this function returns char and universal char values correspondingly. For other string types it returns value of the same type but containing only one (specified) element.

Specified element index should point to element within string boundaries otherwise test case error is generated.

t3rt_value_union_value

Returns the value of the union value.

```
t3rt_value_t t3rt_value_union_value
        (t3rt_value_t union_value,
         t3rt_context_t context);
```

Parameters

value	Valid union value.

Description

This function returns value that represents chosen union variant. If no union variant has been chosen then test case error is generated.

Return Values

Returns chosen union variant value.

t3rt_value_union_index

Returns the index of the current union alternative.

```
unsigned long t3rt_value_union_index
        (t3rt_value_t union_value,
         t3rt_context_t context);
```

Parameters

value	Valid union value.

Description

This function returns zero based index of chosen union variant.If no union variant has been chosen then -1 is returned.

Return Values

Returns index of chosen alternative or -1 if union alternative has not been set.

t3rt_value_get_integer

Extract the corresponding integer from the TTCN-3 runtime system value representation.

```
t3rt_long_integer_t t3rt_value_get_integer
        (const t3rt_value_t integer_value,
         t3rt_context_t context);
```

Parameters

integer_value	Value to extract from. If the value is not appropriate for the extraction operation a test case error will be generated.

Description

This is the function that is used for mapping simple values from the RTS value representation to a corresponding representation in the target language (currently the C language).

t3rt_value_get_enum_number

Extract integer of the corresponding enumeration from the TTCN-3 runtime system value representation.

```
t3rt_long_integer_t t3rt_value_get_enum_number
        (const t3rt_value_t enum_value,
         t3rt_context_t context);
```

Parameters

enum_value	Value to extract from. If the value is not appropriate for the extraction operation a test case error will be generated.

Description

This is a function that is used for mapping simple values from the RTS value representation to a corresponding representation in the target language (currently the C language).

t3rt_value_get_enum_name

Extract the corresponding name of the enumeration value.

```
const char* t3rt_value_get_enum_name
        (const t3rt_value_t enum_value,
         t3rt_context_t context);
```

Parameters

enum_value	Value to extract from. If the value is not appropriate for the extraction operation a test case error will be generated.

Description

This function is used to extract the name of the enumeration value according to the information in the value's type.

t3rt_value_get_float

Extract the corresponding floating-point value from the TTCN-3 runtime system value representation.

```
double t3rt_value_get_float
        (const t3rt_value_t float_value,
         t3rt_context_t context) ;
```

Parameters

float_value	Value to extract from. If the value is not appropriate for the extraction operation a test case error will be generated.

Description

This is the function that is used for mapping simple values from the RTS value representation to a corresponding representation in the target language (currently the C language).

t3rt_value_get_boolean

Extract the corresponding boolean value from the TTCN-3 runtime system value representation.

```
bool t3rt_value_get_boolean
        (const t3rt_value_t boolean_value,
         t3rt_context_t context);
```

Parameters

boolean_value	Value to extract from. If the value is not appropriate for the extraction operation a test case error will be generated.

Description

This is the function that is used for mapping simple values from the RTS value representation to a corresponding representation in the target language (currently the C language).

t3rt_value_get_char

Extract the corresponding character value from the TTCN-3 runtime system value representation.

```
char t3rt_value_get_char
        (const t3rt_value_t char_value,
         t3rt_context_t context);
```

Parameters

char_value	Value to extract from. If the value is not appropriate for the extraction operation a test case error will be generated.

Description

This is the function that is used for mapping simple values from the RTS value representation to a corresponding representation in the target language (currently the C language).

t3rt_value_get_string

Extract the corresponding string value from the TTCN-3 runtime system value representation.

```
const char* t3rt_value_get_string
        (const t3rt_value_t string_value,
         t3rt_context_t context);
```

Parameters

string_value	Value to extract from. If the value is not appropriate for the extraction operation a test case error will be generated.

Description

This is the function that is used for mapping simple values from the RTS value representation to a corresponding representation in the target language (currently the C language).

t3rt_value_get_universal_char

Extract the corresponding universal character value from the TTCN-3 runtime system value representation.

```
const t3rt_wide_char_t * t3rt_value_get_universal_char
        (const t3rt_value_t universal_char_value,
         t3rt_context_t context);
```

Parameters

universal_char_value	Value to extract from. If the value is not appropriate for the extraction operation a test case error will be generated.

Description

This is the function that is used for mapping simple values from the RTS value representation to a corresponding representation in the target language (currently the C language).

t3rt_value_get_universal_charstring

Extract the corresponding wide string value from the TTCN-3 runtime system value representation.

```
t3rt_wide_string_t t3rt_value_get_universal_charstring
        (const t3rt_value_t widestring_value,
         t3rt_context_t context);
```

Parameters

widestring_value	Value to extract from. If the value is not appropriate for the extraction operation a test case error will be generated.

Description

This is the function that is used for map simple values from the RTS value representation to a corresponding representation in the target language (currently the C language).

t3rt_value_get_binary_string

Extract the corresponding binary data value from the TTCN-3 runtime system value representation.

```
t3rt_binary_string_t t3rt_value_get_binary_string
        (const t3rt_value_t binary_value,
         t3rt_context_t context);
```

Parameters

binary_value	Value to extract from. If the value is not appropriate for the extraction operation a test case error will be generated.

Description

This is the function that is used for mapping simple values from the RTS value representation to a corresponding representation in the target language (currently the C language).

Note

This type of a binary_string value is not defined in ETSI ES 201 873-1 V2.2.1. This is a proprietary value type that has been introduced to be able to pass around generic binary data in a uniform way.

t3rt_value_get_verdict

Extract the corresponding verdict value from the TTCN-3 runtime system value representation.

```
t3rt_verdict_t t3rt_value_get_verdict
        (const t3rt_value_t verdict_value,
         t3rt_context_t context);
```

Parameters

verdict_value	Value to extract from. If the value is not appropriate for the extraction operation a test case error will be generated.

Description

This is the function that is used for mapping simple values from the RTS value representation to a corresponding representation in the target language (currently the C language).

t3rt_value_get_port_address

Extract the corresponding address from the TTCN-3 runtime system value representation.

```
t3rt_binary_string_t t3rt_value_get_port_address
        (const t3rt_value_t portref_value,
         t3rt_context_t context);
```

Parameters

portref_value	Value to extract from. If the value is not appropriate for the extraction operation a test case error will be generated.

Description

This is the function that is used for mapping simple values from the RTS value representation to a corresponding representation in the target language (currently the C language).

t3rt_value_get_objectid_element

Retrieves object identifier number from the specified position.

```
unsigned long t3rt_value_get_objectid_element
        (const t3rt_value_t objid_value,
         unsigned long index,
         t3rt_context_t context);
```

Parameters

objid_value	Value to extract from. If the value is not appropriate for the extraction operation a test case error will be generated.

Description

This function extracts object identifier element from the specified position. Index parameter should be a zero based integer value. If index is greater than the length of object identifier value minus 1 then test case error is generated. Use t3rt_value_vector_size function to get the length of object identifier value.

Return Values

Returns integer representing object identifier element at specified position.

t3rt_value_assign

Assign one, fully initialized, value to another.

```
void t3rt_value_assign
        (t3rt_value_t lvalue,
         const t3rt_value_t rvalue,
         t3rt_context_t context);
```

Parameters

`lvalue`	An instantiated value of the appropriate type to assign to.
`rvalue`	A fully initialized value to assign from.

Description

First, this function checks if rvalue is compatible with the type of the lvalue (done by calling the t3rt_type_check function). Then, for the values of basic types, it simply copies the target value from rvalue into lvalue. For all other types, it does an element-wise assignment recursively.

See also

"t3rt_type_instantiate_value" on page 248

"t3rt_value_copy" on page 274

"t3rt_value_assign_vector_element" on page 295

"t3rt_value_assign_string_element" on page 296

t3rt_value_assign_vector_element

Assigns the contents of the element into the indicated position of the vector.

```
void t3rt_value_assign_vector_element
        (t3rt_value_t vector,
         const unsigned long index,
         const t3rt_value_t element,
         t3rt_context_t context);
```

Parameters

`vector`	Valid value of one of vector types.
`index`	Zero based position in vector value to assign to.
`element`	Element to assigned to the specified position of vector value.

Description

This function operates on array, set, record, recordof, setof, template and signature values. For all kinds of values except recordof and setof specified field index should not exceed ordinal index of last field value (i.e. the size of compound value minus 1).

When applied to recordof and setof value while “field_index” is greater than value size this function expands recordof/setof assigning t3rt_not_used_value_c to all new elements.

This function may be used to assign omit to optional fields of record and set values.

Type check for vector value is performed after assignment.

t3rt_value_assign_string_element

Assigns the content of the one_char string into the indicated position of the string.

```
void t3rt_value_assign_string_element
        (t3rt_value_t string,
         const unsigned long index,
         const t3rt_value_t one_char,
         t3rt_context_t context);
```

Parameters

string	Valid value of one of string types.
index	Zero based position in vector value to assign to.
one_char	Element to assigned to the specified position of string value.

Description

This function operates on charstring, bitstring, octetstring, hexstring and universal charstring types. When applied to charstring and universal charstring values assigned element should be of char or universal char type correspondingly. For other string values assigned element should have the same type as string value but with length equal to 1.

Assigning value to the position outside current string boundaries generates test case error.

Type check for vector value is performed after assignment.

t3rt_value_set_integer

Set a TTCN-3 RTS value from the corresponding C representation.

```
t3rt_value_t t3rt_value_set_integer
        (t3rt_value_t value,
         t3rt_long_integer_t number,
         t3rt_context_t context);
```

Parameters

value	The value to be modified. If the value is not appropriate for this operation a test case error will be generated.
number	The new integer value.

Description

Sets the TTCN-3 RTS value to the given value.

Return Values

The modified value, that is, the same value passed as first argument to the function.

See also

"t3rt_value_get_integer" on page 288

t3rt_value_set_boolean

Set a TTCN-3 RTS value from the corresponding C representation.

```
t3rt_value_t t3rt_value_set_boolean
        (t3rt_value_t boolean_value,
         const bool flag,
         t3rt_context_t context);
```

Parameters

`boolean_value`	The value to be modified. If the value is not appropriate for this operation a test case error will be generated.
`flag`	The new boolean value.

Description

Sets the TTCN-3 RTS value to the given value.

Return Values

The modified value, that is, the same value passed as first argument to the function.

See also

"t3rt_value_get_boolean" on page 290

t3rt_value_set_enum

Set a TTCN-3 RTS value from the corresponding C representation.

```
t3rt_value_t t3rt_value_set_enum
        (t3rt_value_t enum_value,
         t3rt_long_integer_t number,
         t3rt_context_t context);
```

Parameters

`enum_value`	The value to be modified. If the value is not appropriate for this operation a test case error will be generated.
`number`	The new enumeration value.

Description

Sets the TTCN-3 RTS value to the given value.

Return Values

The modified value, that is, the same value passed as first argument to the function.

See also

"t3rt_value_get_enum_number" on page 288

"t3rt_value_get_enum_name" on page 289

t3rt_value_set_float

Set a TTCN-3 RTS value from the corresponding C representation.

```
t3rt_value_t t3rt_value_set_float
        (t3rt_value_t float_value,
         const double number,
         t3rt_context_t context);
```

Parameters

`float_value`	The value to be modified. If the value is not appropriate for this operation a test case error will be generated.
`number`	The new floating-point value.

Description

Sets the TTCN-3 RTS value to the given value.

Return Values

The modified value, that is, the same value passed as first argument to the function.

See also

"t3rt_value_get_float" on page 289

t3rt_value_set_verdict

Set a TTCN-3 RTS value from the corresponding C representation.

```
t3rt_value_t t3rt_value_set_verdict
        (t3rt_value_t verdict_value,
         const t3rt_verdict_t verdict,
         t3rt_context_t context);
```

Parameters

verdict_value	The value to be modified. If the value is not appropriate for this operation a test case error will be generated.
verdict	The new verdict value.

Description

Sets the TTCN-3 RTS value to the given value.

Return Values

The modified value, that is, the same value passed as first argument to the function.

See also

"t3rt_value_get_verdict" on page 293

t3rt_value_set_char

Set a TTCN-3 RTS value from the corresponding C representation.

```
t3rt_value_t t3rt_value_set_char
        (t3rt_value_t char_value,
         const char single_char,
         t3rt_context_t context);
```

Parameters

char_value	The value to be modified. If the value is not appropriate for this operation a test case error will be generated.
single_char	The new character value.

Description

Sets the TTCN-3 RTS value to the given value.

Return Values

The modified value, that is, the same value passed as first argument to the function.

See also

"t3rt_value_get_char" on page 290

t3rt_value_set_string

Set a TTCN-3 RTS value from the corresponding C representation.

```
t3rt_value_t t3rt_value_set_string
        (t3rt_value_t string_value,
         const char *string,
         t3rt_context_t context);
```

Parameters

`string_value`	The value to be modified. If the value is not appropriate for this operation a test case error will be generated.
`string`	The new string value.

Description

Sets the TTCN-3 RTS value to the given value. Assigned octetstring and hexstring values are always converted to the upper case.

Return Values

The modified value, that is, the same value passed as first argument to the function.

See also

"t3rt_value_get_string" on page 291

t3rt_value_set_universal_char

t3rt_value_set_universal_char, t3rt_value_set_universal_char_to_ascii

Set a TTCN-3 RTS value from the corresponding C representation.

```
t3rt_value_t t3rt_value_set_universal_char
        (t3rt_value_t universal_char_value,
         const t3rt_wide_char_t single_wchar,
         t3rt_context_t context);

t3rt_value_t t3rt_value_set_universal_char_to_ascii
        (t3rt_value_t universal_char_value,
         const char single_char,
         t3rt_context_t context);
```

Parameters

`universal_char_value`	The value to be modified. If the value is not appropriate for this operation a test case error will be generated.
`single_wchar`	The new universal character value.
`single_char`	The new char value

Description

Sets the TTCN-3 RTS value to the given value. Two different routines are available to initialize universal character value. It can be filled either from t3rt_wide_char_t value or from ASCII char symbol.

Return Values

The modified value, that is, the same value passed as first argument to the function.

See also

"t3rt_value_get_universal_char" on page 291

t3rt_value_set_universal_charstring

t3rt_value_set_universal_charstring, t3rt_value_set_universal_charstring_to_ascii, t3rt_value_set_universal_charstring_from_wchar_array

Set a TTCN-3 RTS value from the corresponding C representation.

```
t3rt_value_t t3rt_value_set_universal_charstring
        (t3rt_value_t wide_string_value,
         const t3rt_wide_string_t string,
         t3rt_context_t context);

t3rt_value_t t3rt_value_set_universal_charstring_to_ascii
        (t3rt_value_t wide_string_value,
         const char * ascii_data,
         t3rt_context_t context);

t3rt_value_t
t3rt_value_set_universal_charstring_from_wchar_array
        (t3rt_value_t wide_string_value,
         const t3rt_wide_char_t * wchar_data,
         unsigned long length,
         t3rt_context_t context);
```

Parameters

wide_string_value	The value to be modified. If the value is not appropriate for this operation a test case error will be generated.
string	The new string value.
ascii_data	ASCII string
wchar_data	Array of t3rt_wide_char_t elements
length	Number of elements in the “wchar_data” array

Description

Sets the TTCN-3 RTS value to the given value. Three different routines are available to initialize universal character string. It can be filled either from t3rt_wide_string_t value or from array of t3rt_wide_char_t elements (each element representing one symbol) or from ASCII null-terminated character string.

Return Values

The modified value, that is, the same value passed as first argument to the function.

See also

"t3rt_value_get_universal_charstring" on page 292

t3rt_value_set_binary_string

Set a RTS value from the corresponding C representation.

```
t3rt_value_t t3rt_value_set_binary_string
        (t3rt_value_t bstring_value,
         const t3rt_binary_string_t data,
         t3rt_context_t context);
```

Parameters

`binary_string_value`	The value to be modified. If the value is not appropriate for this operation a test case error will be generated.
`data`	The new data.

Description

Sets the TTCN-3 RTS value to the given value.

Return Values

The modified value, that is, the same value passed as first argument to the function.

See also

"t3rt_value_get_verdict" on page 293

t3rt_value_set_address_value

t3rt_value_add_vector_element

Add a new value element to a recordof or setof typed value.

```
t3rt_value_t t3rt_value_add_vector_element
        (t3rt_value_t value,
         const t3rt_value_t element,
         t3rt_context_t context);
```

Parameters

value	The vector value
element	The element to add.

Description

This appends a (copy of a) value to a recordof or setof value.

Return Values

The modified value, that is, the same value passed as first argument to the function.

See also

"t3rt_value_assign_vector_element" on page 295

"t3rt_value_remove_vector_element" on page 305

"t3rt_value_vector_element" on page 285

t3rt_value_remove_vector_element

Remove an element from a recordof or setof typed value.

```
void t3rt_value_remove_vector_element
        (t3rt_value_t value,
         const unsigned long index,
         t3rt_context_t context);
```

Parameters

value	The list value to remove an element from.
index	The index of the element to remove.

Description

Removes an element from the recordof or setof value and decreases the length with 1.

See also

"t3rt_value_add_vector_element" on page 304

"t3rt_value_assign_vector_element" on page 295

"t3rt_value_vector_element" on page 285

t3rt_value_add_objectid_element

Add a number to the end of the object identifier value.

```
t3rt_value_t t3rt_value_add_objectid_element
        (t3rt_value_t objid_value,
         unsigned long element,
         t3rt_context_t context);
```

Parameters

objid_value	The objectidentifier value to be modified. If the value is not appropriate for this operation a test case error will be generated.
element	Integer to be added to the end of objectidentifier list.

Description

This function added specified integer to the end of objectidentifier list.

Return Values

Returns modified object identifier value passed as first parameter.

t3rt_value_set_omit

Set an optional field of a record or set as omitted.

```
void t3rt_value_set_omit
        (t3rt_value_t value,
```

```
        unsigned long field_index,
        t3rt_context_t context);
```

Parameters

value	The record or set value to be modified. If the value is not appropriate for this operation a test case error will be generated.
field_index	The field to be set as omitted.

Description

Changes the status of a record or set field to be omitted. To check if a field is omitted, use “t3rt_ispresent” on page 308.

t3rt_verdict_string

Convert a verdict value to its name.

```
const char *t3rt_verdict_string(t3rt_verdict_t verdict)
```

Parameters

verdict	The verdict to be converted. See type for constants.

Description

Primitive function that converts from verdict constant to string representation. Intended to be used for logging purposes.

For example, for the t3rt_verdict_pass_c constant, the string “pass” will be returned.

Return Values

String representation of a verdict constant.

t3rt_value_check

Checks that the value fulfills its type restrictions.

```
bool t3rt_value_check
```

```
        (t3rt_value_t value,
         t3rt_context_t context);
```

Description

Verifies that the value fulfills the restrictions of its own type. Calls the t3rt_type_check function.

Return Values

True if the value fulfills it own type, otherwise false.

RTL Predefined Operations Functions

t3rt_ispresent

Checks if the indicated field is present in the record or set value (that is, not omitted).

```
bool t3rt_ispresent
        (t3rt_value_t record_value,
         unsigned long field_index,
         t3rt_context_t context);
```

Parameters

record_value	A fully instantiated record or set value.
field_index	The index of the field to be checked. Indexes starts from 0.

Description

This is only applicable to record and set values.

Return Values

Returns true if the field is omitted, false otherwise.

t3rt_ischosen

Checks if the indicated field is present in the union value.

```
bool t3rt_ischosen
        (t3rt_value_t union_value,
         unsigned long alt_index,
         t3rt_context_t context);
```

Parameters

union_value	A fully instantiated union value.
alt_index	A valid index (according to the type) for a union selection.

Description

This is only applicable to union values and checks whether a given field is selected/chosen in the instantiated union value.

If only the type field name is available, use "t3rt_type_field_index" on page 256 to convert the field name to an index and then use this as argument to this function.

Return Values

Returns true if the alternative is selected, false otherwise.

t3rt_concatenate

Returns a newly created string value that contains a concatenated string.

```
t3rt_value_t t3rt_concatenate
        (t3rt_value_t string1,
         t3rt_value_t string2,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

string1	First (left) part of concatenation.
string2	Second (right) part of the concatenation.
strategy	Memory allocation strategy for the result string

Description

The requirement for concatenation is that the two strings have to be of the same type kind. The type of the resulting value is based on this type kind. So, if two hexstrings are concatenated and they are of type T1 and T2 respectively, the result will be a string of the type hexstring.

Return Values

Returns concatenation of the given strings allocated to the specified memory allocation strategy.

t3rt_is_equal

Checks whether two values are equal.

```
bool t3rt_is_equal
        (t3rt_value_t value1,
         t3rt_value_t value2,
         t3rt_context_t ctx);
```

Parameters

value1	One value descriptor.
value2	Another value descriptor.

Description

Both values have to be fully initialized otherwise test case error is generated. It's not necessary for compared values to have one and the same type however they should be of one and the same value kind, i.e. they should have one and the same base type.

Return Values

Returns true is values are equal, false otherwise.

t3rt_is_greater

Checks whether first value is greater than second.

```
bool t3rt_is_greater
        (t3rt_value_t value1,
         t3rt_value_t value2,
```

```
        t3rt_context_t ctx);
```

Parameters

value1	One value descriptor.
value2	Another value descriptor.

Description

This function may be used to compare two values. It returns true if first value is greater than the second one. This function is applicable only to objectidentifer values.

Return Values

Returns true if first value is greater than the second, false otherwise.

t3rt_is_lesser

Checks whether first value is lesser than second.

```
bool t3rt_is_lesser
        (t3rt_value_t value1,
         t3rt_value_t value2,
         t3rt_context_t ctx);
```

Parameters

value1	One value descriptor.
value2	Another value descriptor.

Description

This function may be used to compare two values. It returns true if first value is lesser than the second one. This function is applicable only to objectidentifer values.

Return Values

Returns true if first value is lesserthan the second, false otherwise.

t3rt_not4b

Returns a copy of the operand on which the predefined operation has been applied.

```
t3rt_value_t t3rt_not4b
        (t3rt_value_t string,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

`string`	The string operand.
`strategy`	Memory allocation strategy for the resulting value.

Description

This performs the not4b operation according to ETSI ES 201 873-1 V2.2.1.

This is applicable to string value of type `bitstring`, `hexstring` and `octetstring`.

Return Values

A copy of the resulting value, allocated according to the given strategy.

t3rt_and4b

Returns a copy of the operand on which the predefined operation has been applied.

```
t3rt_value_t t3rt_and4b
        (t3rt_value_t string1,
         t3rt_value_t string2,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

`string1`	One string operand.
`string2`	The other string operand.
`strategy`	Memory allocation strategy for the resulting value.

Description

This performs the and4b operation according to ETSI ES 201 873-1 V2.2.1.

This is applicable to string value of type `bitstring`, `hexstring` and `octetstring`.

Return Values

A copy of the resulting value, allocated according to the given strategy.

t3rt_or4b

Returns a copy of the operand on which the predefined operation has been applied.

```
t3rt_value_t t3rt_or4b
        (t3rt_value_t string1,
         t3rt_value_t string2,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

`string1`	One string operand.
`string2`	The other string operand.
`strategy`	Memory allocation strategy for the resulting value.

Description

This performs the or4b operation according to ETSI ES 201 873-1 V2.2.1.

This is applicable to string value of type `bitstring`, `hexstring` and `octetstring`.

Return Values

A copy of the resulting value, allocated according to the given strategy.

t3rt_xor4b

Returns a copy of the operand on which the predefined operation has been applied.

```
t3rt_value_t t3rt_xor4b
        (t3rt_value_t string1,
         t3rt_value_t string2,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

string1	One string operand.
string2	The other string operand.
strategy	Memory allocation strategy for the resulting value.

Description

This performs the xor4b operation according to ETSI ES 201 873-1 V2.2.1.

This is applicable to string value of type `bitstring`, `hexstring` and `octetstring`.

Return Values

A copy of the resulting value, allocated according to the given strategy.

t3rt_rotateleft

Performs a rotation operation on a copy of a string operand.

```
t3rt_value_t t3rt_rotateleft
        (t3rt_value_t string,
         unsigned long count,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

string	The string to rotate.
count	The number of rotations.
strategy	Memory allocation strategy for the resulting value.

Description

Rotates the string element in the string according to ETSI ES 201 873-1 V2.2.1.

Return Values

A copy of the rotated string allocated with the specified allocation strategy.

t3rt_rotateright

Performs a rotation operation on a copy of a string operand.

```
t3rt_value_t t3rt_rotateright
        (t3rt_value_t string,
         unsigned long count,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

`string`	The string to rotate.
`count`	The number of rotations.
`strategy`	Memory allocation strategy for the resulting value.

Description

Rotates the string element in the string according to ETSI ES 201 873-1 V2.2.1.

Return Values

A copy of the rotated string allocated with the specified allocation strategy.

t3rt_shiftleft

Shift a string a number of elements to the left.

```
t3rt_value_t t3rt_shiftleft
        (t3rt_value_t string,
         unsigned long count,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

`string`	The string to shift.
`count`	The number of elements to shift.
`strategy`	Memory allocation strategy for the resulting value.

Description

This produces a copy of the operand that is shifted the given number of element.

Return Values

A copy of the resulting string according to the specified allocation strategy.

t3rt_shiftright

Shift a string a number of elements.

```
t3rt_value_t t3rt_shiftright
        (t3rt_value_t string,
         unsigned long count,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

`string`	The string to shift.
`count`	The number of elements to shift.
`strategy`	Memory allocation strategy for the resulting value.

Description

This produces a copy of the operand that is shifted the given number of element.

Return Values

A copy of the resulting string according to the specified allocation strategy.

t3rt_bit2int

Predefined conversion function.

```
t3rt_value_t t3rt_bit2int
        (t3rt_value_t bitstring_value,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

bitstring_value	Value to be converted. If the value is not appropriate for the conversion operation, a test case error will be generated.
strategy	Memory allocation strategy for the resulting value.

Description

Converts a value of one kind to a value of another kind. Runtime system doesn't support integer values wider than 64-bit representation thus it's possible to get overflow when using this function. Integer overflow during conversion results in runtime error.

Return Values

The converted value allocated according to the specified strategy.

t3rt_hex2int

Predefined conversion function.

```
t3rt_value_t t3rt_hex2int
        (t3rt_value_t hexstring_value,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

hexstring_value	Value to be converted. If the value is not appropriate for the conversion operation, a test case error will be generated.
strategy	Memory allocation strategy for the resulting value.

Description

Converts a value of one kind to a value of another kind. Runtime system doesn't support integer values wider than 64-bit representation thus it's possible to get overflow when using this function. Integer overflow during conversion results in runtime error.

Return Values

The converted value allocated according to the specified strategy.

t3rt_oct2int

Predefined conversion function.

```
t3rt_value_t t3rt_oct2int
        (t3rt_value_t octetstring_value,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

`octetstring_value`	Value to be converted. If the value is not appropriate for the conversion operation, a test case error will be generated.
`strategy`	Memory allocation strategy for the resulting value.

Description

Converts a value of one kind to a value of another kind. Runtime system doesn't support integer values wider than 64-bit representation thus it's possible to get overflow when using this function. Integer overflow during conversion results in runtime error.

Return Values

The converted value allocated according to the specified strategy.

t3rt_str2int

Predefined conversion function.

```
t3rt_value_t t3rt_str2int
```

```
        (t3rt_value_t charstring_value,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

charstring_value	Value to be converted. If the value is not appropriate for the conversion operation, a test case error will be generated.
strategy	Memory allocation strategy for the resulting value.

Description

Converts a value of one kind to a value of another kind. Runtime system doesn't support integer values wider than 64-bit representation thus it's possible to get overflow when using this function. Integer overflow during conversion results in runtime error.

Return Values

The converted value allocated according to the specified strategy.

t3rt_str2float

Predefined conversion function.

```
t3rt_value_t t3rt_str2float
        (t3rt_value_t charstring_value,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

charstring_value	Value to be converted. If the value is not appropriate for the conversion operation, a test case error will be generated.
strategy	Memory allocation strategy for the resulting value.

Description

Converts a value of one kind to a value of another kind.

Return Values

The converted value allocated according to the specified strategy.

t3rt_char2int

Predefined conversion function.

```
t3rt_value_t t3rt_char2int
        (t3rt_value_t char_value,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

`char_value`	Value to be converted. If the value is not appropriate for the conversion operation, a test case error will be generated.
`strategy`	Memory allocation strategy for the resulting value.

Description

Converts a value of one kind to a value of another kind. This function converts given character into its ASCII character code.

Return Values

The converted value allocated according to the specified strategy.

t3rt_unichar2int

Predefined conversion function.

```
t3rt_value_t t3rt_unichar2int
        (t3rt_value_t wide_char_value,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

wide_char_value	Value to be converted. If the value is not appropriate for the conversion operation, a test case error will be generated.
strategy	Memory allocation strategy for the resulting value.

Description

Converts a value of one kind to a value of another kind. This function converts wide character representation (i.e. byte array) into an integer value. This function calls t3rt_wchar2int.

Return Values

The converted value allocated according to the specified strategy.

t3rt_int2bit

Predefined conversion function.

```
t3rt_value_t t3rt_int2bit
        (t3rt_value_t int_value,
         unsigned long length,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

int_value	Value to be converted. If the value is not appropriate for the conversion operation, a test case error will be generated.
length	Length of the resulting bitstring.
strategy	Memory allocation strategy for the resulting value.

Description

Converts a value of one kind to a value of another kind. Specified length should be large enough to receive all bits of given integer value. If length is greater than necessary to store binary representation of given integer then resulting bitstring is padded with zeros.

Return Values

The converted value allocated according to the specified strategy.

t3rt_int2hex

Predefined conversion function.

```
t3rt_value_t t3rt_int2hex
        (t3rt_value_t int_value,
         unsigned long length,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

`int_value`	Value to be converted. If the value is not appropriate for the conversion operation, a test case error will be generated.
`length`	The length of the resulting string containing the converted integer value.
`strategy`	Memory allocation strategy for the resulting value.

Description

Converts a value of one kind to a value of another kind. Specified length should be large enough to receive all hex chars of given integer value. If length is greater than necessary to store hexadecimal representation of given integer then resulting hexstring is padded with zeros.

Return Values

The converted value allocated according to the specified strategy.

t3rt_int2oct

Predefined conversion function.

```
t3rt_value_t t3rt_int2oct
        (t3rt_value_t int_value,
         unsigned long length,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

int_value	Value to be converted. If the value is not appropriate for the conversion operation, a test case error will be generated.
length	The length of the resulting string containing the converted integer value.
strategy	Memory allocation strategy for the resulting value.

Description

Converts a value of one kind to a value of another kind. Length is measured in octets, i.e. it should be twice smaller than the character length of octet representation. Specified length should be large enough to receive all octets of given integer value. If length is greater than necessary to store octet representation of given integer then resulting octetstring is padded with zeros

Return Values

The converted value allocated according to the specified strategy.

t3rt_int2str

Predefined conversion function.

```
t3rt_value_t t3rt_int2str
        (t3rt_value_t int_value,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

int_value	Value to be converted. If the value is not appropriate for the conversion operation, a test case error will be generated.
strategy	Memory allocation strategy for the resulting value.

Description

Converts a value of one kind to a value of another kind.

Return Values

The converted value allocated according to the specified strategy.

t3rt_int2char

Predefined conversion function.

```
t3rt_value_t t3rt_int2char
        (t3rt_value_t int_value,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

`int_value`	Value to be converted. If the value is not appropriate for the conversion operation, a test case error will be generated.
`strategy`	Memory allocation strategy for the resulting value.

Description

Converts a value of one kind to a value of another kind. It's assumed that given integer value represents 7-bit ASCII code of a certain character. Character code should be in range from 0 to 127, otherwise test case error is generated.

Return Values

The converted value allocated according to the specified strategy.

t3rt_int2unichar

Predefined conversion function.

```
t3rt_value_t t3rt_int2unichar
        (t3rt_value_t int_value,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

int_value	Value to be converted. If the value is not appropriate for the conversion operation, a test case error will be generated.
strategy	Memory allocation strategy for the resulting value.

Description

Converts a value of one kind to a value of another kind. This function converts integer into a wide char representation (i.e. byte array). This function calls t3rt_int2wchar.

Return Values

The converted value allocated according to the specified strategy.

t3rt_bit2str

Predefined conversion function.

```
t3rt_value_t t3rt_bit2str
        (t3rt_value_t bitstring,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

bitstring	Value to be converted. If the value is not appropriate for the conversion operation, a test case error will be generated.
strategy	Memory allocation strategy for the resulting value.

Description

Converts a value of one kind to a value of another kind. This function converts bitstring into its text representation, e.g. '1110101'B => "1110101".

Return Values

The converted value allocated according to the specified strategy.

t3rt_hex2str

Predefined conversion function.

```
t3rt_value_t t3rt_hex2str
        (t3rt_value_t hexstring,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

hexstring	Value to be converted. If the value is not appropriate for the conversion operation, a test case error will be generated.
strategy	Memory allocation strategy for the resulting value.

Description

Converts a value of one kind to a value of another kind. This function converts hexstring into its text representation, e.g. '78ADF'H => "78ADF".

Return Values

The converted value allocated according to the specified strategy.

t3rt_oct2str

Predefined conversion function.

```
t3rt_value_t t3rt_oct2str
        (t3rt_value_t octeststring,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

octetstring	Value to be converted. If the value is not appropriate for the conversion operation, a test case error will be generated.
strategy	Memory allocation strategy for the resulting value.

Description

Converts a value of one kind to a value of another kind. This function converts octetstring into its text representation, e.g. '7788'O => "7788".

Return Values

The converted value allocated according to the specified strategy.

t3rt_str2oct

Predefined conversion function.

```
t3rt_value_t t3rt_str2oct
        (t3rt_value_t charstring,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

`charstring`	Value to be converted. If the value is not appropriate for the conversion operation, a test case error will be generated.
`strategy`	Memory allocation strategy for the resulting value.

Description

Converts a value of one kind to a value of another kind. This function converts octetstring represented by its text representation into octetstring, e.g. “7788” => ‘7788’O.

Return Values

The converted value allocated according to the specified strategy.

t3rt_oct2char

Predefined conversion function.

```
t3rt_value_t t3rt_oct2char
        (t3rt_value_t octetstring,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

octetstring	Value to be converted. If the value is not appropriate for the conversion operation, a test case error will be generated.
strategy	Memory allocation strategy for the resulting value.

Description

The input parameter invalue shall not contain octet values higher than 7F. The resulting charstring shall have the same length as the input octetstring. The octets are interpreted as ISO/IEC 646 codes (according to the IRV) and the resulting characters are appended to the returned value..

Return Values

The converted value allocated according to the specified strategy.

t3rt_char2oct

Predefined conversion function.

```
t3rt_value_t t3rt_char2oct
        (t3rt_value_t charstring,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

charstring	Value to be converted. If the value is not appropriate for the conversion operation, a test case error will be generated.
strategy	Memory allocation strategy for the resulting value.

Description

Converts a value of one kind to a value of another kind. Each octet of the octetstring will contain the ISO/IEC 646 codes (according to the IRV) of the appropriate characters of invalue, e.g. “Tinky-Winky" -> '54696E6B792D57696E6B79'O

Return Values

The converted value allocated according to the specified strategy.

t3rt_bit2hex

Predefined conversion function.

```
t3rt_value_t t3rt_bit2hex
        (t3rt_value_t bitstring,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

bitstring	Value to be converted. If the value is not appropriate for the conversion operation, a test case error will be generated.
strategy	Memory allocation strategy for the resulting value.

Description

Converts a value of one kind to a value of another kind. This function converts bitstring into hexstring, e.g. ‘111010’B=> ‘3A’H.

Return Values

The converted value allocated according to the specified strategy.

t3rt_hex2oct

Predefined conversion function.

```
t3rt_value_t t3rt_hex2oct
        (t3rt_value_t hexstring,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

hexstring	Value to be converted. If the value is not appropriate for the conversion operation, a test case error will be generated.
strategy	Memory allocation strategy for the resulting value.

Description

Converts a value of one kind to a value of another kind. This function converts hexstring into octetstring. If the length of given hexstring is odd then its padded wit zero character, e.g. ‘ABC’H=> ‘0ABC’O.

Return Values

The converted value allocated according to the specified strategy.

t3rt_bit2oct

Predefined conversion function.

```
t3rt_value_t t3rt_bit2oct
        (t3rt_value_t bitstring,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

`bitstring`	Value to be converted. If the value is not appropriate for the conversion operation, a test case error will be generated.
`strategy`	Memory allocation strategy for the resulting value.

Description

Converts a value of one kind to a value of another kind. This function converts bitstring into octetstring, e.g. ‘101100111010’B=> ‘0B3A’O.

Return Values

The converted value allocated according to the specified strategy.

t3rt_hex2bit

Predefined conversion function.

```
t3rt_value_t t3rt_hex2bit
        (t3rt_value_t hexstring,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

`hexstring`	Value to be converted. If the value is not appropriate for the conversion operation, a test case error will be generated.
`strategy`	Memory allocation strategy for the resulting value.

Description

Converts a value of one kind to a value of another kind. This function converts hexstring into bitstring, e.g. ‘3A’H =>‘111010’B.

Return Values

The converted value allocated according to the specified strategy.

t3rt_oct2hex

Predefined conversion function.

```
t3rt_value_t t3rt_oct2hex
        (t3rt_value_t octetstring,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

`octetstring`	Value to be converted. If the value is not appropriate for the conversion operation, a test case error will be generated.
`strategy`	Memory allocation strategy for the resulting value.

Description

Converts a value of one kind to a value of another kind. This function converts octetstring into hexstring, e.g., ‘0ABC’H=> ‘0ABC’H.

Return Values

The converted value allocated according to the specified strategy.

t3rt_oct2bit

Predefined conversion function.

```
t3rt_value_t t3rt_oct2bit
        (t3rt_value_t octetstring,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

octetstring	Value to be converted. If the value is not appropriate for the conversion operation, a test case error will be generated.
strategy	Memory allocation strategy for the resulting value.

Description

Converts a value of one kind to a value of another kind. This function converts octetstring into bitstring, e.g. ‘0B3A’O=>‘101100111010’B .

Return Values

The converted value allocated according to the specified strategy.

t3rt_int2float

Predefined conversion function.

```
t3rt_value_t t3rt_int2float
        (t3rt_value_t int_value,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

int_value	Value to be converted. If the value is not appropriate for the conversion operation, a test case error will be generated.
strategy	Memory allocation strategy for the resulting value.

Description

Converts a value of one kind to a value of another kind. This function converts integer into float, e.g. 123=>123.0 .

Return Values

The converted value allocated according to the specified strategy.

t3rt_float2int

Predefined conversion function.

```
t3rt_value_t t3rt_float2int
        (t3rt_value_t float_value,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

int_value	Value to be converted. If the value is not appropriate for the conversion operation, a test case error will be generated.
strategy	Memory allocation strategy for the resulting value.

Description

Converts a value of one kind to a value of another kind. This function converts float into integer. Using this function may result in data loss since all fractional digits are thrown away, e.g. 123.78=>123. No rounding is performed.

Return Values

The converted value allocated according to the specified strategy.

t3rt_rnd

This is the direct mapping of the TTCN-3 “rnd“ function.

```
t3rt_value_t t3rt_rnd
        (t3rt_value_t seed_value,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

seed_value	Float type seed value for random value generator.
strategy	Memory allocation strategy for the resulting value.

Description

This function generates pseudo random float type value in the range from 0 to 1. Use “seed_value” parameter to initialize random value generator. You may pass t3rt_no_value_c as a seed value thus telling runtime system to use internal seed value.

Return Values

Random float value in the range from 0 to 1.

t3rt_decomp

This is the direct mapping of TTCN-3 “decomp“ function.

```
t3rt_value_t t3rt_decomp
        (t3rt_value_t objid_value,
         unsigned long index,
         unsigned long return_count,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

objid_value	Valid value of objectidentifier type.
index	Index of the first element to be extracted.
return_count	Number of elements to be extracted.
strategy	Memory allocation strategy for the resulting value.

Description

This function operates on objectidentifier type values. Given value should be fully initialized. Index of the first and last element of the extracted objectidentifier should be within the length of the given value.

Return Values

Objectidentifier value representing the part of the provided value.

t3rt_substr

This is the direct mapping of TTCN-3 “substr“ function.

```
t3rt_value_t t3rt_substr
        (t3rt_value_t string_value,
         unsigned long index,
         unsigned long return_count,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

string_value	Valid value of one of string types.
index	Index of the first element to be extracted.
return_count	Number of elements to be extracted.
strategy	Memory allocation strategy for the resulting value.

Description

This function operates on charstring, octetstring, bitstring, hexstring and universal charstring type values. Given string value should be fully initialized. Index of the first and last element of the extracted substring should be within the length of the given string value. Note that index and the length parameters for a octetstring are given in elements (not in characters).

Return Values

Substring of the given string.

t3rt_replace

This is the direct mapping of TTCN-3 “replace“ function.

```
t3rt_value_t t3rt_replace
        (t3rt_value_t string_value,
         unsigned long index,
         unsigned long return_count,
         t3rt_value_t str_replace_with,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

string_value	Valid value of one of string types.
index	Index of the first element to be extracted.
return_count	Number of elements to be extracted.
str_replace_with	Valid value of one of string types. The type should be compatible with the type of string_value parameter.
strategy	Memory allocation strategy for the resulting value.

Description

This function operates on charstring, octetstring, bitstring, hexstring and universal charstring type values. Given string value should be fully initialized. Index of the first and last element of the replaced substring should be within the length of the given string value. Note that index and the length parameters for a octetstring are given in elements (not in characters). The types of the first and forth parameters should be compatible otherwise testcase error is generated.

Return Values

New string with changed content.

t3rt_lengthof

This is the direct mapping of TTCN-3 “lengthof” function.

```
unsigned long t3rt_lengthof
        (t3rt_value_t string,
         t3rt_context_t ctx);
```

Parameters

string	Valid value of one of string types.

Description

This function operates on charstring, octetstring, bitstring, hexstring and universal charstring type values. It simply calls t3rt_value_string_length function.

Return Values

Length of the given string measured in elements.

t3rt_sizeof

This is the direct mapping of TTCN-3 “sizeof” function.

```
unsigned long t3rt_sizeof
        (t3rt_value_t value,
         t3rt_context_t ctx);
```

Parameters

value	Valid value of one of the vector types including templates and object identifiers.

Description

This function operates on array, recordof, setof, record, set, objectidentifier, signature and template values. For template values, this function returns sizeof (valueof(value)), if valueof(value) is defined. When applied to record and set values this function counts only defined values, i.e. optional values explicitly set to omit are not considered.

Return Values

Actual number of elements in the given value.

t3rt_sizeoftype

This is the direct mapping of TTCN-3 “sizeoftype” function.

```
unsigned long t3rt_sizeoftype
        (t3rt_value_t value,
         t3rt_context_t ctx);
```

Parameters

value	Valid value of recordof, setof or array type or template of one of these types.

Description

This function operates on array, recordof, setof and template values. This function shall be applied to values of types with length restriction. The actual number to be returned is the sequential number of the last element without respect to whether its value is defined or not (i.e. the upper length index of the type definition on which the parameter of the function is based on plus 1).

Return Values

Maximum allowed length for a length restricted type.

t3rt_mod

Calculate the module operation according to ETSI ES 201 873-1 V2.2.1.

```
t3rt_long_integer_t t3rt_mod
        (t3rt_long_integer_t x,
         t3rt_long_integer_t y,
         t3rt_context_t ctx);
```

Parameters

x	First integer operand.
y	Second (non-zero) integer operand.

Description

This function computes the rest that remains from an integer division of x by y. For positive arguments x and y this function behaves similar to t3rt_rem, but the result is different when arguments are negative, e.g. -2 mod 3 = 1.

Return Values

The module value of the operands.

t3rt_rem

Calculate the remainder operation according to ETSI ES 201 873-1 V2.2.1.

```
t3rt_long_integer_t t3rt_rem
        (t3rt_long_integer_t x,
         t3rt_long_integer_t y,
         t3rt_context_t ctx);
```

Parameters

x	First operand.
y	Second (non-zero) operand.

Description

This function computes the rest that remains from an integer division of x by y. For positive arguments x and y this function behaves similar to t3rt_mod, but the result is different when arguments are negative, e.g. -2 rem 3 = -2.

Return Values

The remainder when dividing the operands (x/y).

t3rt_log

Logs the string value on the information log channel as a TTCN-3 message.

```
void t3rt_log
        (t3rt_value_t char_string,
         t3rt_context_t ctx);
```

Parameters

char_string	Valid string value.

Description

This function sends given string to the log channels of all registered log mechanisms. It simply calls t3rt_log_string_to_all for the specified string. The message kind of the logged string is “ttcn-3“.

t3rt_regexp_regexp

This is the direct mapping of the TTCN-3 “regexp” function.

```
t3rt_value_t t3rt_regexp_regexp
        (t3rt_value_t value,
         t3rt_value_t pattern,
         int group_index,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

`value`	Matching value.
`pattern`	Matching pattern.
`group_index`	Zero-based group index.
`strategy`	Memory allocation strategy for the resulting value.

Description

This function matches string against regular expression. It's important to keep in mind that the whole given value is matched against pattern. It means that pattern parameter should specify matching pattern for the whole string, not for searched element (as it may be done in Perl). If pattern declares groups then this function may be used to extract the value of certain group using ordinal zero-based group number. The type of returned value is the same as the type of the given matching value.

Return Values

Returns extracted match substring if match succeeded, empty string ("") otherwise.

RTL Timer Functions

RTL Timer Related Type Definitions

t3rt_timer_handle_t

This is a union type used to store timer handles. It is either an unsigned long or a void*.

t3rt_timer_state_t

This is used for reporting the state of a timer (as in t3pl_timer_read for example). The values can either be t3rt_timer_state_stopped, t3rt_timer_state_running or t3rt_timer_state_timedout.

t3rt_timer_timed_out

Inform the RTS that a timer has timed out.

```
void t3rt_timer_timed_out
        (const t3rt_timer_handle_t handle,
         t3rt_context_t ctx);
```

Parameters

handle	The timer that has timed out.

Description

This function is usually used in integration to notify runtime system that timer specified by its handle has timed out. In “Example“ integration it’s called as a result of invoking triTimeout TRI function.

This function must be called for every timer if test suite has been started with “t3rt.timers.assuming_all_active“ RTConf key enabled. Specifying this RTConf key tells runtime system that all timers in test suite are “active“, i.e. timeout event is generated by the integration. This may be critical for real-time systems.

When “t3rt.timers.assuming_all_active“ RTConf key is not specified runtime system performs evaluation if timeout event for every timer using internal real-time clock. Thus in such case call to t3rt_timer_timed_out (or triTimeout) may be omitted.

RTL Component Functions

t3rt_component_main

Main function for a new component thread.

```
void t3rt_component_main
        (t3rt_binary_string_t control_port_address,
         t3rt_context_t ctx);
```

Parameters

control_port_address	The control port address for this new component. This address may not be NULL.

Description

This function is used in the t3pl_task_create as the main function. It needs the control_port_address to be able to create the context of the new component.

t3rt_component_self

This is the direct mapping of TTCN-3 “self” component reference.

```
t3rt_value_t t3rt_component_self(t3rt_context_t ctx);
```

Description

This function returns reference to the component instance on which this function has been invoked.

Return values

Returns reference to the current component.

t3rt_component_mtc

This is the direct mapping of TTCN-3 “mtc” component reference.

```
t3rt_value_t t3rt_component_mtc(t3rt_context_t ctx);
```

Description

This function returns reference to the main test component instance.

Return values

Returns reference to the MTC component.

t3rt_component_system

This is the direct mapping of TTCN-3 “system” component reference.

```
t3rt_value_t t3rt_component_system(t3rt_context_t ctx);
```

Description

This function returns reference to the system component instance.

Return values

Returns reference to the TSI component.

t3rt_component_set_local_verdict

This is the direct mapping of TTCN-3 “setverdict” statement.

```
void t3rt_component_set_local_verdict
        (t3rt_value_t verdict_value,
         t3rt_context_t ctx);
```

Parameters

verdict_value	Valid verdict value to set.

Description

This function sets changes verdict of a component on which this function has been invoked. It maintains TTCN-3 verdict hierarchy thus attempting to change “inconc“ verdict to “pass“ does nothing. The same is applicable to “error“ verdict.

t3rt_component_get_local_verdict

This is the direct mapping of TTCN-3 “getverdict” statement.

```
t3rt_value_t t3rt_component_get_local_verdict
(t3rt_context_t ctx);
```

Description

This function returns local verdict of a component on which this function has been invoked. Use t3rt_value_get_verdict function to extract actual verdict from the returned value.

Return Values

Returns local component verdict.

t3rt_component_element

Returns indicated component element value.

```
t3rt_value_t t3rt_component_element
        (const char* element,
         t3rt_context_t ctx);
```

Parameters

element	Name of component type field.

Description

This function returns component element value (port record, constant, variable, or timer) of a component on which this function has been invoked. Component value is identified by its name as defined in component type declaration.

The names of component fields may be obtained by processing component type using t3rt_type_field_name function.

Return Values

Returns value of a component field.

t3rt_component_mute

Turns on/off logging of events on the component.

```
void t3rt_component_mute
        (bool on_off,
         t3rt_context_t ctx);
```

Parameters

on_off	Flag signalling new logging state.

Description

This function switches on and off logging of all events on the current component. Component is specified through provided context reference. When logging is switched off all log mechanisms including built-in log stop generating events. This function doesn't have impact on real-time debugger.

RTL Port Functions

t3rt_port_insert_message

Inserts specified data on behalf of sender into the local port input queue.

```
void t3rt_port_insert_message
        (t3rt_binary_string_t port_address,
         t3rt_binary_string_t sut_address,
         t3rt_binary_string_t bstring,
         t3rt_context_t ctx);
```

Parameters

port_address	Receiving port address.
sut_address	Address inside SUT.
bstring	Encoded message.

Description

This function is usually used in integration. It's invoked upon receiving message and adds message to the incoming queue of a specified port. For messages received from SUT it's possible to specify sender address that distinguishes certain SUT entity from all other entities that communicate with test system through this port.

t3rt_port_insert_call

Appends a call event with specified parameters to the queue associated with the port.

```
void t3rt_port_insert_call
        (t3rt_binary_string_t port_address,
         t3rt_binary_string_t sut_address,
         t3rt_type_t signature_type,
         t3rt_binary_string_t parameters[],
         t3rt_binary_string_t sender,
         t3rt_context_t ctx);
```

Parameters

port_address	Receiving port address.
sut_address	Address inside SUT.
signature_type	Signature of the received procedure call.
parameters	Array of encoded actual parameters.
sender	Obsolete, should be NULL.

Description

This function is usually used in integration. It's invoked upon receiving remote procedure call and adds call to the incoming queue of a specified port. For procedure calls received from SUT it's possible to specify sender address that distinguishes certain SUT entity from all other entities that communicate with test system through this port.

t3rt_port_insert_reply

Appends a reply event with specified parameters and return value to the queue associated with the port.

```
void t3rt_port_insert_reply
        (t3rt_binary_string_t port_address,
         t3rt_binary_string_t sut_address,
         t3rt_type_t signature_type,
         t3rt_binary_string_t parameters[],
         t3rt_binary_string_t return_value,
         t3rt_binary_string_t sender,
         t3rt_context_t ctx);
```

Parameters

port_address	Receiving port address.
sut_address	Address inside SUT.
signature_type	Signature of the received procedure reply.
parameters	Array of encoded actual parameters.
return_value	Encoded return value.
sender	Obsolete, should be NULL.

Description

This function is usually used in integration. It's invoked upon receiving reply to a remote procedure call and adds reply to the incoming queue of a specified port. For procedure replies received from SUT it's possible to specify sender address that distinguishes certain SUT entity from all other entities that communicate with test system through this port.

t3rt_port_insert_exception

Appends an exception event with specified data to the queue associated with the port.

```
void t3rt_port_insert_exception
        (t3rt_binary_string_t port_address,
         t3rt_binary_string_t sut_address,
         t3rt_type_t signature_type,
         t3rt_binary_string_t exception_data,
         t3rt_binary_string_t sender,
         t3rt_context_t ctx);
```

Parameters

port_address	Receiving port address.
sut_address	Address inside SUT.
signature_type	Signature of the procedure that raised exception.
exception_data	Encoded exception value.
sender	Obsolete, should be NULL.

Description

This function is usually used in integration. It's invoked upon receiving exception to a remote procedure call and adds exception to the incoming queue of a specified port. For exceptions raised from SUT it's possible to specify sender address that distinguishes certain SUT entity from all other entities that communicate with test system through this port.

RTL Log Functions

This is the functions that handle logging, both from the perspective of logging events and also from the perspective of implementing a log mechanism.

RTL Log Related Type Definitions

t3rt_log_mechanism_init_function_t

A function of this prototype is one of the functions registered for a log mechanism. It will be called once (in each process) for all components and should initialize the log mechanism to a working state.

t3rt_log_mechanism_finalize_function_t

A function of this prototype is one of the functions registered for a log mechanism. It will be called once (in each process) for all components and should make any necessary clean up on the log mechanism level.

t3rt_log_mechanism_open_function_t

A function of this prototype is one of the functions registered for a log mechanism. It will be called once per component with a newly created log instance and has the option of setting any auxiliary data for this particular instance.

t3rt_log_mechanism_close_function_t

A function of this prototype is one of the functions registered for a log mechanism. It will be called once per component with the log instance in question and should make any necessary clean up and closing of this log instance.

t3rt_log_mechanism_log_event_function_t

A function of this prototype is one of the functions registered for a log mechanism. It will be called whenever a log event is generated by the RTS. It should implement the desired filtering of the set of events and take care of the actual log event visualization (printing to standard I/O, for example).

t3rt_log_mechanism_version_t

This represents a version of the log mechanism interface. If this changes, old log mechanisms can still function if they just tell the RTS which version they support. (This is currently not used.)

t3rt_log_message_kind_t

This represents severity of a message logged in the runtime system interface. It should be one of the t3rt_log_predefined_message_kind_t values. These values cover “info”, “ttcn-3“, “warning”, “error“ and “debug“ messages. “ttcn-3” kind messages are the result of TTCN-3 log statement.

t3rt_log_event_kind_t

This represents type of a message logged in the runtime system. It’s passed to the event handler installed with t3rt_log_register_listener every time event is generated. Most of events have special functions that decode array of event parameters and extract certain values relevant to the generated event. Event decoding function is chosen basing on the event kind. Valid event kinds are:

t3rt_log_mechanism_t

This is a descriptor of a log mechanism. Each registered log mechanism is assigned with such descriptor. t3rt_log_mechanism_t object is one and the same for all instances of the log mechanism (i.e. it's a sort of a type). Log mechanism instances are of t3rt_log_t type.

t3rt_log_t

This is a log instance (retrieved from a registered log mechanism) through which logging is channeled. This is used as a handle when giving log events to the log mechanisms. When component is created in the test suite runtime system creates new instances of every registered log mechanism for this component.

t3rt_codecs_strategy_t

This represents encoding (decoding) strategy that has been used by the runtime system to encode (decode) message. There are two possible options: registered (user-provided) codec or built-in codec. t3rt_codecs_strategy_t type is defined as enumeration:

```
typedef enum t3rt__codecs_strategy_internal_t
{
        t3rt_codecs_strategy_registered_c,
        t3rt_codecs_strategy_builtin_c
} t3rt_codecs_strategy_t;
```

Events generated in RTS

RTS generates exhaustive set of events that fully describe runtime behavior of test suite. Some of the events are one-to-one mapping of TTCN-3 operations (e.g. mapping of a port), in other cases one TTCN-3 statement (e.g. "alt" statement) may correspond to several runtime events.

Most of events are augmented with event parameters that are passed to user-defined event handler function as NULL-terminated array of t3rt_value_t objects. If event has parameters then they may be extracted from this array using certain extraction function (e.g. t3rt_log_extract_message_sent).

Message Sent

This event has three kinds `t3rt_log_event_message_sent_c`, `t3rt_log_event_message_sent_mc_c` and `t3rt_log_event_message_sent_bc_c` for unicast, multicast and broadcast operations correspondingly. It's generated as a result of successful

TTCN-3 “send” operation on connected port.Use one of t3rt_log_extract_message_sent, t3rt_log_extract_message_sent_mc or t3rt_log_extract_message_sent_bc functions to extract event parameters.

SUT Message Sent

This event has three kinds `t3rt_log_event_sut_message_sent_c`, `t3rt_log_event_sut_message_sent_mc_c` and `t3rt_log_event_sut_message_sent_bc_c` for unicast, multicast and broadcast operations correspondingly. It’s generated as a result of successful TTCN-3 “send” operation on mapped port. Use one of t3rt_log_extract_message_sent, t3rt_log_extract_message_sent_mc or t3rt_log_extract_message_sent_bc functions to extract event parameters.

Message Sent Failed

This event has three kinds `t3rt_log_event_message_sent_failed_c`, `t3rt_log_event_message_sent_failed_mc_c` and `t3rt_log_event_message_sent_failed_bc_c` for unicast, multicast and broadcast operations correspondingly. It’s generated as a result of failed TTCN-3 “send” operation on connected port due to encoding or transmission error. Use one of t3rt_log_extract_message_sent_failed, t3rt_log_extract_message_sent_failed_mc or t3rt_log_extract_message_sent_failed_bc functions to extract event parameters.

SUT Message Sent Failed

This event has three kinds `t3rt_log_event_message_sut_sent_failed_c`, `t3rt_log_event_message_sut_sent_failed_mc_c` and `t3rt_log_event_message_sut_sent_failed_bc_c` for unicast, multicast and broadcast operations correspondingly. It’s generated as a result of failed TTCN-3 “send” operation on mapped port due to encoding or transmission error. Use one of t3rt_log_extract_message_sent_failed, t3rt_log_extract_message_sent_failed_mc or t3rt_log_extract_message_sent_failed_bc functions to extract event parameters.

Message Detected

This event has kind `t3rt_log_event_message_detected_c`. It's generated when a local message (i.e. not from SUT) is put into component incoming port queue. Use t3rt_log_extract_message_detected function to extract event parameters.

SUT Message Detected

This event has kind `t3rt_log_event_sut_message_detected_c`. It's generated when a message from SUT is put into component incoming port queue. Use t3rt_log_extract_message_detected function to extract event parameters.

Message Received

This event has kind `t3rt_log_event_message_received_c`. It's generated as a result of successful TTCN-3 "receive" operation on connected port. Use t3rt_log_extract_message_received, t3rt_log_extract_message_found functions to extract event parameters.

SUT Message Received

This event has kind `t3rt_log_event_sut_message_received_c`. It's generated as a result of successful TTCN-3 "receive" operation on mapped port. Use t3rt_log_extract_message_received, t3rt_log_extract_message_found functions to extract event parameters.

Message Found

This event has kind `t3rt_log_event_message_found_c`. It's generated as a result of successful TTCN-3 "check(receive)" operation on connected port. Use t3rt_log_extract_message_received, t3rt_log_extract_message_found functions to extract event parameters.

SUT Message Found

This event has kind `t3rt_log_event_sut_message_found_c`. It's generated as a result of successful TTCN-3 "check(receive)" operation on mapped port. Use t3rt_log_extract_message_received, t3rt_log_extract_message_found functions to extract event parameters.

Message Discarded

This event has kind `t3rt_log_event_message_discarded_c`. It's generated as a result of successful TTCN-3 "trigger" operation on connected port. Use t3rt_log_extract_message_discarded function to extract event parameters.

SUT Message Discarded

This event has kind `t3rt_log_event_sut_message_discarded_c`. It's generated as a result of successful TTCN-3 "trigger" operation on mapped port. Use t3rt_log_extract_message_discarded function to extract event parameters.

Call Initiated

This event has three kinds `t3rt_log_event_call_initiated_c`, `t3rt_log_event_call_initiated_mc_c` and `t3rt_log_event_call_initiated_bc_c` for unicast, multicast and broadcast operations correspondingly. It's generated as a result of successful TTCN-3 "call" operation on connected port. Use one of t3rt_log_extract_call_initiated, t3rt_log_extract_call_initiated_mc or t3rt_log_extract_call_initiated_bc functions to extract event parameters.

SUT Call Initiated

This event has three kinds `t3rt_log_event_sut_call_initiated_c`, `t3rt_log_event_sut_call_initiated_mc_c` and `t3rt_log_event_sut_call_initiated_bc_c` for unicast, multicast and broadcast operations correspondingly. It's generated as a result of successful TTCN-3 "call" operation on mapped port. Use one of t3rt_log_extract_call_initiated, t3rt_log_extract_call_initiated_mc or t3rt_log_extract_call_initiated_bc functions to extract event parameters.

Call Failed

This event has three kinds `t3rt_log_event_call_failed_c`, `t3rt_log_event_call_failed_mc_c` and `t3rt_log_event_call_failed_bc_c` for unicast, multicast and broadcast operations correspondingly. It's generated as a result of failed TTCN-3

"call" operation on connected port due to encoding or transmission errors. Use one of t3rt_log_extract_call_failed, t3rt_log_extract_call_failed_mc or t3rt_log_extract_call_failed_bc functions to extract event parameters.

SUT Call Failed

This event has three kinds `t3rt_log_event_sut_call_failed_c`, `t3rt_log_event_sut_call_failed_mc_c` and `t3rt_log_event_sut_call_failed_bc_c` for unicast, multicast and broadcast operations correspondingly. It's generated as a result of failed TTCN-3 "call" operation on mapped port due to encoding or transmission errors. Use one of t3rt_log_extract_call_failed, t3rt_log_extract_call_failed_mc or t3rt_log_extract_call_failed_bc functions to extract event parameters.

Call Timed Out

This event has kind `t3rt_log_event_call_timed_out_c`. It's generated as a result of failed TTCN-3 "call" operation on connected port due to timeout event. Use t3rt_log_extract_call_timed_out function to extract event parameters.

SUT Call Timed Out

This event has kind `t3rt_log_event_sut_call_timed_out_c`. It's generated as a result of failed TTCN-3 "call" operation on mapped port due to timeout event. Use t3rt_log_extract_call_timed_out function to extract event parameters.

Call Detected

This event has kind `t3rt_log_event_call_detected_c`. It's generated when a local (i.e. not from SUT) procedure call request is put into component incoming port queue. Use t3rt_log_extract_call_detected function to extract event parameters.

SUT Call Detected

This event has kind `t3rt_log_event_sut_call_detected_c`. It's generated when a procedure call request from SUT is put into component incoming port queue. Use t3rt_log_extract_call_detected function to extract event parameters.

Call Received

This event has kind `t3rt_log_event_call_received_c`. It's generated as a result of a successful TTCN-3 "getcall" operation on connected port. Use t3rt_log_extract_call_received, t3rt_log_extract_call_found functions to extract event parameters.

SUT Call Received

This event has kind `t3rt_log_event_sut_call_received_c`. It's generated as a result of a successful TTCN-3 "getcall" operation on mapped port. Use t3rt_log_extract_call_received, t3rt_log_extract_call_found functions to extract event parameters.

Call Found

This event has kind `t3rt_log_event_call_found_c`. It's generated as a result of a successful TTCN-3 "check(getcall)" operation on connected port. Use t3rt_log_extract_call_received, t3rt_log_extract_call_found functions to extract event parameters.

SUT Call Found

This event has kind `t3rt_log_event_sut_call_found_c`. It's generated as a result of a successful TTCN-3 "check(getcall)" operation on mapped port. Use t3rt_log_extract_call_received, t3rt_log_extract_call_found functions to extract event parameters.

Reply Sent

This event has three kinds `t3rt_log_event_reply_sent_c`, `t3rt_log_event_reply_sent_mc_c` and `t3rt_log_event_reply_sent_bc_c` for unicast, multicast and broadcast operations correspondingly. It's generated as a result of successful TTCN-3 "reply" operation on connected port.Use one of t3rt_log_extract_reply_sent, t3rt_log_extract_reply_sent_mc or t3rt_log_extract_reply_sent_bc functions to extract event parameters.

SUT Reply Sent

This event has three kinds `t3rt_log_event_sut_reply_sent_c`, `t3rt_log_event_sut_reply_sent_mc_c` and `t3rt_log_event_sut_reply_sent_bc_c` for unicast, multicast and broadcast operations correspondingly. It's generated as a result of successful TTCN-3 "reply" operation on mapped port.Use one of t3rt_log_extract_reply_sent, t3rt_log_extract_reply_sent_mc or t3rt_log_extract_reply_sent_bc functions to extract event parameters.

Reply Failed

This event has three kinds `t3rt_log_event_reply_failed_c`, `t3rt_log_event_reply_failed_mc_c` and `t3rt_log_event_reply_failed_bc_c` for unicast, multicast and broadcast operations correspondingly. It's generated as a result of failed TTCN-3 "reply" operation on connected port due to encoding or transmission errors. Use one of t3rt_log_extract_reply_failed, t3rt_log_extract_reply_failed_mc or t3rt_log_extract_reply_failed_bc functions to extract event parameters.

SUT Reply Failed

This event has three kinds `t3rt_log_event_sut_reply_failed_c`, `t3rt_log_event_sut_reply_failed_mc_c` and `t3rt_log_event_sut_reply_failed_bc_c` for unicast, multicast and broadcast operations correspondingly. It's generated as a result of failed TTCN-3 "reply" operation on mapped port due to encoding or transmission errors. Use one of t3rt_log_extract_reply_failed, t3rt_log_extract_reply_failed_mc or t3rt_log_extract_reply_failed_bc functions to extract event parameters.

Reply Detected

This event has kind `t3rt_log_event_reply_detected_c`. It's generated when a local (i.e. not from SUT) procedure call reply is put into component incoming port queue. Use t3rt_log_extract_reply_detected function to extract event parameters.

SUT Reply Detected

This event has kind `t3rt_log_event_sut_reply_detected_c`. It's generated when a procedure call reply from SUT is put into component incoming port queue. Use t3rt_log_extract_reply_detected function to extract event parameters.

Reply Received

This event has kind `t3rt_log_event_reply_received_c`. It's generated as a result of a successful TTCN-3 "getreply" operation on connected port. Use t3rt_log_extract_reply_received, t3rt_log_extract_reply_found functions to extract event parameters.

SUT Reply Received

This event has kind `t3rt_log_event_sut_reply_received_c`. It's generated as a result of a successful TTCN-3 "getreply" operation on mapped port. Use t3rt_log_extract_reply_received, t3rt_log_extract_reply_found functions to extract event parameters.

Reply Found

This event has kind `t3rt_log_event_reply_found_c`. It's generated as a result of a successful TTCN-3 "check(getreply)" operation on connected port. Use t3rt_log_extract_reply_received, t3rt_log_extract_reply_found functions to extract event parameters.

SUT Reply Found

This event has kind `t3rt_log_event_sut_reply_found_c`. It's generated as a result of a successful TTCN-3 "check(getreply)" operation on mapped port. Use t3rt_log_extract_reply_received, t3rt_log_extract_reply_found functions to extract event parameters.

Exception Raised

This event has three kinds `t3rt_log_event_exception_raised_c`, `t3rt_log_event_exception_raised_mc_c` and `t3rt_log_event_exception_raised_bc_c` for unicast, multicast and broadcast operations correspondingly. It's generated as a result of successful TTCN-3 "raise" operation on connected port. Use one of

t3rt_log_extract_exception_raised, t3rt_log_extract_exception_raised_mc or t3rt_log_extract_exception_raised_bc functions to extract event parameters.

SUT Exception Raised

This event has three kinds `t3rt_log_event_sut_exception_raised_c`, `t3rt_log_event_sut_exception_raised_mc_c` and `t3rt_log_event_sut_exception_raised_bc_c` for unicast, multicast and broadcast operations correspondingly. It's generated as a result of successful TTCN-3 "raise" operation on mapped port.Use one of t3rt_log_extract_exception_raised, t3rt_log_extract_exception_raised_mc or t3rt_log_extract_exception_raised_bc functions to extract event parameters.

Raise Failed

This event has three kinds `t3rt_log_event_raise_failed_c`, `t3rt_log_event_raise_failed_mc_c` and `t3rt_log_event_raise_failed_bc_c` for unicast, multicast and broadcast operations correspondingly. It's generated as a result of failed TTCN-3 "raise" operation on connected port due to encoding or transmission errors. Use one of t3rt_log_extract_raise_failed, t3rt_log_extract_raise_failed_mc or t3rt_log_extract_raise_failed_bc functions to extract event parameters.

SUT Raise Failed

This event has three kinds `t3rt_log_event_sut_raise_failed_c`, `t3rt_log_event_sut_raise_failed_mc_c` and `t3rt_log_event_sut_raise_failed_bc_c` for unicast, multicast and broadcast operations correspondingly. It's generated as a result of failed TTCN-3 "raise" operation on mapped port due to encoding or transmission errors. Use one of t3rt_log_extract_raise_failed, t3rt_log_extract_raise_failed_mc or t3rt_log_extract_raise_failed_bc functions to extract event parameters.

Exception Detected

This event has kind `t3rt_log_event_exception_detected_c`. It's generated when a local (i.e. not in SUT) procedure call exception is put into component incoming port queue. Use t3rt_log_extract_exception_detected function to extract event parameters.

SUT Exception Detected

This event has kind `t3rt_log_event_sut_exception_detected_c`. It's generated when a SUT procedure call exception is put into component incoming port queue. Use t3rt_log_extract_exception_detected function to extract event parameters.

Exception Caught

This event has kind `t3rt_log_event_exception_caught_c`. It's generated as a result of a successful TTCN-3 "catch" operation on connected port. Use t3rt_log_extract_exception_caught, t3rt_log_extract_exception_found functions to extract event parameters.

SUT Exception Caught

This event has kind `t3rt_log_event_sut_exception_caught_c`. It's generated as a result of a successful TTCN-3 "catch" operation on mapped port. Use t3rt_log_extract_exception_caught, t3rt_log_extract_exception_found functions to extract event parameters.

Exception Found

This event has kind `t3rt_log_event_exception_found_c`. It's generated as a result of a successful TTCN-3 "check(catch)" operation on connected port. Use t3rt_log_extract_exception_caught, t3rt_log_extract_exception_found functions to extract event parameters.

SUT Exception Found

This event has kind `t3rt_log_event_sut_exception_found_c`. It's generated as a result of a successful TTCN-3 "check(catch)" operation on mapped port. Use t3rt_log_extract_exception_caught, t3rt_log_extract_exception_found functions to extract event parameters.

Timeout Exception Detected

This event has kind `t3rt_log_event_timeout_exception_detected_c`. It's generated when a local (i.e. not in SUT) procedure call timeout exception is put into component incoming port queue. Use t3rt_log_extract_timeout_exception_detected function to extract event parameters.

SUT Timeout Exception Detected

This event has kind `t3rt_log_event_sut_timeout_exception_detected_c`. It's generated when a SUT procedure call timeout exception is put into component incoming port queue. Use t3rt_log_extract_timeout_exception_detected function to extract event parameters.

Timeout Exception Caught

This event has kind `t3rt_log_event_timeout_exception_caught_c`. It's generated as a result of a successful TTCN-3 "catch(timeout)" operation on connected port. Use t3rt_log_extract_timeout_exception_caught, t3rt_log_extract_timeout_exception_found functions to extract event parameters.

SUT Timeout Exception Caught

This event has kind `t3rt_log_event_sut_timeout_exception_caught_c`. It's generated as a result of a successful TTCN-3 "catch(timeout)" operation on mapped port. Use t3rt_log_extract_timeout_exception_caught, t3rt_log_extract_timeout_exception_found functions to extract event parameters.

Timeout Exception Found

This event has kind `t3rt_log_event_timeout_exception_found_c`. It's generated as a result of a successful TTCN-3 "check(catch(timeout))" operation on connected port. Use t3rt_log_extract_timeout_exception_caught, t3rt_log_extract_timeout_exception_found functions to extract event parameters.

SUT Timeout Exception Found

This event has kind `t3rt_log_event_sut_timeout_exception_found_c`. It's generated as a result of a successful TTCN-3 "check(catch(timeout))" operation on mapped port. Use t3rt_log_extract_timeout_exception_caught, t3rt_log_extract_timeout_exception_found functions to extract event parameters.

SUT Action Performed

This event has kind `t3rt_log_event_sut_action_performed_c`. It's generated as a result of TTCN-3 "action" operation. Use t3rt_log_extract_sut_action function to extract event parameters.

Timer Started

This event has kind `t3rt_log_event_timer_started_c`. It's generated as a result of TTCN-3 "start" timer operation. Use t3rt_log_extract_timer_started function to extract event parameters.

Timer Stopped

This event has kind `t3rt_log_event_timer_stopped_c`. It's generated as a result of TTCN-3 "stop" timer operation. Use t3rt_log_extract_timer_stopped function to extract event parameters.

Timer Read

This event has kind `t3rt_log_event_timer_read_c`. It's generated as a result of TTCN-3 "read" timer operation. Use t3rt_log_extract_timer_read function to extract event parameters.

Timer Is Running Check Performed

This event has kind `t3rt_log_event_timer_is_running_c`. It's generated as a result of TTCN-3 "running" timer operation. Use t3rt_log_extract_timer_is_running function to extract event parameters.

Timer Timeout Detected

This event has kind `t3rt_log_event_timeout_detected_c`. It's generated when RTS is notified about timer timeout be means of triTimeout operation. Use t3rt_log_extract_timeout_detected function to extract event parameters.

Timer Timed Out Check Succeeded

This event has kind `t3rt_log_event_timeout_received_c`. It's generated when timer timed out alternative matches. Use t3rt_log_extract_timeout_received function to extract event parameters.

Timer Timed Out Check Failed

This event has kind `t3rt_log_event_timeout_mismatch_c`. It's generated each time timer timed out alternative fails to match. Use t3rt_log_extract_timeout_mismatch function to extract event parameters.

Component Created

This event has kind `t3rt_log_event_component_created_c`. It's generated as a result of TTCN-3 "create" and "execute" component operations. Use t3rt_log_extract_component_created function to extract event parameters.

Component Started

This event has kind `t3rt_log_event_component_started_c`. It's generated as a result of TTCN-3 "start" and "execute" component operations. Use t3rt_log_extract_component_started function to extract event parameters.

Component Is Running Check Performed

This event has kind `t3rt_log_event_component_is_running_c`. It's generated as a result of TTCN-3 "running" component operation. Use t3rt_log_extract_component_is_running function to extract event parameters.

Component Is Alive Check Performed

This event has kind `t3rt_log_event_component_is_alive_c`. It's generated as a result of TTCN-3 "alive" component operation. Use t3rt_log_extract_component_is_alive function to extract event parameters.

Component Stopped

This event has kind `t3rt_log_event_component_stopped_c`. It's generated when component terminates. Use t3rt_log_extract_component_stopped function to extract event parameters.

Component Killed

This event has kind `t3rt_log_event_component_killed_c`. It's generated when alive component terminates. Use t3rt_log_extract_component_killed function to extract event parameters.

Component Terminated

This event has kind `t3rt_log_event_component_terminated_c`. It's not generated yet.

Component Done Check Succeeded

This event has kind `t3rt_log_event_done_check_succeeded_c`. It's generated when component done alternative matches. Use t3rt_log_extract_done_check_succeeded function to extract event parameters.

Component Done Check Failed

This event has kind `t3rt_log_event_done_check_failed_c`. It's generated when component done alternative fails to match. Use t3rt_log_extract_done_check_failed function to extract event parameters.

Component Killed Check Succeeded

This event has kind `t3rt_log_event_kill_check_succeeded_c`. It's generated when component killed alternative matches. Use t3rt_log_extract_kill_check_succeeded function to extract event parameters.

Component Killed Check Failed

This event has kind `t3rt_log_event_kill_check_failed_c`. It's generated when component killed alternative fails to match. Use t3rt_log_extract_kill_check_failed function to extract event parameters.

Port Connected

This event has kind `t3rt_log_event_port_connected_c`. It's generated as a result of TTCN-3 "connect" port operation. Use t3rt_log_extract_port_connected function to extract event parameters.

Port Disconnected

This event has kind `t3rt_log_event_port_disconnected_c`. It's generated as a result of TTCN-3 "disconnect" port operation. Use t3rt_log_extract_port_disconnected function to extract event parameters.

Port Mapped

This event has kind `t3rt_log_event_port_mapped_c`. It's generated as a result of TTCN-3 "map" port operation. Use t3rt_log_extract_port_mapped function to extract event parameters.

Port Unmapped

This event has kind `t3rt_log_event_port_unmapped_c`. It's generated as a result of TTCN-3 "unmap" port operation. Use t3rt_log_extract_port_unmapped function to extract event parameters.

Port Enabled

This event has kind `t3rt_log_event_port_enabled_c`. It's generated as a result of TTCN-3 "start" port operation. Use t3rt_log_extract_port_enabled function to extract event parameters.

Port Disabled

This event has kind `t3rt_log_event_port_disabled_c`. It's generated as a result of TTCN-3 "stop" port operation. Use t3rt_log_extract_port_disabled function to extract event parameters.

Port Halted

This event has kind `t3rt_log_event_port_halted_c`. It's generated as a result of TTCN-3 "halt" port operation. Use t3rt_log_extract_port_halted function to extract event parameters.

Port Cleared

This event has kind `t3rt_log_event_port_cleared_c`. It's generated as a result of TTCN-3 "clear" port operation. Use t3rt_log_extract_port_cleared function to extract event parameters.

Scope Entered

This event has kind `t3rt_log_event_scope_entered_c`. It's generated when execution control enters new scope (e.g. function, testcase, altstep or control part). This event is also generated for module initialization and finalization functions. One of the event parameters (see t3rt_scope_kind_t) may be used to obtain the type of entered scope. Use t3rt_log_extract_scope_entered function to extract event parameters.

Scope Changed

This event has kind `t3rt_log_event_scope_changed_c`. It's generated when TTCN-3 source location changes, i.e. next TTCN-3 statement is going to be executed. This event is generated only when test suite is running under TTCN-3 debugger. Use t3rt_log_extract_scope_changed function to extract event parameters.

Scope Left

This event has kind `t3rt_log_event_scope_left_c`. It's generated when execution control leaves scope (e.g. function, testcase, altstep or control part). Use t3rt_log_extract_scope_left function to extract event parameters.

Alternative Activated

This event has kind `t3rt_log_event_alternative_activated_c`. It's generated as a result of TTCN-3 "activate" operation. Use t3rt_log_extract_alternative_activated_event function to extract event parameters.

Alternative Deactivated

This event has kind `t3rt_log_event_alternative_deactivated_c`. It's generated as a result of TTCN-3 "deactivate" operation. Use t3rt_log_extract_alternative_deactivated_event function to extract event parameters.

Local Verdict Set

This event has kind `t3rt_log_event_local_verdict_changed_c`. It's generated as a result of TTCN-3 "setverdict" operation. This event is generated also implicitly by RTS when "error" verdict is set due to runtime error

or overall test case verdict is changed. In later case event is generated for CPC (control component). Use t3rt_log_extract_local_verdict_changed function to extract event parameters.

Local Verdict Read

This event has kind `t3rt_log_event_local_verdict_queried_c`. It's generated as a result of TTCN-3 "getverdict" operation. Use t3rt_log_extract_local_verdict_queried function to extract event parameters.

Variable Modified

This event has kind `t3rt_log_event_variable_modified_c`. It's generated whenever any variable, constant or module parameter (either whole value or some of its elements) is assigned with value. Use t3rt_log_extract_variable_modified function to extract event parameters.

Function called

This event has kind `t3rt_log_event_function_call_c`. It's generated whenever function is invoked. Use t3rt_log_extract_function_call function to extract event parameters.

External Function Called

This event has kind `t3rt_log_event_external_function_call_c`. It's generated whenever external function is invoked. Use t3rt_log_extract_external_function_call function to extract event parameters.

Altstep Called

This event has kind `t3rt_log_event_altstep_call_c`. It's generated whenever altstep is directly invoked. Use t3rt_log_extract_altstep_call function to extract event parameters.

Template Match Failed

This event has kind `t3rt_log_event_template_match_failed_c`. It's generated whenever matching of a value against template fails. This may be the result of mismatching alternative in "alt" statement or a mismatch in di-

rectly called “match“ operation. Use t3rt_log_extract_template_match_failed function to extract event parameters.

Template Match Begin

This event has kind `t3rt_log_event_template_match_begin_c`. It’s generated whenever matching of a subtemplate inside a structured template begins. Use t3rt_log_extract_template_match_begin function to extract event parameters.

Template Match End

This event has kind `t3rt_log_event_template_match_end_c`. It’s generated whenever matching of a subtemplate inside a structured template ends. Use t3rt_log_extract_template_match_end function to extract event parameters.

Template Mismatch

This event has kind `t3rt_log_event_template_match_begin_c`. It’s generated whenever matching of a template or a subtemplate inside a structured template fails. Use t3rt_log_extract_template_mismatch function to extract event parameters.

Test case started

This event has kind `t3rt_log_event_testcase_started_c`. It’s generated whenever test case starts (e.g. as a result of TTCN-3 “execute“ operation). Use t3rt_log_extract_testcase_started function to extract event parameters.

Test case ended

This event has kind `t3rt_log_event_testcase_ended_c`. It’s generated whenever test case terminates . Use t3rt_log_extract_testcase_ended function to extract event parameters.

Test case timed out

This event has kind `t3rt_log_event_testcase_timed_out_c`. It's generated whenever test case times out. Use t3rt_log_extract_testcase_timed_out function to extract event parameters.

Test case verdict

This event has kind `t3rt_log_event_testcase_verdict_c`. It's generated whenever test case terminates. This event is obsolete and will be removed in future. Listen to "test case ended" event instead of it. Use t3rt_log_extract_test_case_verdict function to extract event parameters.

Test case error

This event has kind `t3rt_log_event_testcase_error_c`. It's generated whenever test case error is signalled. Use t3rt_log_extract_testcase_error function to extract event parameters.

Information Message

This event has kind `t3rt_log_event_info_message_c`. It's generated whenever information message is sent to registered log mechanisms. Use t3rt_log_extract_text_message_string or t3rt_log_extract_text_message_widestring functions to extract event parameters.

Warning Message

This event has kind `t3rt_log_event_warning_message_c`. It's generated whenever warning message is sent to registered log mechanisms. Use t3rt_log_extract_text_message_string or t3rt_log_extract_text_message_widestring functions to extract event parameters.

Error Message

This event has kind `t3rt_log_event_error_message_c`. It's generated whenever error message is sent to registered log mechanisms. Usually this event is followed by "test case error" event. Use

t3rt_log_extract_text_message_string or t3rt_log_extract_text_message_widestring functions to extract event parameters.

Debug Message

This event has kind `t3rt_log_event_debug_message_c`. It's generated whenever debug message is sent to registered log mechanisms. Use t3rt_log_extract_text_message_string or t3rt_log_extract_text_message_widestring functions to extract event parameters.

TTCN-3 Message

This event has kind `t3rt_log_event_ttcn3_message_c`. It's generated whenever TTCN-3 message is sent to registered log mechanisms. Usually this event is the result of the TTCN-3 log statement. Use t3rt_log_extract_text_message_string or t3rt_log_extract_text_message_widestring functions to extract event parameters.

Data Encoded

This event has kind `t3rt_log_event_message_encoded_c`. It's generated to log successful encoding of a value into binary string. Use t3rt_log_extract_message_encoded function to extract event parameters.

Data Encoding Failed

This event has kind `t3rt_log_event_message_encode_failed_c`. It's generated to log failure while encoding a value into binary string. Use t3rt_log_extract_message_encode_failed function to extract event parameters.

Data Decoded

This event has kind `t3rt_log_event_message_decoded_c`. It's generated to log successful decoding of a binary string. Use t3rt_log_extract_message_decoded function to extract event parameters.

Data Decoding Failed

This event has kind `t3rt_log_event_message_decode_failed_c`. It's generated to log failure while decoding binary string. Use t3rt_log_extract_message_decode_failed function to extract event parameters.

Alt Statement Entered

This event has kind `t3rt_log_event_alt_entered_c`. It's generated when execution controls reaches "alt" statement. No event parameters are associated with this event.

Alt Statement Left

This event has kind `t3rt_log_event_alt_left_c`. It's generated when execution controls leaves "alt" statement. No event parameters are associated with this event.

Alternative Rejected

This event has kind `t3rt_log_event_alt_rejected_c`. It's generated when guard expression evaluates to false thus skipping matching of guarded alternative. No event parameters are associated with this event.

Else Alternative Entered

This event has kind `t3rt_log_event_alt_else_c`. It's generated when execution control enters statement block of "else" alternative. No event parameters are associated with this event.

Defaults Processing Started

This event has kind `t3rt_log_event_alt_defaults_c`. It's generated to log starting execution of defaults. No event parameters are associated with this event.

Repeat Encountered

This event has kind `t3rt_log_event_alt_repeat_c`. It's generated whenever execution controls encounters "repeat" statement. No event parameters are associated with this event.

Alt Statement Waits New Events

This event has kind `t3rt_log_event_alt_wait_c`. It's generated whenever execution controls reaches end of "alt" statement without successful matching of any alternative. Thus component execution is suspended until new events occur. No event parameters are associated with this event.

Sender Mismatch

This event has kind `t3rt_log_event_sender_mismatch_c`. It's generated as a result of a failed TTCN-3 "receive", "getcall", "getreply" or "catch" operation due to sender mismatch. It means that alternative mismatched because actual sender of an operation doesn't match expected one. Use t3rt_log_extract_sender_mismatch function to extract event parameters.

RTS Log Handling Functions

t3rt_log_register_listener

Register a new log mechanism.

```
void t3rt_log_register_listener
        (const char * mechanism_name,
         t3rt_log_mechanism_version_t version,
         t3rt_log_mechanism_init_function_t init_func,
         t3rt_log_mechanism_finalize_function_t final_func,
         t3rt_log_mechanism_open_function_t open_func,
         t3rt_log_mechanism_close_function_t close_func,
         t3rt_log_mechanism_log_event_function_t log_func);
```

Parameters

mechanism_name	Log mechanism name.
version	Log mechanism version.
init_func	Initializing function.
final_func	Finalizing function.
open_func	Opening function.
close_func	Closing function.
log_func	Event handling function.

Description

This function registers the log mechanism to listen to the event channel. All functions except event handling one may be NULL. The name of the log mechanism is used to uniquely identify it inside RTS. This is necessary in order to send event to certain log mechanism (see t3rt_log_event). The version supported by the mechanism should be stated using the version constants.

Initializing function is called once during the initialization of RTS. Finalizing function is called once during the finalization of the RTS and it doesn't have t3rt_context_t parameter. Opening function is called once for every created component during the component initialization. Closing function is called once for every component during the component termination.

This function does not have a t3rt_context_t parameter.

t3rt_log_mechanism_set_auxiliary

Associates user-defined untyped buffer with the log mechanism.

```
void t3rt_log_mechanism_set_auxiliary
        (t3rt_log_mechanism_t log_mechanism,
         void * aux,
         t3rt_context_t context);
```

Parameters

log_mechanism	Log mechanism descriptor.
aux	Pointer to log mechanism auxiliary buffer.

Description

This function allows associating any kind of user defined data with the log mechanism. Data buffer is shared between all instances of the log mechanism. It means that each component may get pointer to this buffer using t3rt_log_mechanism_get_auxiliary function.

Note

Ensure that access to this buffer is serialized if test suite creates parallel components. Since components execute concurrently access to this buffer outside critical section may result in unpredictable behavior.

t3rt_log_mechanism_get_auxiliary

Retrieves user-defined untyped buffer from the given log mechanism.

```
void * t3rt_log_mechanism_get_auxiliary
        (t3rt_log_mechanism_t log_mechanism,
         t3rt_context_t context);
```

Parameters

log_mechanism	Log mechanism descriptor.

Description

This function returns pointer to the log mechanism auxiliary data buffer previously set by t3rt_log_mechanism_set_auxiliary function.

Return Values

Returns pointer to the auxiliary log mechanism data buffer. NULL if non is set.

t3rt_log_set_auxiliary

Associates user-defined untyped buffer with the log instance.

```
void t3rt_log_set_auxiliary
        (t3rt_log_t log_instance,
         void * aux,
         t3rt_context_t context);
```

Parameters

log_instance	Log mechanism instance descriptor.
aux	Pointer to log mechanism auxiliary buffer.

Description

This function allows associating any kind of user defined data with the instance of the log mechanism. This buffer is private to the executing component. It may be queried using t3rt_log_get_auxiliary function.

t3rt_log_get_auxiliary

Retrieves user-defined untyped buffer from the given the log instance.

```
void * t3rt_log_get_auxiliary
        (t3rt_log_t log_instance,
         t3rt_context_t context);
```

Parameters

log_instance	Log mechanism instance descriptor.

Description

This function returns pointer to the log instance auxiliary data buffer previously set by t3rt_log_set_auxiliary function.

Return Values

Returns pointer to the auxiliary log instance data buffer. NULL if non is set.

t3rt_log_get_log_mechanism

Retrieves log mechanism descriptor for the given log instance.

```
t3rt_log_mechanism_t t3rt_log_get_log_mechanism
        (t3rt_log_t log,
         t3rt_context_t context);
```

Parameters

log_instance	Log mechanism instance descriptor.

Return Values

Returns log mechanism descriptor for the given log instance.

t3rt_log_message_kind_name

Returns message kind (severity) as an ASCII string.

```
const char*
t3rt_log_message_kind_name(t3rt_log_message_kind_t kind)
```

Parameters

kind	Message kind.

Return Values

Returns string representation for the given message kind.

t3rt_log_is_concentrator

Obsolete function. Should not be used.

bool t3rt_log_is_concentrator(t3rt_context_t ctx)

t3rt_log_string

Logs string to the specified log mechanism.

```
void t3rt_log_string
        (const char* dest,
         t3rt_log_message_kind_t msg_kind,
         const char *string,
         t3rt_context_t ctx);
```

Parameters

`dest`	Name of destination log mechanism.
`msg_kind`	Message kind.
`string`	ASCII string to log.

Description

This function logs the ASCII string message into event stream. Depending on message kind Information Message, Warning Message, Error Message, Debug Message or TTCN-3 Message event is logged.

The destination is the (registered) name of the log mechanism to pass the string to. The t3rt_log_all_mechanisms_c constant may be used to pass the string to all mechanisms.

t3rt_log_string_to_all

Logs string to all listening log mechanisms.

```
void t3rt_log_string_to_all
        (t3rt_log_message_kind_t msg_kind,
         const char *string,
         t3rt_context_t ctx);
```

Parameters

`msg_kind`	Message kind.
`string`	ASCII string to log.

Description

This function logs the ASCII string message into event stream. Depending on message kind Information Message, Warning Message, Error Message, Debug Message or TTCN-3 Message event is logged.

Message is received by all log mechanisms registered in the runtime system.

t3rt_log_wide_string

Logs wide string to the specified log mechanism.

```
void t3rt_log_wide_string
        (const char * dest,
         t3rt_log_message_kind_t msg_kind,
         t3rt_wide_string_t string,
         t3rt_context_t ctx);
```

Parameters

dest	Name of destination log mechanism.
msg_kind	Message kind.
string	Wide string to log.

Description

This function logs the wide (possibly internationalized) string message into event stream. Depending on message kind Information Message, Warning Message, Error Message, Debug Message or TTCN-3 Message event is logged.

The destination is the (registered) name of the log mechanism to pass the string to. The t3rt_log_all_mechanisms_c constant may be used to pass the string to all mechanisms.

t3rt_log_wide_string_to_all

```
void t3rt_log_wide_string_to_all
        (t3rt_log_message_kind_t msg_kind,
         t3rt_wide_string_t string,
         t3rt_context_t ctx);
```

Parameters

msg_kind	Message kind.
string	Wide string to log.

Description

This function logs the wide (possibly internationalized) string message into event stream. Depending on message kind Information Message, Warning Message, Error Message, Debug Message or TTCN-3 Message event is logged.

Message is received by all log mechanisms registered in the runtime system.

t3rt_log_event

Logs the event to the event log channel of specified log mechanism.

```
void t3rt_log_event
        (const char * dest,
         t3rt_log_event_kind_t event_kind,
         t3rt_value_t params[],
         t3rt_context_t ctx);
```

Parameters

dest	Name of destination log mechanism.
event_kind	Event kind.
params	NULL terminated array of event parameters.

Description

This function logs event into event stream. Each event is identified by the event kind (see t3rt_log_event_kind_t).

Event parameters should be specified as NULL terminated array of t3rt_value_t values.

The destination is the (registered) name of the log mechanism to pass the string to. The t3rt_log_all_mechanisms_c constant may be used to pass the event to all mechanisms.

t3rt_log_event_to_all

Logs the event to the event log channel of all registered log mechanisms.

```
void t3rt_log_event_to_all
        (t3rt_log_event_kind_t event_kind,
         t3rt_value_t params[],
         t3rt_context_t ctx);
```

Parameters

event_kind	Event kind.
params	NULL terminated array of event parameters.

Description

This function logs event into event stream. Each event is identified by the event kind (see t3rt_log_event_kind_t).

Event parameters should be specified as NULL terminated array of t3rt_value_t values.

Message is received by all log mechanisms registered in the runtime system.

t3rt_log_event_kind_string

Returns a textual representation of the log event kind.

```
const char *
t3rt_log_event_kind_string(t3rt_log_event_kind_t event);
```

Parameters

event_kind	Event kind.

Description

Returns the t3rt_log_unknown_event_kind_name_c string constant if the event kind cannot be identified.

Return Values

Returns string representation for the given event kind.

t3rt_log_extract_message_sent

Decode parameters of Message Sent and SUT Message Sent events.

```
void t3rt_log_extract_message_sent
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_value_t *template_message,
         t3rt_value_t *sent_message,
```

```
        t3rt_binary_string_t *encoded_msg,
        unsigned long *seq_no,
        t3rt_binary_string_t *destination_comp_address,
        t3rt_binary_string_t *destination_port_address,
        t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
local_port_name	Sending port name.
local_port_address	Sending port address.
template_message	Template value for the sent message.
sent_message	Actual value sent.
encoded_msg	Encoded message.
seq_no	Unique message number.
destination_comp_address	Receiving component address (SUT address for SUT messages).
destination_port_address	Receiving port address.

Description

t3rt_log_extract_message_sent extracts information describing a successfull unicast "send" TTCN-3 statement. This function is available for user-defined log mechanisms.

t3rt_log_extract_message_sent_mc

Decode parameters of Message Sent and SUT Message Sent events.

```
void t3rt_log_extract_message_sent_mc
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_value_t *template_message,
         t3rt_value_t *sent_message,
         t3rt_binary_string_t *encoded_msg,
         unsigned long *seq_no,
         t3rt_binary_string_t **destination_comp_addr_list,
         t3rt_binary_string_t **destination_port_addr_list,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
local_port_name	Sending port name.
local_port_address	Sending port address.
template_message	Template value for the sent message.
sent_message	Actual value sent.
encoded_msg	Encoded message.
seq_no	Unique message number.
destination_comp_addr_list	NULL-terminated list of receiving components addresses (SUT address for SUT messages).
destination_port_addr_list	NULL-terminated list of receiving ports addresses.

Description

t3rt_log_extract_message_sent_mc extracts information describing a successfull multicast "send" TTCN-3 statement. This function is available for user-defined log mechanisms.

t3rt_log_extract_message_sent_bc

Decode parameters of Message Sent and SUT Message Sent events.

```
void t3rt_log_extract_message_sent_bc
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_value_t *template_message,
         t3rt_value_t *sent_message,
         t3rt_binary_string_t *encoded_msg,
         unsigned long *seq_no,
         t3rt_context_t ctx);
```

Parameters

`params`	Array of event parameters, received from RTS.
`local_port_name`	Sending port name.
`local_port_address`	Sending port address.
`template_message`	Template value for the sent message.
`sent_message`	Actual value sent.
`encoded_msg`	Encoded message.
`seq_no`	Unique message number.

Description

`t3rt_log_extract_message_sent_bc` extracts information describing a successfull broadcast "send" TTCN-3 statement. This function is available for user-defined log mechanisms.

t3rt_log_extract_message_sent_failed

Decode parameters of Message Sent Failed and SUT Message Sent Failed events.

```
void t3rt_log_extract_message_sent_failed
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_value_t *template_message,
         t3rt_value_t *sent_message,
         t3rt_binary_string_t *encoded_msg,
         unsigned long *seq_no,
         t3rt_binary_string_t *destination_comp_address,
         t3rt_binary_string_t *destination_port_address,
         bool *codec_status,
         bool *communication_status,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
local_port_name	Sending port name.
local_port_address	Sending port address.
template_message	Template value for the sent message.
sent_message	Actual value sent.
encoded_msg	Encoded message.
seq_no	Unique message number.
destination_comp_address	Receiving component address (SUT address for SUT messages).
destination_port_address	Receiving port address.
codec_status	Status of encoding operation.
communication_status	Status of transmission operation.

Description

t3rt_log_extract_message_sent_failed extracts information describing a failed unicast "send" TTCN-3 statement. This function is available for user-defined log mechanisms.

t3rt_log_extract_message_sent_failed_mc

Decode parameters of Message Sent Failed and SUT Message Sent Failed events.

```
void t3rt_log_extract_message_sent_failed_mc
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_value_t *template_message,
         t3rt_value_t *sent_message,
         t3rt_binary_string_t *encoded_msg,
         unsigned long *seq_no,
         t3rt_binary_string_t **destination_comp_addr_list,
         t3rt_binary_string_t **destination_port_addr_list,
         bool *codec_status,
         bool *communication_status,
```

```
        t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
local_port_name	Sending port name.
local_port_address	Sending port address.
template_message	Template value for the sent message.
sent_message	Actual value sent.
encoded_msg	Encoded message.
seq_no	Unique message number.
destination_comp_addr_list	NULL-terminated list of receiving components addresses (SUT address for SUT messages).
destination_port_addr_list	NULL-terminated list of receiving ports addresses.
codec_status	Status of encoding operation.
communication_status	Status of transmission operation.

Description

t3rt_log_extract_message_sent_failed_mc extracts information describing a failed multicast "send" TTCN-3 statement. This function is available for user-defined log mechanisms.

t3rt_log_extract_message_sent_failed_bc

Decode parameters of Message Sent Failed and SUT Message Sent Failed events.

```
void t3rt_log_extract_message_sent_failed_bc
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_value_t *template_message,
         t3rt_value_t *sent_message,
         t3rt_binary_string_t *encoded_msg,
         unsigned long *seq_no,
         bool *codec_status,
         bool *communication_status,
```

```
        t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
local_port_name	Sending port name.
local_port_address	Sending port address.
template_message	Template value for the sent message.
sent_message	Actual value sent.
encoded_msg	Encoded message.
seq_no	Unique message number.
codec_status	Status of encoding operation.
communication_status	Status of transmission operation.

Description

t3rt_log_extract_message_sent_failed_bc extracts information describing a failed broadcast "send" TTCN-3 statement. This function is available for user-defined log mechanisms.

t3rt_log_extract_message_detected

Decode parameters of Message Detected and SUT Message Detected events.

```
void t3rt_log_extract_message_detected
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_binary_string_t *detected_data,
         unsigned long *seq_no,
         t3rt_binary_string_t *sender_address,
         t3rt_context_t ctx);
```

Parameters

`params`	Array of event parameters, received from RTS.
`local_port_name`	Receiving port name.
`local_port_address`	Receiving port address.
`detected_data`	Encoded message.
`seq_no`	Unique message number.
`sender_address`	Address of the sending component (SUT address for SUT messages).

Description

`t3rt_log_extract_message_detected` extracts information describing a message (received by the integration) inserted into port queue. This function is available for user-defined log mechanisms.

t3rt_log_extract_message_received, t3rt_log_extract_message_found

Decode parameters of Message Received, SUT Message Received, Message Found and SUT Message Found events.

```
void t3rt_log_extract_message_received
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_value_t *template_value,
         t3rt_value_t *received_value,
         t3rt_binary_string_t *encoded_msg,
         unsigned long *seq_no,
         t3rt_binary_string_t *sender_address,
         t3rt_context_t ctx);

void t3rt_log_extract_message_found
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_value_t *template_value,
         t3rt_value_t *received_value,
         t3rt_binary_string_t *encoded_msg,
         unsigned long *seq_no,
         t3rt_binary_string_t *sender_address,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
local_port_name	Receiving port name.
local_port_address	Receiving port address.
template_value	Template value for the received message.
received_value	Actual value received.
encoded_msg	Encoded message.
seq_no	Unique message number.
sender_address	Address of the sending component (SUT address for SUT messages).

Description

t3rt_log_extract_message_received extracts information describing TTCN-3 “receive“ statement. t3rt_log_extract_message_found extracts information describing TTCN-3 “check(receive)“ statement. These functions are available for user-defined log mechanisms.

t3rt_log_extract_message_discarded

Decode parameters of Message Discarded and SUT Message Discarded events.

```
void t3rt_log_extract_message_discarded
        (t3rt_value_t params[],
        const char **local_port_name,
        t3rt_binary_string_t *local_port_address,
        t3rt_value_t *template_discarded,
        t3rt_value_t *discarded_value,
        unsigned long *seq_no,
        t3rt_binary_string_t *sender_address,
        t3rt_context_t ctx);
```

Parameters

`params`	Array of event parameters, received from RTS.
`local_port_name`	Receiving port name.
`local_port_address`	Receiving port address.
`template_discarded`	Template value for the discarded message.
`discarded_value`	Actual value discarded.
`seq_no`	Unique message number.
`sender_address`	Address of the sending component (SUT address for SUT messages).

Description

`t3rt_log_extract_message_discarded` extracts information describing "trigger" TTCN-3 statement. This function is available for user-defined log mechanisms.

t3rt_log_extract_call_initiated

Decode parameters of Call Initiated and SUT Call Initiated events.

```
void t3rt_log_extract_call_initiated
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_value_t *template_call,
         t3rt_value_t *call_value,
         unsigned long *seq_no,
         t3rt_binary_string_t *destination_comp_address,
         t3rt_binary_string_t *destination_port_address,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
local_port_name	Sending port name.
local_port_address	Sending port address.
template_call	Template value for the called signature.
call_value	Actual signature value called.
seq_no	Unique call number.
destination_comp_address	Receiving component address (SUT address for SUT calls).
destination_port_address	Receiving port address.

Description

t3rt_log_extract_call_initiated extracts information describing successfull unicast "call" TTCN-3 statement. This function is available for user-defined log mechanisms.

t3rt_log_extract_call_initiated_mc

Decode parameters of Call Initiated and SUT Call Initiated events.

```
void t3rt_log_extract_call_initiated_mc
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_value_t *template_call,
         t3rt_value_t *call_value,
         unsigned long *seq_no,
         t3rt_binary_string_t **destination_comp_addr_list,
         t3rt_binary_string_t **destination_port_addr_list,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
local_port_name	Sending port name.
local_port_address	Sending port address.
template_call	Template value for the called signature.
call_value	Actual signature value called.
seq_no	Unique call number.
destination_comp_addr_list	NULL-terminated list of receiving components addresses (SUT address for SUT calls).
destination_port_addr_list	NULL-terminated list of receiving ports addresses.

Description

t3rt_log_extract_call_initiated_mc extracts information describing successfull multicast "call" TTCN-3 statement. This function is available for user-defined log mechanisms.

t3rt_log_extract_call_initiated_bc

Decode parameters of Call Initiated and SUT Call Initiated events.

```
void t3rt_log_extract_call_initiated_bc
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_value_t *template_call,
         t3rt_value_t *call_value,
         unsigned long *seq_no,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
local_port_name	Sending port name.
local_port_address	Sending port address.
template_call	Template value for the called signature.
call_value	Actual signature value called.
seq_no	Unique call number.

Description

t3rt_log_extract_call_initiated_bc extracts information describing successfull broadcast "call" TTCN-3 statement. This function is available for user-defined log mechanisms.

t3rt_log_extract_call_failed

Decode parameters of Call Failed and SUT Call Failed events.

```
void t3rt_log_extract_call_failed
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_value_t *template_call,
         t3rt_value_t *call_value,
         unsigned long *seq_no,
         t3rt_binary_string_t *destination_comp_address,
         t3rt_binary_string_t *destination_port_address,
         bool *codec_status,
         bool *communication_status,
         t3rt_context_t ctx);
```

Parameters

`params`	Array of event parameters, received from RTS.
`local_port_name`	Sending port name.
`local_port_address`	Sending port address.
`template_call`	Template value for the called signature.
`call_value`	Actual signature value called.
`seq_no`	Unique call number.
`destination_comp_address`	Receiving component address (SUT address for SUT calls).
`destination_port_address`	Receiving port address.
`codec_status`	Status of encoding operation.
`communication_status`	Status of transmission operation.

Description

`t3rt_log_extract_call_failed` extracts information describing failed unicast "call" TTCN-3 statement. This function is available for user-defined log mechanisms.

t3rt_log_extract_call_failed_mc

Decode parameters of Call Failed and SUT Call Failed events.

```
void t3rt_log_extract_call_failed_mc
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_value_t *template_call,
         t3rt_value_t *call_value,
         unsigned long *seq_no,
         t3rt_binary_string_t **destination_comp_addr_list,
         t3rt_binary_string_t **destination_port_addr_list,
         bool *codec_status,
         bool *communication_status,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
local_port_name	Sending port name.
local_port_address	Sending port address.
template_call	Template value for the called signature.
call_value	Actual signature value called.
seq_no	Unique call number.
destination_comp_addr_list	NULL-terminated list of receiving components addresses (SUT address for SUT calls).
destination_port_addr_list	NULL-terminated list of receiving ports addresses.
codec_status	Status of encoding operation.
communication_status	Status of transmission operation.

Description

`t3rt_log_extract_call_failed_mc` extracts information describing failed multicast "call" TTCN-3 statement. This function is available for user-defined log mechanisms.

t3rt_log_extract_call_failed_bc

Decode parameters of Call Failed and SUT Call Failed events.

```
void t3rt_log_extract_call_failed_bc
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_value_t *template_call,
         t3rt_value_t *call_value,
         unsigned long *seq_no,
         bool *codec_status,
         bool *communication_status,
         t3rt_context_t ctx);
```

Parameters

`params`	Array of event parameters, received from RTS.
`local_port_name`	Sending port name.
`local_port_address`	Sending port address.
`template_call`	Template value for the called signature.
`call_value`	Actual signature value called.
`seq_no`	Unique call number.
`codec_status`	Status of encoding operation.
`communication_status`	Status of transmission operation.

Description

`t3rt_log_extract_call_failed_bc` extracts information describing failed broadcast "call" TTCN-3 statement. This function is available for user-defined log mechanisms.

t3rt_log_extract_call_timed_out

Decode parameters of Call Timed Out and SUT Call Timed Out events.

```
void t3rt_log_extract_call_timed_out
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_context_t ctx);
```

Parameters

`params`	Array of event parameters, received from RTS.
`local_port_name`	Sending port name.
`local_port_address`	Sending port address.

Description

`t3rt_log_extract_call_timed_out` extracts information describing timed out "call" TTCN-3 statement. This function is available for user-defined log mechanisms.

t3rt_log_extract_call_detected

Decode parameters of Call Detected and SUT Call Detected events.

```
void t3rt_log_extract_call_detected
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_binary_string_t *sender_address,
         unsigned long *seq_no,
         t3rt_context_t ctx);
```

Parameters

`params`	Array of event parameters, received from RTS.
`local_port_name`	Receiving port name.
`local_port_address`	Receiving port address.
`sender_address`	Sending component address (SUT address for SUT calls).

Description

`t3rt_log_extract_call_detected` extracts information describing a procedure call (received by the integration) inserted into port queue. This function is available for user-defined log mechanisms.

t3rt_log_extract_call_received, t3rt_log_extract_call_found

Decode parameters of Call Received, SUT Call Received, Call Found and SUT Call Found events.

```
void t3rt_log_extract_call_received
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_value_t *template_call,
         t3rt_value_t *call_value,
         unsigned long *seq_no,
         t3rt_binary_string_t *destination_comp_address,
         t3rt_context_t ctx);

void t3rt_log_extract_call_found
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_value_t *template_call,
         t3rt_value_t *call_value,
         unsigned long *seq_no,
         t3rt_binary_string_t *destination_comp_address,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
local_port_name	Receiving port name.
local_port_address	Receiving port address.
template_call	Template value for the called signature.
call_value	Actual signature value called.
seq_no	Unique call number.
destination_comp_address	Receiving component address (SUT address for SUT calls).

Description

t3rt_log_extract_call_received extracts information describing TTCN-3 “getcall“ statement. t3rt_log_extract_call_found extracts information describing TTCN-3 “check(getcall)“ statement. These functions are available for user-defined log mechanisms.

t3rt_log_extract_reply_sent

Decode parameters of Reply Sent and SUT Reply Sent events.

```
void
t3rt_log_extract_reply_sent
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_value_t *template_reply,
         t3rt_value_t *reply_value,
         t3rt_value_t *template_return,
         t3rt_value_t *return_value,
         unsigned long *seq_no,
         t3rt_binary_string_t *destination_comp_address,
         t3rt_binary_string_t *destination_port_address,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
local_port_name	Sending port name.
local_port_address	Sending port address.
template_reply	Template value for the reply signature.
reply_value	Actual reply signature value.
template_return	Template for the returned value.
return_value	Actual value returned.
seq_no	Unique reply number.
destination_comp_address	Receiving component address (SUT address for SUT replies).
destination_port_address	Receiving port address.

Description

t3rt_log_extract_reply_sent extracts information describing a successfull unicast "reply" TTCN-3 statement. This function is available for user-defined log mechanisms.

t3rt_log_extract_reply_sent_mc

Decode parameters of Reply Sent and SUT Reply Sent events.

```
void
t3rt_log_extract_reply_sent_mc
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_value_t *template_reply,
         t3rt_value_t *reply_value,
         t3rt_value_t *template_return,
         t3rt_value_t *return_value,
         unsigned long *seq_no,
         t3rt_binary_string_t **destination_comp_addr_list,
         t3rt_binary_string_t **destination_port_addr_list,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
local_port_name	Sending port name.
local_port_address	Sending port address.
template_reply	Template value for the reply signature.
reply_value	Actual reply signature value.
template_return	Template for the returned value.
return_value	Actual value returned.
seq_no	Unique reply number.
destination_comp_addr_list	NULL-terminated list of receiving components addresses (SUT address for SUT replies).
destination_port_addr_list	NULL-terminated list of receiving ports addresses.

Description

t3rt_log_extract_reply_sent_mc extracts information describing a successfull multicast "reply" TTCN-3 statement. This function is available for user-defined log mechanisms.

t3rt_log_extract_reply_sent_bc

Decode parameters of Reply Sent and SUT Reply Sent events.

```
void
t3rt_log_extract_reply_sent_bc
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_value_t *template_reply,
         t3rt_value_t *reply_value,
         t3rt_value_t *template_return,
         t3rt_value_t *return_value,
         unsigned long *seq_no,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
local_port_name	Sending port name.
local_port_address	Sending port address.
template_reply	Template value for the reply signature.
reply_value	Actual reply signature value.
template_return	Template for the returned value.
return_value	Actual value returned.
seq_no	Unique reply number.

Description

t3rt_log_extract_reply_sent_bc extracts information describing a successfull broadcast "reply" TTCN-3 statement. This function is available for user-defined log mechanisms.

t3rt_log_extract_reply_failed

Decode parameters of Reply Failed and SUT Reply Failed events.

```
void t3rt_log_extract_reply_failed
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_value_t *template_reply,
```

```
t3rt_value_t *reply_value,
t3rt_value_t *template_return,
t3rt_value_t *return_value,
unsigned long *seq_no,
t3rt_binary_string_t *destination_comp_address,
t3rt_binary_string_t *destination_port_address,
bool *codec_status,
bool *communication_status,
t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
local_port_name	Sending port name.
local_port_address	Sending port address.
template_reply	Template value for the reply signature.
reply_value	Actual reply signature value.
template_return	Template for the returned value.
return_value	Actual value returned.
seq_no	Unique reply number.
destination_comp_address	Receiving component address (SUT address for SUT replies).
destination_port_address	Receiving port address.
codec_statsu	Status of encoding operation.
communication_status	Status of transmission operation.

Description

`t3rt_log_extract_reply_failed` extracts information describing failed unicast "getcall" TTCN-3 statement. This function is available for user-defined log mechanisms.

t3rt_log_extract_reply_failed_mc

Decode parameters of Reply Failed and SUT Reply Failed events.

```
void t3rt_log_extract_reply_failed_mc
```

```
(t3rt_value_t params[],
 const char **local_port_name,
 t3rt_binary_string_t *local_port_address,
 t3rt_value_t *template_reply,
 t3rt_value_t *reply_value,
 t3rt_value_t *template_return,
 t3rt_value_t *return_value,
 unsigned long *seq_no,
 t3rt_binary_string_t **destination_comp_addr_list,
 t3rt_binary_string_t **destination_port_addr_list,
 bool *codec_status,
 bool *communication_status,
 t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
local_port_name	Sending port name.
local_port_address	Sending port address.
template_reply	Template value for the reply signature.
reply_value	Actual reply signature value.
template_return	Template for the returned value.
return_value	Actual value returned.
seq_no	Unique reply number.
destination_comp_addr_list	NULL-terminated list of receiving components addresses (SUT address for SUT replies).
destination_port_addr_list	NULL-terminated list of receiving ports addresses.
codec_statsu	Status of encoding operation.
communication_status	Status of transmission operation.

Description

`t3rt_log_extract_reply_failed_mc` extracts information describing failed multicast "getcall" TTCN-3 statement. This function is available for user-defined log mechanisms.

t3rt_log_extract_reply_failed_bc

Decode parameters of Reply Failed and SUT Reply Failed events.

```
void t3rt_log_extract_reply_failed_bc
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_value_t *template_reply,
         t3rt_value_t *reply_value,
         t3rt_value_t *template_return,
         t3rt_value_t *return_value,
         unsigned long *seq_no,
         bool *codec_status,
         bool *communication_status,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
local_port_name	Sending port name.
local_port_address	Sending port address.
template_reply	Template value for the reply signature.
reply_value	Actual reply signature value.
template_return	Template for the returned value.
return_value	Actual value returned.
seq_no	Unique reply number.
codec_statsu	Status of encoding operation.
communication_status	Status of transmission operation.

Description

`t3rt_log_extract_reply_failed_bc` extracts information describing failed broadcast "getcall" TTCN-3 statement. This function is available for user-defined log mechanisms.

t3rt_log_extract_reply_detected

Decode parameters of Reply Detected and SUT Reply Detected events.

```
void t3rt_log_extract_reply_detected
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_binary_string_t *destination_comp_address,
         unsigned long *seq_no,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
local_port_name	Receiving port name.
local_port_address	Receiving port address.
destination_comp_address	Sending component address (SUT address for SUT replies).
seq_no	Unique reply number.

Description

t3rt_log_extract_reply_detected extracts information describing a procedure reply (received by the integration) inserted into port queue. This function is available for user-defined log mechanisms.

t3rt_log_extract_reply_received, t3rt_log_extract_reply_found

Decode parameters of Reply Received, SUT Reply Received, Reply Found and SUT Reply Found events.

```
void t3rt_log_extract_reply_received
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_value_t *template_reply,
         t3rt_value_t *reply_value,
         unsigned long *seq_no,
         t3rt_binary_string_t *destination_comp_address,
         t3rt_context_t ctx);

void t3rt_log_extract_reply_found
         (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_value_t *template_reply,
         t3rt_value_t *reply_value,
```

```
        unsigned long *seq_no,
        t3rt_binary_string_t *destination_comp_address,
        t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
local_port_name	Receiving port name.
local_port_address	Receiving port address.
template_value	Signature template for the reply.
reply_value	Signature value for the reply.
seq_no	Unique reply number.
destination_comp_address	Sending component address (SUT address for SUT replies).

Description

t3rt_log_extract_reply_received extracts information describing TTCN-3 “getreply“ statement. t3rt_log_extract_reply_found extracts information describing TTCN-3 “check(getreply)“ statement. These functions are available for user-defined log mechanisms.

t3rt_log_extract_exception_raised

Decode parameters of Exception Raised and SUT Exception Raised events.

```
void t3rt_log_extract_exception_raised
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_value_t *template_exception,
         t3rt_value_t *exception_value,
         unsigned long *seq_no,
         t3rt_binary_string_t *destination_comp_address,
         t3rt_binary_string_t *destination_port_address,
         t3rt_context_t ctx);
```

Parameters

`params`	Array of event parameters, received from RTS.
`local_port_name`	Sending port name.
`local_port_address`	Sending port address.
`template_exception`	Template for the raised exception.
`exception_value`	Actual exception value raised.
`seq_no`	Unique exception number.
`destination_comp_address`	Receiving component address (SUT address for SUT replies).
`destination_port_address`	Receiving port address.

Description

`t3rt_log_extract_exception_raised` extracts information describing a successfull unicast "raise" TTCN-3 statement. This function is available for user-defined log mechanisms.

t3rt_log_extract_exception_raised_mc

Decode parameters of Exception Raised and SUT Exception Raised events.

```
void t3rt_log_extract_exception_raised_mc
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_value_t *template_exception,
         t3rt_value_t *exception_value,
         unsigned long *seq_no,
         t3rt_binary_string_t **destination_comp_addr_list,
         t3rt_binary_string_t **destination_port_addr_list,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
local_port_name	Sending port name.
local_port_address	Sending port address.
template_exception	Template for the raised exception.
exception_value	Actual exception value raised.
seq_no	Unique exception number.
destination_comp_addr_list	NULL-terminated list of receiving components addresses (SUT address for SUT replies).
destination_port_addr_list	NULL-terminated list of receiving ports addresses.

Description

t3rt_log_extract_exception_raised_mc extracts information describing a successfull multicast "raise" TTCN-3 statement. This function is available for user-defined log mechanisms.

t3rt_log_extract_exception_raised_bc

Decode parameters of Exception Raised and SUT Exception Raised events.

```
void t3rt_log_extract_exception_raised_bc
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_value_t *template_exception,
         t3rt_value_t *exception_value,
         unsigned long *seq_no,
         t3rt_context_t ctx);
```

Parameters

`params`	Array of event parameters, received from RTS.
`local_port_name`	Sending port name.
`local_port_address`	Sending port address.
`template_exception`	Template for the raised exception.
`exception_value`	Actual exception value raised.
`seq_no`	Unique exception number.

Description

`t3rt_log_extract_exception_raised_bc` extracts information describing a successfull broadcast "raise" TTCN-3 statement. This function is available for user-defined log mechanisms.

t3rt_log_extract_raise_failed

Decode parameters of Raise Failed and SUT Raise Failed events.

```
void t3rt_log_extract_raise_failed
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_value_t *template_exception,
         t3rt_value_t *exception_value,
         unsigned long *seq_no,
         t3rt_binary_string_t *destination_comp_address,
         t3rt_binary_string_t *destination_port_address,
         bool *codec_status,
         bool *communication_status,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
local_port_name	Sending port name.
local_port_address	Sending port address.
template_exception	Template for the raised exception.
exception_value	Actual exception value raised.
seq_no	Unique exception number.
destination_comp_address	Receiving component address (SUT address for SUT replies).
destination_port_address	Receiving port address.
codec_status	Status of encoding operation.
communication_status	Status of communication operation.

Description

t3rt_log_extract_raise_failed extracts information describing failed unicast "raise" TTCN-3 statement. This function is available for user-defined log mechanisms.

t3rt_log_extract_raise_failed_mc

Decode parameters of Raise Failed and SUT Raise Failed events.

```
void t3rt_log_extract_raise_failed_mc
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_value_t *template_exception,
         t3rt_value_t *exception_value,
         unsigned long *seq_no,
         t3rt_binary_string_t **destination_comp_addr_list,
         t3rt_binary_string_t **destination_port_addr_list,
         bool *codec_status,
         bool *communication_status,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
local_port_name	Sending port name.
local_port_address	Sending port address.
template_exception	Template for the raised exception.
exception_value	Actual exception value raised.
seq_no	Unique exception number.
destination_comp_addr_list	NULL-terminated list of receiving components addresses (SUT address for SUT replies).
destination_port_addr_list	NULL-terminated list of receiving ports addresses.
codec_status	Status of encoding operation.
communication_status	Status of communication operation.

Description

`t3rt_log_extract_raise_failed_mc` extracts information describing failed multicast "raise" TTCN-3 statement. This function is available for user-defined log mechanisms.

t3rt_log_extract_raise_failed_bc

Decode parameters of Raise Failed and SUT Raise Failed events.

```
void t3rt_log_extract_raise_failed_bc
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_value_t *template_exception,
         t3rt_value_t *exception_value,
         unsigned long *seq_no,
         bool *codec_status,
         bool *communication_status,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
local_port_name	Sending port name.
local_port_address	Sending port address.
template_exception	Template for the raised exception.
exception_value	Actual exception value raised.
seq_no	Unique exception number.
codec_status	Status of encoding operation.
communication_status	Status of communication operation.

Description

t3rt_log_extract_raise_failed_bc extracts information describing failed broadcast "raise" TTCN-3 statement. This function is available for user-defined log mechanisms.

t3rt_log_extract_exception_detected

Decode parameters of Exception Detected and SUT Exception Detected events.

```
void t3rt_log_extract_exception_detected
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_binary_string_t *detected_data,
         unsigned long *seq_no,
         t3rt_binary_string_t *destination_comp_address,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
local_port_name	Receiving port name.
local_port_address	Receiving port address.
detected_data	Encoded exception value.
seq_no	Unique exception number.
destination_comp_address	Receiving component address (SUT address for SUT replies).

Description

t3rt_log_extract_exception_detected extracts information describing a procedure exception (received by the integration) inserted into port queue. This function is available for user-defined log mechanisms.

t3rt_log_extract_exception_caught, t3rt_log_extract_exception_found

Decode parameters of Exception Caught, SUT Exception Caught, Exception Found and SUT Exception Found events.

```
void t3rt_log_extract_exception_caught
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_value_t *template_value,
         t3rt_value_t *caught_value,
         unsigned long *seq_no,
         t3rt_binary_string_t *destination_comp_address,
         t3rt_context_t ctx);

void t3rt_log_extract_exception_found
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_value_t *template_value,
         t3rt_value_t *caught_value,
         unsigned long *seq_no,
         t3rt_binary_string_t *destination_comp_address,
         t3rt_context_t ctx);
```

Parameters

`params`	Array of event parameters, received from RTS.
`local_port_name`	Receiving port name.
`local_port_address`	Receiving port address.
`template_value`	Encoded exception value.
`caught_value`	Actual exception value caught.
`seq_no`	Unique exception number.
`destination_comp_address`	Sending component address (SUT address for SUT exceptions).

Description

`t3rt_log_extract_exception_caught` extracts information describing TTCN-3 “catch“ statement. `t3rt_log_extract_exception_found` extracts information describing TTCN-3 “check(catch)“ statement. These functions are available for user-defined log mechanisms.

t3rt_log_extract_timeout_exception_detected

Decode parameters of Timeout Exception Detected and SUT Timeout Exception Detected events.

```
void t3rt_log_extract_timeout_exception_detected
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_context_t ctx);
```

Parameters

`params`	Array of event parameters, received from RTS.
`local_port_name`	Receiving port name.
`local_port_address`	Receiving port address.

Description

`t3rt_log_extract_timeout_exception_detected` extracts information describing a detected procedure call timeout exception. This function is available for user-defined log mechanisms.

t3rt_log_extract_timeout_exception_caught, t3rt_log_extract_timeout_exception_found

Decode parameters of Timeout Exception Caught, SUT Timeout Exception Caught, Timeout Exception Found and SUT Timeout Exception Found events.

```
void t3rt_log_extract_timeout_exception_caught
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_binary_string_t *destination_comp_address,
         t3rt_context_t ctx);

void t3rt_log_extract_timeout_exception_found
        (t3rt_value_t params[],
         const char **local_port_name,
         t3rt_binary_string_t *local_port_address,
         t3rt_binary_string_t *destination_comp_address,
         t3rt_context_t ctx);
```

Parameters

`params`	Array of event parameters, received from RTS.
`local_port_name`	Receiving port name.
`local_port_address`	Receiving port address.
`destination_comp_address`	Sending component address (SUT address for SUT exceptions).

Description

`t3rt_log_extract_timeout_exception_caught` extracts information describing TTCN-3 “catch(timeout)“ statement.
`t3rt_log_extract_timeout_exception_found` extracts information describing TTCN-3 “check(catch(timeout))“ statement. These functions are available for user-defined log mechanisms.

t3rt_log_extract_sender_mismatch

Decode parameters of Sender Mismatch event.

```
void t3rt_log_extract_sender_mismatch
        (t3rt_value_t params[],
         t3rt_value_t *port,
         t3rt_binary_string_t *actual_sender,
         t3rt_binary_string_t *expected_sender,
         unsigned long *seq_no,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
port	Receiving port value.
actual_sender	Encoded actual sender.
expected_sender	Encoded expected sender.
seq_no	Unique message/call/reply/exception number.

Description

t3rt_log_extract_sender_mismatch extracts information describing a mismatch of a sender address. This function is available for user-defined log mechanisms.

t3rt_log_extract_sut_action

Decode parameters of SUT Action Performed event.

```
void t3rt_log_extract_sut_action
        (t3rt_value_t params[],
         t3rt_value_t *string_or_template,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
string_or_template	String or template argument to SUT action.

Description

t3rt_log_extract_sut_action extracts information describing a TTCN-3 “action“ statement. This function is available for user-defined log mechanisms.

t3rt_log_extract_timer_started

Decode parameters of Timer Started event.

```
void t3rt_log_extract_timer_started
        (t3rt_value_t params[],
         const char **timer_name,
```

```
        unsigned long *unique_id,
        double *duration,
        double *default_duration,
        t3rt_timer_handle_t *handle,
        t3rt_context_t ctx);
```

Parameters

`params`	Array of event parameters, received from RTS.
`timer_name`	Timer name.
`unique_id`	Unique timer id.
`duration`	Timer duration.
`default_duration`	Default timer duration.
`handle`	Timer handle.

Description

`t3rt_log_extract_timer_started` extracts information describing a TTCN-3 timer “start“ statement. This function is available for user-defined log mechanisms.

t3rt_log_extract_timer_stopped

Decode parameters of Timer Stopped event.

```
void t3rt_log_extract_timer_stopped
        (t3rt_value_t params[],
         const char **timer_name,
         unsigned long *unique_id,
         t3rt_context_t ctx);
```

Parameters

`params`	Array of event parameters, received from RTS.
`timer_name`	Timer name.
`unique_id`	Unique timer id.

Description

`t3rt_log_extract_timer_stopped` extracts information describing a TTCN-3 timer “stop“ statement. This function is available for user-defined log mechanisms.

t3rt_log_extract_timer_read

Decode parameters of Timer Read event.

```
void t3rt_log_extract_timer_read
        (t3rt_value_t params[],
         const char **timer_name,
         unsigned long *unique_id,
         double *elapsed_time,
         t3rt_timer_state_t *state,
         double *duration,
         double *default_duration,
         t3rt_timer_handle_t *handle,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
timer_name	Timer name.
unique_id	Unique timer id.
elapsed_time	Time elapsed since timer start.
state	Timer state.
duration	Timer duration.
default_duration	Default timer duration.
handle	Timer handle.

Description

t3rt_log_extract_timer_read extracts information describing a TTCN-3 timer “read“ statement. This function is available for user-defined log mechanisms.

t3rt_log_extract_timer_is_running

Decode parameters of Timer Is Running Check Performed event.

```
void t3rt_log_extract_timer_is_running
        (t3rt_value_t params[],
         const char **timer_name,
         unsigned long *unique_id,
         double *elapsed_time,
         t3rt_timer_state_t *state,
         double *duration,
         double *default_duration,
         t3rt_timer_handle_t *handle,
         t3rt_context_t ctx);
```

Parameters

`params`	Array of event parameters, received from RTS.
`timer_name`	Timer name.
`unique_id`	Unique timer id.
`elapsed_time`	Time elapsed since timer start.
`state`	Timer state.
`duration`	Timer duration.
`default_duration`	Default timer duration.
`handle`	Timer handle.

Description

`t3rt_log_extract_timer_is_running` extracts information describing a TTCN-3 timer “running“ statement. This function is available for user-defined log mechanisms.

t3rt_log_extract_timeout_detected

Decode parameters of timer Timer Timeout Detected event.

```
void t3rt_log_extract_timeout_detected
        (t3rt_value_t params[],
         const char **timer_name,
         unsigned long *unique_id,
         t3rt_context_t ctx);
```

Parameters

`params`	Array of event parameters, received from RTS.
`timer_name`	Timer name.
`unique_id`	Unique timer id.

Description

`t3rt_log_extract_timeout_detected` extracts information describing a detected timer timeout (when triTimeout is called). This function is available for user-defined log mechanisms.

t3rt_log_extract_timeout_received

Decode parameters of Timer Timed Out Check Succeeded event.

```
void t3rt_log_extract_timeout_received
        (t3rt_value_t params[],
         const char **timer_name,
         unsigned long *unique_id,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
timer_name	Timer name.
unique_id	Unique timer id.

Description

t3rt_log_extract_timeout_received extracts information describing matched timeout alternative. This function is available for user-defined log mechanisms.

t3rt_log_extract_timeout_mismatch

Decode parameters of Timer Timed Out Check Failed event.

```
void t3rt_log_extract_timeout_mismatch
        (t3rt_value_t params[],
         const char **timer_name,
         unsigned long *unique_id,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
timer_name	Timer name.
unique_id	Unique timer id.

Description

t3rt_log_extract_timeout_mismatch extracts information describing mismatched timeout alternative. This function is available for user-defined log mechanisms.

t3rt_log_extract_component_created

Decode parameters of Component Created event.

```
void t3rt_log_extract_component_created
        (t3rt_value_t params[],
         const char **component_name,
         t3rt_binary_string_t *component_address,
         t3rt_context_t ctx);
```

Parameters

`params`	Array of event parameters, received from RTS.
`component_name`	Name of the created component.
`component_address`	Address of the created component.

Description

`t3rt_log_extract_component_created` extracts information describing component creation event that is generated for TTCN-3 “create“ and “execute“ operations. This function is available for user-defined log mechanisms.

t3rt_log_extract_component_started

Decode parameters of Component Started event.

```
void t3rt_log_extract_component_started
        (t3rt_value_t params[],
         t3rt_binary_string_t *component_address,
         const char **module,
         const char **function,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
component_address	Address of the started component.
module	Module name of the started function.
function	Name of the started function.

Description

t3rt_log_extract_component_started extracts information describing component start event that is generated for TTCN-3 “start“ and “execute“ operations. This function is available for user-defined log mechanisms.

t3rt_log_extract_component_is_running

Decode parameters of Component Is Running Check Performed event.

```
void t3rt_log_extract_component_is_running
        (t3rt_value_t params[],
         t3rt_binary_string_t *component_address,
         bool *is_running,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
component_address	Address of the checked component.
is_running	State of the checked component.

Description

t3rt_log_extract_component_is_running extracts information describing TTCN-3 component “running“ operation. This function is available for user-defined log mechanisms.

t3rt_log_extract_component_is_alive

Decode parameters of Component Is Alive Check Performed event.

```
void t3rt_log_extract_component_is_alive
```

```
        (t3rt_value_t params[],
         t3rt_binary_string_t *component_address,
         bool *is_alive,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
component_address	Address of the checked component.
is_alive	State of the checked component.

Description

`t3rt_log_extract_component_is_alive` extracts information describing TTCN-3 component “alive“ operation. This function is available for user-defined log mechanisms.

t3rt_log_extract_component_stopped

Decode parameters of Component Stopped event.

```
void t3rt_log_extract_component_stopped
        (t3rt_value_t params[],
         t3rt_binary_string_t *component_address,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
component_address	Address of the stopped component.

Description

`t3rt_log_extract_component_stopped` extracts information describing TTCN-3 component termination event. This function is available for user-defined log mechanisms.

t3rt_log_extract_component_killed

Decode parameters of Component Killed event.

```
void t3rt_log_extract_component_killed
```

```
        (t3rt_value_t params[],
         t3rt_binary_string_t *component_address,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
component_address	Address of the killed component.

Description

t3rt_log_extract_component_killed extracts information describing TTCN-3 alive component termination event. This function is available for user-defined log mechanisms.

t3rt_log_extract_done_check_succeeded

Decode parameters of Component Done Check Succeeded event.

```
void t3rt_log_extract_done_check_succeeded
        (t3rt_value_t params[],
         t3rt_binary_string_t *component_address,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
component_address	Address of the checked component.

Description

t3rt_log_extract_done_check_succeeded extracts information describing matched component “done“ alternative. This function is available for user-defined log mechanisms.

t3rt_log_extract_done_check_failed

Decode parameters of Component Done Check Failed event.

```
void t3rt_log_extract_done_check_failed
        (t3rt_value_t params[],
         t3rt_binary_string_t *component_address,
         t3rt_context_t ctx);
```

Parameters

`params`	Array of event parameters, received from RTS.
`component_address`	Address of the checked component.

Description

`t3rt_log_extract_done_check_failed` extracts information describing mismatched component “done“ alternative. This function is available for user-defined log mechanisms.

t3rt_log_extract_kill_check_succeeded

Decode parameters of Component Killed Check Succeeded event.

```
void t3rt_log_extract_kill_check_succeeded
        (t3rt_value_t params[],
         t3rt_binary_string_t *component_address,
         t3rt_context_t ctx);
```

Parameters

`params`	Array of event parameters, received from RTS.
`component_address`	Address of the checked component.

Description

`t3rt_log_extract_kill_check_succeeded` extracts information describing matched component “killed“ alternative. This function is available for user-defined log mechanisms.

t3rt_log_extract_kill_check_failed

Decode parameters of Component Killed Check Failed event.

```
void t3rt_log_extract_kill_check_failed
        (t3rt_value_t params[],
         t3rt_binary_string_t *component_address,
         t3rt_context_t ctx);
```

Parameters

`params`	Array of event parameters, received from RTS.
`component_address`	Address of the checked component.

Description

`t3rt_log_extract_kill_check_failed` extracts information describing mismatched component “killed“ alternative. This function is available for user-defined log mechanisms.

t3rt_log_extract_port_connected

Decode parameters of Port Connected event.

```
void t3rt_log_extract_port_connected
        (t3rt_value_t params[],
         t3rt_binary_string_t *component_address1,
         const char **port_name1,
         t3rt_binary_string_t *port_address1,
         t3rt_binary_string_t *component_address2,
         const char **port_name2,
         t3rt_binary_string_t *port_address2,
         t3rt_context_t ctx);
```

Parameters

`params`	Array of event parameters, received from RTS.
`component_address1`	First port component address.
`port_name1`	First port.name.
`port_address1`	First port address
`component_address2`	Second port component address.
`port_name2`	Second port name.
`port_address2`	Second port address.

Description

`t3rt_log_extract_port_connected` extracts information describing TTCN-3 port “connect“ operation. This function is available for user-defined log mechanisms.

t3rt_log_extract_port_disconnected

Decode parameters of Port Disconnected event.

```
void t3rt_log_extract_port_disconnected
        (t3rt_value_t params[],
         t3rt_binary_string_t *component_address1,
         const char **port_name1,
         t3rt_binary_string_t *port_address1,
         t3rt_binary_string_t *component_address2,
         const char **port_name2,
         t3rt_binary_string_t *port_address2,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
component_address1	First port component address.
port_name1	First port.name.
port_address1	First port address
component_address2	Second port component address.
port_name2	Second port name.
port_address2	Second port address.

Description

t3rt_log_extract_port_disconnected extracts information describing TTCN-3 port “disconnect“ operation. This function is available for user-defined log mechanisms.

t3rt_log_extract_port_mapped

Decode parameters of Port Mapped event.

```
void t3rt_log_extract_port_mapped
        (t3rt_value_t params[],
         t3rt_binary_string_t *component_address,
         const char **local_port_name,
         const char **system_port_name,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
component_address	Local port component address.
local_port_name	Local port.name.
system_port_name	System (TSI) port name.

Description

t3rt_log_extract_port_mapped extracts information describing TTCN-3 port “map“ operation. This function is available for user-defined log mechanisms.

t3rt_log_extract_port_unmapped

Decode parameters of Port Unmapped event.

```
void t3rt_log_extract_port_unmapped
        (t3rt_value_t params[],
         t3rt_binary_string_t *component_address,
         const char **local_port_name,
         const char **system_port_name,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
component_address	Local port component address.
local_port_name	Local port.name.
system_port_name	System (TSI) port name.

Description

t3rt_log_extract_port_unmapped extracts information describing TTCN-3 port “unmap“ operation. This function is available for user-defined log mechanisms.

t3rt_log_extract_port_enabled

Decode parameters of Port Enabled event.

```
void t3rt_log_extract_port_enabled
        (t3rt_value_t params[],
         const char **port_name,
         t3rt_binary_string_t *port_address,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
port_name	Port.name.
port_address	Port address.

Description

t3rt_log_extract_port_enabled extracts information describing TTCN-3 port “start“ operation. This function is available for user-defined log mechanisms.

t3rt_log_extract_port_disabled

Decode parameters of Port Disabled event.

```
void t3rt_log_extract_port_disabled
        (t3rt_value_t params[],
         const char **port_name,
         t3rt_binary_string_t *port_address,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
port_name	Port.name.
port_address	Port address.

Description

t3rt_log_extract_port_disabled extracts information describing TTCN-3 port “stop“ operation. This function is available for user-defined log mechanisms.

t3rt_log_extract_port_halted

Decode parameters of Port Halted event.

```
void t3rt_log_extract_port_halted
        (t3rt_value_t params[],
         const char **port_name,
         t3rt_binary_string_t *port_address,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
port_name	Port.name.
port_address	Port address.

Description

t3rt_log_extract_port_halted extracts information describing TTCN-3 port “halt“ operation. This function is available for user-defined log mechanisms.

t3rt_log_extract_port_cleared

Decode parameters of Port Cleared event.

```
void t3rt_log_extract_port_cleared
        (t3rt_value_t params[],
         const char **port_name,
         t3rt_binary_string_t *port_address,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
port_name	Port.name.
port_address	Port address.

Description

t3rt_log_extract_port_cleared extracts information describing TTCN-3 port “clear“ operation. This function is available for user-defined log mechanisms.

t3rt_log_extract_local_verdict_changed

Decode parameters of Local Verdict Set event.

```
void t3rt_log_extract_local_verdict_changed
        (t3rt_value_t params[],
         t3rt_verdict_t *prev_verdict,
         t3rt_verdict_t *attempt_verdict,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
prev_verdict	Previous verdict.
attempt_verdict	New verdict to be set.

Description

t3rt_log_extract_verdict_changed extracts information describing TTCN-3 “setverdict“ operation. This event is generated in several other cases (see Local Verdict Set event description for more info). This function is available for user-defined log mechanisms.

t3rt_log_extract_local_verdict_queried

Decode parameters of Local Verdict Read event.

```
void t3rt_log_extract_local_verdict_queried
        (t3rt_value_t params[],
         t3rt_verdict_t *verdict,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
verdict	Current verdict.

Description

t3rt_log_extract_verdict_queried extracts information describing TTCN-3 “getverdict“ operation. This function is available for user-defined log mechanisms.

t3rt_log_extract_template_match_failed

Decode parameters of Template Match Failed event.

```
void t3rt_log_extract_template_match_failed
        (t3rt_value_t params[],
         t3rt_value_t *template_value,
         t3rt_value_t *unmatched_value,
         t3rt_context_t ctx);
```

Parameters

`params`	Array of event parameters, received from RTS.
`template_value`	Template value for the matching operation.
`unmatched_value`	Mismatched value.

Description

`t3rt_log_extract_template_match_failed` extracts information describing failed matching operation. This function is available for user-defined log mechanisms.

t3rt_log_extract_template_mismatch

Decode parameters of Template Mismatch event.

```
void t3rt_log_extract_template_mismatch
        (t3rt_value_t params[],
         t3rt_value_t *field_or_item_specifier,
         t3rt_value_t *unmatched_value,
         t3rt_value_t* reference_value,
         const char** reference_value_descr,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
field_or_item_specifier	Integer or charstring value describing the specifier of unmatched field.
unmatched_value	Unmatched value.
reference_value	Reference value used in match operation.
reference_value_descr	Description of failed matching operation.

Description

t3rt_log_extract_template_mismatch extracts information describing failed match of a value or a field value. This function is available for user-defined log mechanisms.

t3rt_log_extract_template_match_begin

Decode parameters of Template Match Begin event.

```
void t3rt_log_extract_template_match_begin
        (t3rt_value_t params[],
         t3rt_value_t * matched_value,
         t3rt_value_t * reference_value,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
matched_value	Value to be matched.
reference_value	Reference value used in match operation.

Description

t3rt_log_extract_template_match_begin extracts information describing start of a match operation. This function is available for user-defined log mechanisms.

t3rt_log_extract_template_match_end

Decode parameters of Template Match End event.

```
void t3rt_log_extract_template_match_end
        (t3rt_value_t params[],
         bool *status,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
status	Result of the matching operation (success or fail).

Description

t3rt_log_extract_template_match_end extracts information describing end of a match operation. This function is available for user-defined log mechanisms.

t3rt_log_extract_testcase_started

Decode parameters of Test case started event.

```
void t3rt_log_extract_testcase_started
        (t3rt_value_t params[],
         const char **module_name,
         const char **testcase_name,
         double *timeout,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
module_name	Module name of the started test case.
testcase_name	Name of the started test case.
timeout	Test case timeout.

Description

t3rt_log_extract_testcase_started extracts information describing start of a test case. This function is available for user-defined log mechanisms.

t3rt_log_extract_testcase_ended

Decode parameters of Test case ended event.

```
void t3rt_log_extract_testcase_ended
        (t3rt_value_t params[],
         const char **module_name,
         const char **testcase_name,
         t3rt_verdict_t *verdict,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
module_name	Module name of the terminated test case.
testcase_name	Name of the terminated test case.
verdict	Test case verdict.

Description

t3rt_log_extract_testcase_ended extracts information describing termination of a test case. This function is available for user-defined log mechanisms.

t3rt_log_extract_testcase_timed_out

Decode parameters of Test case timed out event.

```
void t3rt_log_extract_testcase_timed_out
        (t3rt_value_t params[],
         const char **module_name,
         const char **testcase_name,
         double *timeout,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
module_name	Module name of the timed out test case.
testcase_name	Name of the timed out test case.
timeout	Test case timeout.

Description

t3rt_log_extract_testcase_timed_out extracts information describing test case timeout event. This function is available for user-defined log mechanisms.

t3rt_log_extract_testcase_error

Decode parameters of Test case error event.

```
void t3rt_log_extract_testcase_error
        (t3rt_value_t params[],
         const char **module_name,
         const char **scope_name,
         const char **error_msg,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
module_name	Module name of the scope that generated error.
scope_name	Name of the scope (i.e. function) that generated error.
error_msg	Error description.

Description

t3rt_log_extract_testcase_error extracts information describing test case error event. This function is available for user-defined log mechanisms.

t3rt_log_extract_test_case_verdict

Decode parameters of Test case verdict event.

```
void t3rt_log_extract_test_case_verdict
        (t3rt_value_t params[],
         const char **testcase_name,
         t3rt_verdict_t *verdict,
         t3rt_context_t ctx);
```

Parameters

`params`	Array of event parameters, received from RTS.
`testcase_name`	Name of the terminated test case.
`verdict`	Test case verdict.

Description

`t3rt_log_extract_test_case_verdict` extracts information describing setting overall test case verdict. This function is available for user-defined log mechanisms.

t3rt_log_extract_variable_modified

Decode parameters of Variable Modified event.

```
void t3rt_log_extract_variable_modified
        (t3rt_value_t params[],
         t3rt_value_t *value,
         t3rt_context_t ctx);
```

Parameters

`params`	Array of event parameters, received from RTS.
`value`	Modified value (after modification).

Description

`t3rt_log_extract_variable_modified` extracts information describing modification odf a variable, constant or module parameter. This function is available for user-defined log mechanisms.

t3rt_log_extract_scope_entered

Decode parameters of Scope Entered event.

```
void t3rt_log_extract_scope_entered
```

```
        (t3rt_value_t params[],
         const char **scope_name,
         t3rt_scope_kind_t *scope_kind,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
scope_name	Name of the entered scope.
scope_kind	Scope kind.

Description

t3rt_log_extract_scope_entered extracts information describing entering a new scope (function, testcase, altstep, etc). This function is available for user-defined log mechanisms.

t3rt_log_extract_scope_changed

Decode parameters of Scope Changed event.

```
void t3rt_log_extract_scope_changed
        (t3rt_value_t params[],
         unsigned long *line_number,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
line_number	Source TTCN-3 position in the scope.

Description

t3rt_log_extract_scope_changed extracts information describing change in the scope source TTCN-3 position. This function is available for user-defined log mechanisms.

t3rt_log_extract_scope_left

Decode parameters of Scope Left event.

```
void t3rt_log_extract_scope_left
        (t3rt_value_t params[],
```

```
        const char **scope_name,
        t3rt_scope_kind_t *scope_kind,
        t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
scope_name	Name of the left scope.
scope_kind	Scope kind.

Description

t3rt_log_extract_scope_left extracts information describing leaving of a scope (fucntion, testcase, altstep, etc). This function is available for user-defined log mechanisms.

t3rt_log_extract_alternative_activated_event

Decode parameters of Alternative Activated event.

```
void t3rt_log_extract_alternative_activated_event
        (t3rt_value_t params[],
         const char **module_name,
         const char **altstep_name,
         t3rt_value_t *default_reference,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
module_name	Module name of the activated altstep.
altstep_name	Name of the activated altstep.
default_reference	Default reference of the activated altstep.

Description

t3rt_log_extract_alternative_activated_event extracts information describing TTCN-3 “activate“ operation. This function is available for user-defined log mechanisms.

t3rt_log_extract_alternative_deactivated_event

Decode parameters of Alternative Deactivated event.

```
void t3rt_log_extract_alternative_deactivated_event
        (t3rt_value_t params[],
         t3rt_value_t *default_reference,
         t3rt_context_t ctx);
```

Parameters

`params`	Array of event parameters, received from RTS.
`default_reference`	Default reference of the deactivated altstep.

Description

`t3rt_log_extract_alternative_deactivated_event` extracts information describing TTCN-3 “deactivate“ operation. This function is available for user-defined log mechanisms.

t3rt_log_extract_message_decoded

Decode parameters of Data Decoded event.

```
void t3rt_log_extract_message_decoded
        (t3rt_value_t params[],
         t3rt_binary_string_t * encoded_data,
         t3rt_value_t * decoded_value,
         t3rt_codecs_strategy_t * strategy,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
encoded_data	Encoded data.
decoded_value	Decoded value.
strategy	Decoding strategy selected by the RTS.

Description

t3rt_log_extract_message_decoded extracts information describing successfull decoding of an encoded data. This function is available for user-defined log mechanisms.

t3rt_log_extract_message_decode_failed

Decode parameters of Data Decoding Failed event.

```
void t3rt_log_extract_message_decode_failed
        (t3rt_value_t params[],
         t3rt_binary_string_t * encoded_data,
         t3rt_codecs_strategy_t * strategy,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
encoded_data	Encoded data.
strategy	Decoding strategy selected by the RTS.

Description

t3rt_log_extract_message_decode_failed extracts information describing failed decoding of an encoded data. This function is available for user-defined log mechanisms.

t3rt_log_extract_message_encoded

Decode parameters of Data Encoded event.

```
void t3rt_log_extract_message_encoded
        (t3rt_value_t params[],
         t3rt_value_t * value,
```

```
        t3rt_binary_string_t * encoded_data,
        t3rt_codecs_strategy_t * strategy,
        t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
value	Encoded value.
encoded_data	Encoded data.
strategy	Encoding strategy selected by the RTS.

Description

t3rt_log_extract_message_encoded extracts information describing successfull encoding of a value. This function is available for user-defined log mechanisms.

t3rt_log_extract_message_encode_failed

Decode parameters of Data Encoding Failed event.

```
void t3rt_log_extract_message_encode_failed
        (t3rt_value_t params[],
         t3rt_value_t * value,
         t3rt_codecs_strategy_t * strategy,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
value	Value to be encoded.
strategy	Encoding strategy selected by the RTS.

Description

t3rt_log_extract_message_encode_failed extracts information describing failed encoding of a value. This function is available for user-defined log mechanisms.

t3rt_log_extract_text_message_string

Decode parameters of Information Message, Warning Message, Error Message, Debug Message and TTCN-3 Message events.

```
void t3rt_log_extract_text_message_string
        (t3rt_value_t params[],
         const char** text,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
text	ASCII message description.

Description

t3rt_log_extract_text_message_string extracts information describing test message event. Information is extracted into ASCII string. This function is available for user-defined log mechanisms.

t3rt_log_extract_text_message_widestring

Decode parameters of Information Message, Warning Message, Error Message, Debug Message and TTCN-3 Message events.

```
void t3rt_log_extract_text_message_widestring
        (t3rt_value_t params[],
         t3rt_wide_string_t* text,
         t3rt_context_t ctx);
```

Parameters

params	Array of event parameters, received from RTS.
text	Possibly internationalized message description.

Description

t3rt_log_extract_text_message_widestring extracts information describing test message event. Information is extracted into wide string. This function is available for user-defined log mechanisms.

t3rt_log_extract_function_call

Decode parameters of Function called event.

```
void t3rt_log_extract_function_call
        (t3rt_value_t params[],
         const char **function_name,
         t3rt_value_t *signature_value,
         t3rt_context_t ctx);
```

Parameters

`params`	Array of event parameters, received from RTS.
`function_name`	Invoked function name.
`signature_value`	Signature value describing actual parameters.

Description

`t3rt_log_extract_function_call` extracts information describing invocation of a function. This function is available for user-defined log mechanisms.

t3rt_log_extract_external_function_call

Decode parameters of External Function Called event.

```
void t3rt_log_extract_external_function_call
        (t3rt_value_t params[],
         const char **function_name,
         t3rt_value_t *signature_value,
         t3rt_context_t ctx);
```

Parameters

`params`	Array of event parameters, received from RTS.
`function_name`	Invoked external function name.
`signature_value`	Signature value describing actual parameters.

Description

`t3rt_log_extract_external_function_call` extracts information describing invocation of an external function. This function is available for user-defined log mechanisms.

t3rt_log_extract_altstep_call

Decode parameters of Altstep Called event.

```
void t3rt_log_extract_altstep_call
        (t3rt_value_t params[],
         const char **altstep_name,
         t3rt_value_t *signature_value,
         t3rt_context_t ctx);
```

Parameters

`params`	Array of event parameters, received from RTS.
`altstep_name`	Invoked altstep name.
`signature_value`	Signature value describing actual parameters.

Description

`t3rt_log_extract_altstep_call` extracts information describing invocation of an altstep. This function is available for user-defined log mechanisms.

RTL Wide String Functions

RTL Wide String Related Type Definitions

t3rt_wide_string_t

This is a string that contains multi-byte characters (t3rt_wide_char_t). It is used for localized strings and for the representation of the universal_charstring TTCN-3 type.

t3rt_wide_char_t

Representation of a character inside a t3rt_wide_string_t.

t3rt_wchar2int

Converts wide character content into the corresponding integer code number (ISO-10646).

```
void t3rt_wchar2int
        (const t3rt_wide_char_t wchar,
         unsigned long * wchar_code,
```

```
        t3rt_context_t context);
```

Parameters

wchar	Wide char value to be converted.
wchar_code	Output parameter that receives conversion result.

Description

This function converts wide character representation (i.e. byte array) into an integer value. Each of four bytes in the result integer stores corresponding part from t3rt_wide_char_t character representation.

t3rt_wchar2quad

Converts wide character content into the corresponding quadruple.

```
void t3rt_wchar2quad
        (const t3rt_wide_char_t wchar,
         unsigned char * group,
         unsigned char * plane,
         unsigned char * row,
         unsigned char * cell,
         t3rt_context_t context);
```

Parameters

wchar	Wide char value to be converted.
group	Output parameter that receives group byte of wide character.
plane	Output parameter that receives plane byte of wide character.
row	Output parameter that receives row byte of wide character.
cell	Output parameter that receives cell byte of wide character.

Description

This function extracts group, plane, row and cell bytes from the wide character representation (i.e. byte array).

t3rt_char2wchar

Converts character (ISO-646) into the wide character content (ISO-10646).

```
void t3rt_char2wchar
        (char char_code,
         t3rt_wide_char_t * wchar,
         t3rt_context_t context);
```

Parameters

char_code	Char value to be converted.
wchar	Output parameter that receives conversion result.

Description

This function converts ASCII character represented by its character code into wide char representation.

t3rt_int2wchar

Converts integer code number (ISO-10646) into the wide character content.

```
void t3rt_int2wchar
        (unsigned long wchar_code,
         t3rt_wide_char_t * wchar,
         t3rt_context_t context);
```

Parameters

wchar_code	Integer representing wide char code.
wchar	Output parameter that receives conversion result.

Description

This function converts integer value (that is the code of a wide character) into the wide character representation (i.e. byte array).

t3rt_quad2wchar

Converts quadruple into the wide character content.

```
void t3rt_quad2wchar
        (unsigned char group,
         unsigned char plane,
```

```
        unsigned char row,
        unsigned char cell,
        t3rt_wide_char_t * wchar,
        t3rt_context_t context);
```

Parameters

group	Group byte of wide character.
plane	Plane byte of wide character.
row	Row byte of wide character.
cell	Cell byte of wide character.
wchar	Output parameter that receives conversion result.

Description

This function merges group, plane, row and cell bytes into the wide character representation (i.e. byte array).

t3rt_wchar_cmp

Compare two wide char characters.

```
int t3rt_wchar_cmp
        (const t3rt_wide_char_t wchar1,
         const t3rt_wide_char_t wchar2,
         t3rt_context_t context);
```

Parameters

wchar1	First wide character.
wchar2	Second wide character.

Description

This function compares two wide characters by their codes, i.e. result is evaluated by comparing the output of t3rt_wchar2int for both given wide chars.

Return Values

Returns -1 if “wchar1” is lesser than “wchar”2, 1 if “wchar1” is greater than “wchar2”, 0 if they are equal.

t3rt_wide_string_rotateleft

Makes a left-rotated copy of the wide string.

```
t3rt_wide_string_t t3rt_wide_string_rotateleft
        (t3rt_wide_string_t wstring,
         unsigned long char_count,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t context);
```

Parameters

wstring	The string to rotate.
char_count	The number of rotations.
strategy	Memory allocation strategy for the resulting value.

Description

Rotates the string element in the string according to ETSI ES 201 873-1 V2.2.1.

Return Values

A copy of the rotated string allocated with the specified allocation strategy.

t3rt_wide_string_rotateright

Makes a right-rotated copy of the wide string.

```
t3rt_wide_string_t t3rt_wide_string_rotateright
        (t3rt_wide_string_t wstring,
         unsigned long char_count,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t context);
```

Parameters

wstring	The string to rotate.
char_count	The number of rotations.
strategy	Memory allocation strategy for the resulting value.

Description

Rotates the string element in the string according to ETSI ES 201 873-1 V2.2.1.

Return Values

A copy of the rotated string allocated with the specified allocation strategy.

t3rt_wide_string_set_element

Sets the wide character at the specified index position.

```
void t3rt_wide_string_set_element
        (t3rt_wide_string_t wstring,
         unsigned long index,
         const t3rt_wide_char_t wchar,
         t3rt_context_t context);
```

Parameters

wstring	Wide string to be changed.
index	Zero based position in wide string to assign to.
wchar	Element to assigned to the specified position of wide string.

Description

This functions sets contents of wide string at specified position to the given wide character. Assigning value to the position outside current string boundaries generates test case error.

t3rt_wide_string_set_element_to_ascii_char

Sets the character at the specified index position.

```
void t3rt_wide_string_set_element_to_ascii_char
        (t3rt_wide_string_t wstring,
         unsigned long index,
         char chr,
         t3rt_context_t context);
```

Parameters

wstring	Wide string to be changed.
index	Zero based position in wide string to assign to.
chr	Element to assigned to the specified position of wide string.

Description

This functions sets contents of wide string at specified position to the given ASCII character. Assigning value to the position outside current string boundaries generates test case error.

t3rt_wide_string_element

Returns the wide character at the specified index position.

```
void t3rt_wide_string_element
        (const t3rt_wide_string_t wstring,
         unsigned long index,
         t3rt_wide_char_t * wchar,
         t3rt_context_t context);
```

Parameters

wstring	Queried wide string.
index	Zero based position in wide string.
wchar	Output parameter for the requested wide string element.

Description

This function extracts single wide character from the given position of the wide string. Specified element index should point to element within string boundaries otherwise test case error is generated.

t3rt_wide_string_allocate

Creates an empty wide string and pre-allocates space for the given length.

```
t3rt_wide_string_t t3rt_wide_string_allocate
        (t3rt_alloc_strategy_t strategy,
```

```
        unsigned long alloc_size,
        t3rt_context_t context);
```

Parameters

strategy	Memory allocation strategy for the created string.
alloc_size	Initial size of the string buffer.

Description

This function creates new empty wide string. Allocation size specifies the initial size of the internal string buffer. It may be equal to zero and serves only to increase the performance of the wide string operations. Allocation size is given in number of wide characters.

Return Values

New instance of wide string allocated according the specified strategy.

t3rt_wide_string_deallocate

De-allocates the wide string.

```
void t3rt_wide_string_deallocate
        (t3rt_wide_string_t * wstring,
         t3rt_context_t context);
```

Parameters

wstring	Address of wide string to be deallocated.

Description

This function deletes specified wide string and frees all memory reserved by this string.

t3rt_wide_string_construct_from_ascii

Constructs a wide string out of the NULL terminated ASCII string.

```
t3rt_wide_string_t t3rt_wide_string_construct_from_ascii
        (const char * ascii_string,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t context);
```

Parameters

ascii_string	NULL-terminated free ASCII text string.
strategy	Memory allocation strategy for the resulting string.

Description

This function creates new wide string and fills it with the given ASCII string.

Return Values

New instance of wide string allocated according the specified strategy and filled with the specified value.

t3rt_wide_string_construct_from_wchar

Constructs a single character wide string out of a wide character

```
t3rt_wide_string_t t3rt_wide_string_construct_from_wchar
        (const t3rt_wide_char_t character,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t context);
```

Parameters

character	Single wide character.
strategy	Memory allocation strategy for the resulting string.

Description

This function creates new one element wide string and fills it with the given wide char.

Return Values

New instance of wide string allocated according the specified strategy and filled with the specified value.

t3rt_wide_string_set

Sets the contents of the wide string to a copy of the character array of given length.

```
void t3rt_wide_string_set
        (t3rt_wide_string_t wstring,
         unsigned long length,
         const t3rt_wide_char_t value[],
         t3rt_context_t context);
```

Parameters

wstring	Valid wide string.
length	Length of wide character array.
value	Array of wide characters.

Description

This function rewrites contents of the given wide string with the new value. New value is specified with the array of wide characters. Length of the wide char array is provided in the separate function formal parameter.

t3rt_wide_string_set_ascii

Sets the contents of the wide string to a copy of the null-terminated character string.

```
void t3rt_wide_string_set_ascii
        (t3rt_wide_string_t wstring,
         const char * ascii_string,
         t3rt_context_t context);
```

Parameters

wstring	Valid wide string.
ascii_string	NULL-terminated ASCII string.

Description

This function rewrites contents of the given wide string with the new value. New value is specified with the NULL-terminated ASCII string.

t3rt_wide_string_set_wchar_array

Sets the contents of the wide string to a copy of the array of wide chars.

```
void t3rt_wide_string_set_wchar_array
        (t3rt_wide_string_t wstring,
         const t3rt_wide_char_t * string,
         unsigned long length,
         t3rt_context_t context);
```

Parameters

wstring	Valid wide string.
string	Array of wide characters.
length	Length of the wide char array.

Description

This function rewrites contents of the given wide string with the new value. New value is specified with the array of wide characters. Length of the wide char array is provided in the separate function formal parameter. This function behaves similar to t3rt_wide_string_set.

t3rt_wide_string_copy

Makes a copy of the wide string.

```
t3rt_wide_string_t t3rt_wide_string_copy
        (t3rt_wide_string_t wstring,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t context);
```

Parameters

wstring	Valid wide string.
strategy	memory allocation strategy for the resulting value.

Return Values

Returns copy of the wide string allocated according to the specified strategy.

t3rt_wide_string_length

Returns the number of the wide characters in the wide string.

```
unsigned long t3rt_wide_string_length
        (t3rt_wide_string_t wstring,
         t3rt_context_t context);
```

Parameters

wstring	Valid wide string.

Return Values

Returns the number of the wide characters in the wide string.

t3rt_wide_string_is_equal

Compares two wide strings for equality.

```
bool t3rt_wide_string_is_equal
        (t3rt_wide_string_t wstring1,
         t3rt_wide_string_t wstring2,
         t3rt_context_t context);
```

Parameters

wstring1	First wide string.
wstring2	Second wide string.

Description

This function accepts NULL pointer as a wide string reference. If both wide strings equal to NULL then functions returns true.

Return Values

Returns true if strings are equal, false otherwise.

t3rt_wide_string_content

Returns the actual character array of the wide string.

```
const unsigned char * t3rt_wide_string_content
```

```
        (t3rt_wide_string_t wstring,
         t3rt_context_t context);
```

Parameters

wstring	Valid wide string.

Description

This function returns reference to the internal wide string buffer. Length of this buffer is returned by the t3rt_wide_string_length function.

Return Values

Returns reference to the internal wide string buffer that is the array of wide characters.

t3rt_wide_string_assign

Assigns the src wide string to the dest wide string.

```
void t3rt_wide_string_assign
        (t3rt_wide_string_t dest,
         const t3rt_wide_string_t src,
         t3rt_context_t context);
```

Parameters

dest	Destination wide string.
src	Source wide string.

Description

This function assigns one wide string to the another. Destination wide string has to be allocated prior to assignment.

t3rt_wide_string_append

Appends one wide string the end of another.

```
void t3rt_wide_string_append
        (t3rt_wide_string_t wstring,
         t3rt_wide_string_t appwstr,
         t3rt_context_t context);
```

Parameters

wstring	Wide string that will be changed.
appwstr	Appended wide string

Description

This function appends “appwstr“ to the end of “wstring“.

t3rt_format_char_string, t3rt_format_wide_string

Support for the parametrized wide string formatting.

```
t3rt_wide_string_t t3rt_format_char_string
        (const char * fmt_cstring,
         t3rt_context_t context,
         ...);

t3rt_wide_string_t t3rt_format_wide_string
        (const t3rt_wide_string_t fmt_wstring,
         t3rt_context_t context,
         ...);
```

Parameters

fmt_string	Format string.
...	Optional variable length list of parameters.

Description

Main formatting functions that drives the conversion process and invokes support functions. Format string is represented by the ASCII string or the wide string.

The behavior of these function is similar to the “printf” function. However format string is based on different rules. Format string is a generic free text string that contains format patterns. Each format pattern is substituted from the variable length function parameter list, which should contain enough values. Each format pattern starts with “%“ symbol. First symbol after “%“ describes the ordinal number of the actual parameter. That is one and the same parameter may occur in several patterns. The second (and the last) symbol describes the type of the corresponding data (actual parameter).

For example, pattern “%3s” tells that third parameter is an ASCII string value. Valid type specifiers are:

`'c'`	ASCII character.
`'s'`	ASCII string.
`'w'`	Wide string.
`'d'`	32-bit integer.
`'D'`	64-bit integer.
`'f'`	Float value.
`'b'`	Binary string.
`'V'`	t3rt_value_t reference.

RTL Binary String Functions

The binary string is an arbitrarily sized sequence of bits that is used by encoders/decoders and in various other situation when binary data have to be represented.

RTL Binary String Related Type Definitions

t3rt_binary_string_t

Representation for a sequence of binary data. The string will grow dynamically to the necessary size.

t3rt_binary_string_iter_t

This is an iterator used when reading data from a binary string. Used in, for example, decoders. The binary string iterator API functions have the t3rt_bstring_iter-prefix.

t3rt_binary_string_allocate

Creates an empty binary string of given size

```
t3rt_binary_string_t t3rt_binary_string_allocate
        (t3rt_alloc_strategy_t strategy,
         unsigned long alloc_length,
         t3rt_context_t context);
```

Parameters

`strategy`	Memory allocation strategy for the created string.
`alloc_length`	Initial size of the string buffer.

Description

This function creates new empty binary string. Allocation size specifies the initial size of the internal string buffer. It may be equal to zero and serves only to increase the performance of the binary string operations. Allocation size is given in number of bits.

Return Values

New instance of binary string allocated according the specified strategy.

t3rt_binary_string_deallocate

Deletes binary string data.

```
void t3rt_binary_string_deallocate
        (t3rt_binary_string_t string,
         t3rt_context_t context);
```

Parameters

`string`	Binary string to be deleted.

Description

Use `t3rt_binary_string_deallocate_all` instead. `t3rt_binary_string_deallocate` is a deprecated function and will be removed in future versions.

`t3rt_binary_string_deallocate` frees the internal string buffer used to store actual data. The memory allocated for the service structure itself should be freed manually by the user

t3rt_binary_string_deallocate_all

Fully deletes binary string freeing all allocated memory.

```
void t3rt_binary_string_deallocate_all
        (t3rt_binary_string_t * string,
         t3rt_context_t context);
```

Parameters

string	Pointer to the binary string to be deleted.

Description

t3rt_binary_string_deallocate_all fully frees the memory allocated for a binary string.

t3rt_binary_string_construct

Creates new binary string from the specified data array.

```
t3rt_binary_string_t t3rt_binary_string_construct
        (t3rt_alloc_strategy_t strategy,
         unsigned long length,
         const unsigned char *data,
         t3rt_context_t context);
```

Parameters

strategy	Memory allocation strategy for the resulting string.
length	Length of data buffer in bits.
data	Data buffer.

Description

This function creates new binary string and initializes it with the data from the provided buffer.

Return Values

New instance of binary string allocated according the specified strategy and filled with the specified data.

t3rt_binary_string_copy

Makes a copy of the string.

```
t3rt_binary_string_t t3rt_binary_string_copy
        (t3rt_binary_string_t string,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t context);
```

Parameters

string	Binary string to copy.
strategy	Memory allocation strategy for the resulting string.

Return Values

Returns copy of the given binary string allocated according the specified strategy.

t3rt_binary_string_clear

Clears the contents. This will only set the length to zero.

```
void t3rt_binary_string_clear
        (t3rt_binary_string_t string,
         t3rt_context_t context);
```

Parameters

string	Binary string to clear.

Description

This function sets the length of the binary string to zero. String buffer is not touched and memory is not released.

t3rt_binary_string_length

The number of used bits in the string.

```
unsigned long t3rt_binary_string_length
        (t3rt_binary_string_t string,
         t3rt_context_t context);
```

Parameters

string	Valid binary string.

Return Values

Returns the length of the given string in bits.

t3rt_binary_string_is_equal

Compares two bit strings for equality.

```
bool t3rt_binary_string_is_equal
        (t3rt_binary_string_t s1,
         t3rt_binary_string_t s2,
         t3rt_context_t context);
```

Parameters

s1	Valid binary string.
s1	Valid binary string.

Return Values

Returns true if binary strings are equal, false otherwise.

t3rt_binary_string_pad

Aligns the binary string by padding the last byte with zeroes if not fully used.

```
void t3rt_binary_string_pad
        (t3rt_binary_string_t string,
         t3rt_context_t context);
```

Parameters

string	Valid binary string.

Description

If the given binary string is not byte-aligned then it's appended with certain number of zero bits (from 1 to 7) to make the resulting string byte-aligned.

t3rt_binary_string_assign

Assigns one binary string to another.

```
void t3rt_binary_string_assign
        (t3rt_binary_string_t dest,
```

```
        t3rt_binary_string_t src,
        t3rt_context_t context);
```

Parameters

dest	Destination binary string.
src	Source binary string.

Description

This function assigns one binary string to another. Both strings has to be valid binary strings, allocated prior to the assignment.

t3rt_binary_string_append

t3rt_binary_string_append, t3rt_binary_string_append_1byte, t3rt_binary_string_append_2bytes, t3rt_binary_string_append_4bytes, t3rt_binary_string_append_nbytes, t3rt_binary_string_append_nbits, t3rt_binary_string_append_from_iter, t3rt_binary_string_append_bits

Append binary string from different sources.

```
void t3rt_binary_string_append
        (t3rt_binary_string_t string,
         t3rt_binary_string_t appstr,
         t3rt_context_t context);

void t3rt_binary_string_append_1byte
        (t3rt_binary_string_t string,
         unsigned char data_1,
         t3rt_context_t context);

void t3rt_binary_string_append_2bytes
        (t3rt_binary_string_t string,
         unsigned short data_2,
         t3rt_context_t context);

void t3rt_binary_string_append_4bytes
        (t3rt_binary_string_t string,
         unsigned long data_4,
         t3rt_context_t context);

void t3rt_binary_string_append_nbytes
```

```
        (t3rt_binary_string_t string,
         const unsigned char *data_n,
         unsigned long byte_size,t3rt_context_t context);

void t3rt_binary_string_append_nbits
        (t3rt_binary_string_t string,
         const unsigned char *data_n,
         unsigned long bit_size,
         t3rt_context_t context);

void t3rt_binary_string_append_from_iter
        (t3rt_binary_string_t string,
         t3rt_binary_string_iter_t *iter,
         unsigned long iter_bits,
         t3rt_context_t context);

void t3rt_binary_string_append_bits
        (t3rt_binary_string_t string,
         unsigned char data_1,
         unsigned char n_bits,
         t3rt_context_t context);
```

Parameters

string	Valid binary string.
appstr	Valid binary string.
data_1	8-bit integer (unsigned char).
data_2	16-bit integer (unsigned short).
data_4	32-bit integer (unsigned long)
data_n	Pointer to data buffer
iter	Binary string iterator
byte_size	Length of data buffer in bytes.
bit_size	Length of data buffer in bits.
iter_bits	Number of bits to extract from the iterator.
n_bits	Number of bits to extract from 8-bit integer.

Description

The above functions append given binary string with data that may be provided differently.

In `t3rt_binary_string_append` data is appended from of another binary string. The length of appended data is the length of appended binary string.

In `t3rt_binary_string_append_1byte` data is appended from given unsigned char buffer. The length of appended data is 8 bits.

In `t3rt_binary_string_append_2byte` data is appended from given unsigned short buffer. The length of appended data is 16bits.

In `t3rt_binary_string_append_4byte` data is appended from given unsigned long. The length of appended data is 32bits.

In `t3rt_binary_string_append_nbytes` data is appended from given data buffer. The length of appended data is specified by the “byte_size“ parameter in bytes (i.e. the actual length of appended data is “byte_size” * 8).

In `t3rt_binary_string_append_nbits` data is appended from given data buffer. The length of appended data is specified by the “bit_size“ parameter in bits.

In `t3rt_binary_string_append_bits` data is appended from given unsigned char. The length of appended data is specified by the “n_bits“ parameter in bits. Value of “n_bits“ may not be grater than 8.

In `t3rt_binary_string_append_iter` data is appended from given binary string iterator, which has been previously initialized using one of the binary string iterator functions (e.g. t3rt_binary_string_start). The length of appended data is specified by the “iter_bits“ parameter in bits. Value of “iter_bits“ may not be grater than remaining length of the iterator. It may be queried using t3rt_bstring_iter_remaining_room function. After invoking this function iterator is moved forward to the “iter_bits“ number of bits.

t3rt_bstring_iter_remaining_room

Returns the remaining room in the given iterator.

```
unsigned long t3rt_bstring_iter_remaining_room
        (t3rt_binary_string_iter_t *iter,
         t3rt_context_t context);
```

Parameters

`iter`	Address of the binary string iterator.

Return Values

Returns the number of bits between the current position of the iterator and the end of the binary string. Returns zero then iterator is at the end of the binary string.

t3rt_binary_string_start

Sets the iterator at the beginning of the string.

```
void t3rt_binary_string_start
        (t3rt_binary_string_t string,
         t3rt_binary_string_iter_t *iter,
         t3rt_context_t context);
```

Parameters

string	Valid binary string.
iter	Address of the binary string iterator.

Description

This function initializes iterator and sets it to the start of given binary string.

Example Usage

```
t3rt_binary_string_iter_t iter;
t3rt_binary_string_start(bstring, &iter, ctx);
t3rt_decode(&iter, value_type, t3rt_temp_alloc_c, &value,
ctx);
```

t3rt_binary_string_set_at

Sets the iterator to the indicated bit position of the string.

```
void t3rt_binary_string_set_at
        (t3rt_binary_string_t string,
         unsigned long index,
         t3rt_binary_string_iter_t *iter,
         t3rt_context_t context);
```

Parameters

`string`	Valid binary string.
`index`	Position inside binary string.
`iter`	Address of the binary string iterator.

Description

This function initializes iterator and sets it to the specified position inside binary string. Position should be within the boundaries of the given string.

t3rt_bstring_iter_forward_nbits

Moves the iterator forward.

```
void t3rt_bstring_iter_forward_nbits
        (t3rt_binary_string_iter_t *iter,
         unsigned long n_bits,
         t3rt_context_t context);
```

Parameters

`iter`	Address of the binary string iterator.
`n_bits`	Offset in bits

Description

This function moves iterator forward to the specified number of bits. After moving iterator should point to the valid position within binary string.

t3rt_bstring_iter_backward_nbits

Moves the iterator backwards.

```
void t3rt_bstring_iter_backward_nbits
        (t3rt_binary_string_iter_t *iter,
         unsigned long n_bits,
         t3rt_context_t context);
         t3rt_context_t context);
```

Parameters

iter	Address of the binary string iterator.
n_bits	Offset in bits

Description

This function moves iterator backward to the specified number of bits. After moving iterator should point to the valid position within binary string.

t3rt_bstring_iter_next_byte

Moves the iterator forwards to the start of the next byte.

```
void t3rt_bstring_iter_next_byte
        (t3rt_binary_string_iter_t *iter,
         t3rt_context_t context);
```

Parameters

iter	Address of the binary string iterator.

Description

This function moves the iterator forward to the start of the next byte. If the current position is at a start of a byte, the iterator will move one byte forward.

t3rt_bstring_iter_is_at_boundary

Checks if the iterator is positioned at the beginning of a byte.

```
bool t3rt_bstring_iter_is_at_boundary
        (t3rt_binary_string_iter_t *iter,
         t3rt_context_t context);
```

Parameters

iter	Address of the binary string iterator.

Description

This function checks if the iterator is positioned at the beginning of a byte, i.e. that it is byte aligned.

Return Values

Returns true if iterator is byte aligned, false otherwise.

t3rt_bstring_iter_bits_to_byte_boundary

Returns number of bits left to the next byte boundary.

```
unsigned char t3rt_bstring_iter_bits_to_byte_boundary
        (t3rt_binary_string_iter_t *iter,
         t3rt_context_t context);
```

Parameters

`iter`	Address of the binary string iterator.

Return Values

Returns integer from 0 to 7 that is the number of bits between current iterator position and next byte boundary.

t3rt_bstring_iter_at_end, t3rt_bstring_iter_at_start

Predicates to check if the iterator is positioned at the end or beginning of the string.

```
bool t3rt_bstring_iter_at_end
        (t3rt_binary_string_iter_t *iter,
         t3rt_context_t context);

bool t3rt_bstring_iter_at_start
        (t3rt_binary_string_iter_t *iter,
         t3rt_context_t context);
```

Parameters

`iter`	Address of the binary string iterator.

Return Values

`t3rt_bstring_iter_at_end` returns true if iterator is positioned at the end of binary string, false otherwise.

`t3rt_bstring_iter_at_start` returns true if iterator is positioned at the start of binary string, false otherwise.

t3rt_bstring_iter_is_bit_set

Checks if the bit at the current position is set.

```
bool t3rt_bstring_iter_is_bit_set
        (t3rt_binary_string_iter_t *iter,
         t3rt_context_t context);
```

Parameters

`iter`	Address of the binary string iterator.

Return Values

Returns true if bit is set (i.e. equals to 1) at the current iterator position, false otherwise.

t3rt_bstring_iter_get_bits

t3rt_bstring_iter_get_bits, t3rt_bstring_iter_get_1byte, t3rt_bstring_iter_get_2bytes, t3rt_bstring_iter_get_4bytes, t3rt_bstring_iter_get_nbytes, t3rt_bstring_iter_get_nbits

Extracts data from the binary string.

```
unsigned char t3rt_bstring_iter_get_bits
        (t3rt_binary_string_iter_t *iter,
         unsigned char n_bits,
         t3rt_context_t context);

unsigned char t3rt_bstring_iter_get_1byte
        (t3rt_binary_string_iter_t *iter,
         t3rt_context_t context);

unsigned short t3rt_bstring_iter_get_2bytes
        (t3rt_binary_string_iter_t *iter,
         t3rt_context_t context);

unsigned long t3rt_bstring_iter_get_4bytes
        (t3rt_binary_string_iter_t *iter,
         t3rt_context_t context);

void t3rt_bstring_iter_get_nbytes
        (t3rt_binary_string_iter_t *iter,
         unsigned long byte_size,
```

```
          unsigned char *data,
          t3rt_context_t context);

void t3rt_bstring_iter_get_nbits
        (t3rt_binary_string_iter_t *iter,
          unsigned long bit_size,
          unsigned char *data,
          t3rt_context_t context);
```

Parameters

iter	Address of the binary string iterator.
n_bits	Number of bits to return (n_bits <= 8).
byte_size	Amount of bytes to extract.
bit_size	Amount of bits to extract.
data	Pointer to data buffer.

Description

The above functions extracts specified amount of data from the binary string and return it in this or tat way. Starting position is identified by the current position of the iterator. The remaining room of the iterator (see t3rt_bstring_iter_remaining_room) should be greater or equal to the length of the extracted data. After extracting data iterator is moved forward.

`t3rt_bstring_iter_get_bits` extracts 0 to 8 bits from the string and returns them as unsigned char value.

`t3rt_bstring_iter_get_1byte` extracts 8 bits from the string and returns them as unsigned char value.

`t3rt_bstring_iter_get_2bytes` extracts 16 bits from the string and returns them as unsigned short value.

`t3rt_bstring_iter_get_4bytes` extracts 32 bits from the string and returns them as unsigned long value.

`t3rt_bstring_iter_get_nbytes` extracts specified number of bytes from the string and puts them into given buffer.

`t3rt_bstring_iter_get_nbits` extracts specified number of bits from the string and puts them into given buffer.

Return Values

`t3rt_bstring_iter_get_bits` returns extracted 0-8 bits as unsigned char value.

`t3rt_bstring_iter_get_1byte` returns extracted 8 bits as unsigned char value.

`t3rt_bstring_iter_get_2bytes` extracts returns extracted 16 bits as unsigned short value.

`t3rt_bstring_iter_get_4bytes` returns extracted 32 bits as unsigned long value.

RTL Codecs Functions

RTL Codecs Related Type Definitions

t3rt_codecs_init_function_t

This type of function is registered in "t3rt_codecs_register" on page 474 and invoked when the codecs system must be initialized.

t3rt_codecs_setup_function_t

This type of function is registered in "t3rt_codecs_register" on page 474 and called repeatedly to set up codecs functions for a given TTCN-3 type.

t3rt_codecs_result_t

This represents the return status of the encoder and decoder functions. It signifies that the operation was not applicable, failed, or successful. The symbols used are: t3rt_codecs_result_not_applicable_c, t3rt_codecs_result_failed_c or t3rt_codecs_result_succeeded_c respectively.

t3rt_encoder_function_t

This is the function prototype that all encoder function must have. A function of this prototype is associated with a TTCN-3 type by the registered setup function.

t3rt_decoder_function_t

This is the function prototype that all decoder function must have. A function of this prototype is associated with a TTCN-3 type by the registered setup function.

t3rt_codecs_register

Registers a codecs system to the RTS to provide encoder and decoder functions to all or a subset of the types in the TTCN-3 modules.

```
void t3rt_codecs_register
        (t3rt_codecs_init_function_t init_function,
         t3rt_codecs_setup_function_t setup_function,
         t3rt_context_t ctx);
```

Parameters

`init_function`	The initialization function of the codecs system. This will be called once (for each process)
`setup_function`	This setup function will be called once for every existing type that has not been setup already

t3rt_encode

Encode a value with the available encoder function of the type.

```
t3rt_codecs_result_t t3rt_encode
        (t3rt_value_t value,
         t3rt_binary_string_t encoded_data,
         t3rt_context_t ctx);
```

Parameters

`value`	Any TTCN-3 value to be encoded. This does not apply to values that can not be passed.
`encoded_data`	The inout value container where the encoded data will be appended. This container will grow to the necessary size and can be allocated with zero length. See the t3rt_binary_string_allocate functions.

Description

Encodes the value into a binary string allocated according to the encoding strategy.

Note

This function conforms to the "t3rt_encoder_function_t" on page 473 prototype but should never be set as encoder function for a type by a codecs system's setup function ("t3rt_codecs_setup_function_t" on page 473). Doing so will cause a stack overflow.

Return Values

Compare with "t3rt_codecs_result_t" on page 473 for applicable values.

t3rt_decode

Decode an encoded binary string into the proper TTCN-3 RTS value using the available decoder function of the type.

```
t3rt_codecs_result_t t3rt_decode
        (t3rt_binary_string_iter_t * encoded_data,
         t3rt_type_t type,
         t3rt_alloc_strategy_t strategy,
         t3rt_value_t * decoded_value,
         t3rt_context_t ctx);
```

Parameters

`encoded_data`	A binary string iterator to traverse the binary data.
`type`	The expected type for this decoding. Encoding attributes associated with this type can be extracted from it.
`decoded_value`	The inout result of the decoding that should be filled in if the decoding was successful.

Description

Decodes the data into a value.

Note

This function conforms to the "t3rt_decoder_function_t" on page 473 prototype but should never be set as decoder function for a type by a codecs system's setup function ("t3rt_codecs_setup_function_t" on page 473). Doing so that will cause a stack overflow.

Return Values

Compare with "t3rt_codecs_result_t" on page 473 for applicable values.

See also

"RTL Binary String Functions" on page 459

t3rt_tci_encode

Envelop function for the tciEncode.

```
t3rt_codecs_result_t t3rt_tci_encode
        (t3rt_value_t value,
         t3rt_binary_string_t encoded_data,
         t3rt_context_t ctx);
```

Parameters

`value`	Any TTCN-3 value to be encoded. This does not apply to values that can not be passed.
`encoded_data`	The inout value container where the encoded data will be appended. This container will grow to the necessary size and can be allocated with zero length. See the t3rt_binary_string_allocate functions.

Description

This is a conversion function between t3rt_encode RTS encoding function and tciEncode TCI encoding function. It should be registered by the user for all types that are encoded using tciEncode.

Note

Ensure that you compiled test system with TCI support enabled. See compiler command line option for more info.

Return Values

Compare with “t3rt_codecs_result_t” on page 473 for applicable values.

t3rt_tci_decode

Envelop function for the tciDecode.

```
t3rt_codecs_result_t t3rt_tci_decode
        (t3rt_binary_string_iter_t * encoded_data,
         t3rt_type_t type,
         t3rt_alloc_strategy_t strategy,
         t3rt_value_t * decoded_value,
         t3rt_context_t ctx);
```

Parameters

encoded_data	A binary string iterator to traverse the binary data.
type	The expected type for this decoding. Encoding attributes associated with this type can be extracted from it.
decoded_value	The inout result of the decoding that should be filled in if the decoding was successful.

Description

This is a conversion function between t3rt_decode RTS decoding function and tciDecode TCI decoding function. It should be registered by the user for all types that are decoded using tciDecode.

Note

Ensure that you compiled test system with TCI support enabled. See compiler command line option for more info.

Return Values

Compare with “t3rt_codecs_result_t” on page 473 for applicable values.

RTL Error Handling Functions

RTL Error Handling Related Type Definitions

t3rt_error_description_t

This is a composition of information about an encountered error, used in the t3rt_report_fatal_system_error function. The information is explicitly provided since the current state of the system can not be trusted.

t3rt_report_error

Report test case error and terminate the execution of the test case.

```
void t3rt_report_error
        (unsigned long line,
         const char* file,
         t3rt_wide_string_t err_msg,
         t3rt_context_t ctx);
```

Parameters

`line`	This is the line in the file at which point the error occurred.
`file`	This is the source file in which the error occurred.
`err_msg`	This is the error message, represented by a wide string (possibly localized).

Description

This function formats and sends a log message about the error, followed by propagating the error verdict. This will have the effect that all running components to shut down and the current test case to terminate.

The system will continue execution for the remainder of the control part.

This is the function to be used in implementation of the integration layer, encoders/decoders, log mechanisms, and so on to report an error from which the system can recover and continue execution of the control part.

If an unrecoverable situation has occurred, the function "t3rt_report_fatal_system_error" on page 479 should be used that will terminate execution of the whole test suite.

t3rt_report_fatal_system_error

Report an unrecoverable error and terminate the execution of the whole system.

```
void t3rt_report_fatal_system_error
        (t3rt_error_description_t err,
         t3rt_context_t context);
```

Parameters

err	The error description.

Description

This function logs the error description and then terminate execution without attempting to shut anything down (other components, for example) gracefully.

RTL Execution Control

These functions are controlling the initialization, execution and finalization (clean-up) of the test suite execution. They are only intended to be called from the code generated by the Rational Systems Tester Compiler.

RTL Execution Control Related Type Definitions

t3rt_module_register_function_t

This is a function type that, when invoked, should take care of the registering of a TTCN-3 module, recursively through the imported modules. This function is automatically generated by the Compiler and used in module registering.

t3rt_control_part_function_t

The control part function is a function that, when invoked, execute a control part of a TTCN-3 module. Such a function is generated by the Compiler for each module and the root module's function will be used as default in the execution of the test suite.

t3rt_snapshot_return_t

A value of this type is returned from the t3pl_component_wait function and should tell whether a timeout occurred (t3rt_snapshot_return_timeout), data was detected (t3rt_snapshot_return_data_received) or both at the same time (t3rt_snapshot_return_timeout_and_data_received).

t3rt_run_test_suite

Executes the control part of the test suite's root module. This is the entry point to the RTS.

```
void t3rt_run_test_suite
        (int argc,
         char * argv [],
         t3rt_module_register_function_t root_module_func,
         t3rt_control_part_function_t control_part_func);
```

Parameters

`argc`	The number of command-line arguments.
`argv`	The command-line arguments to the ETS.
`root_module_func`	The function pointer to the (generated) register function of the root module.
`control_part_func`	The function pointer to the (generated) function that executes the root modules control part.

Description

This function encapsulates the whole procedure of initialization, execution, and finalization of the test execution.

First, the runtime engine modules are pre-initialized with the command-line information (in argc and argv). Then, the root module is registered using the provided registration function.

All the loaded modules are initialized (including module parameters initialization) starting from the root module.

After this, the specified control part will be executed. The predefined constant `t3rt_root_control_part_c` can be specified to run the control part of the registered root module.

Last, cleanup is performed (using the generated "finalize" functions).

t3rt_exit

Aborts execution by following the proper shutdown procedures.

```
void t3rt_exit (void);
```

Description

Aborts execution by following the proper shutdown procedures.

The context of the control part component will be internally available. It is necessary for sending messages to the control part component and to find the list of known components.

Shutdown messages will be sent to all known components followed by calling 't3pl_task_kill' with appropriate arguments.

This will also terminate the control part component.

t3rt_abort

Aborts execution abruptly.

```
void t3rt_abort (void);
```

Description

Aborts execution skipping proper shutdown procedures.

This will also terminate the control part component.

RTL Memory Functions

RTL Memory Related Type Definitions

t3rt_alloc_strategy_t

This type is used when functions potentially have to allocate memory to perform its task, or, when you explicitly allocate new values and so on. To allocate memory in temporary memory, use t3rt_temp_alloc_c and to allocate in permanent (heap) memory use t3rt_perm_alloc_c.

t3rt_memory_scope_t

This is a position in a temporary memory area that is used for optimized de-allocation.

t3rt_memory_temp_begin

Saves the current “next allocation position” in the temporary memory area.

```
void t3rt_memory_temp_begin
        (t3rt_memory_scope_t *memory_scope,
         t3rt_context_t ctx);
```

Parameters

memory_scope	A storage variable for a new temporary memory scope. This is an inout parameter.

Description

This function creates a new memory scope in which allocation are made until it is closed by “t3rt_memory_temp_end” on page 482.

This function is intended to be used when a new temporary memory scope is desired, that will be closed to release the memory allocated.

The memory position is a structured object that is intended to be allocated on the stack.

Example Usage

```
{
  t3rt_memory_scope_t mscope;

  t3rt_memory_temp_begin(&mscope, ctx);
  /* Processing that make allocation using the
     t3rt_temp_alloc_c allocation strategy. */
  t3rt_memory_temp_end(&mscope, ctx);
}
```

t3rt_memory_temp_end

Closes (and virtually deallocates) a previously created memory scope.

```
void t3rt_memory_temp_end
        (t3rt_memory_scope_t *memory_scope,
         t3rt_context_t ctx);
```

Parameters

memory_scope	The previously created memory scope.

Description

This function closed the given memory scope which will virtually deallocate the memory allocated since the scope was created.

This is to be treated as “freeing” the temporary memory and subsequent temporary memory allocations will overwrite any residing data.

If poison pilling is enabled, the memory blocks in the scope will be overwritten with a pattern signifying that the memory has been deallocated.

Example Usage

```
{
  t3rt_memory_scope_t mscope;

  t3rt_memory_temp_begin(&mscope, ctx);
  /* Processing that make allocation using the
     t3rt_temp_alloc_c allocation strategy. */
  t3rt_memory_temp_end(&mscope, ctx);
}
```

t3rt_memory_temp_clear

Closes (and virtually deallocates) a previously created memory scope.

```
void t3rt_memory_temp_clear
        (t3rt_memory_scope_t *memory_scope,
         t3rt_context_t ctx);
```

Parameters

memory_scope	The previously created memory scope.

Description

This function clears the current scope, that is, it virtually deallocates the memory allocated since the scope was created.

This is to be treated as “freeing” the temporary memory and subsequent temporary memory allocations will overwrite any residing data.

If poison pilling is enabled, the memory blocks in the scope will be over-written with a pattern signifying that the memory has been deallocated.

Example Usage

```
{
  t3rt_memory_scope_t mscope;

  t3rt_memory_temp_begin(&mscope, ctx);
  /* Processing that make allocation using the
     t3rt_temp_alloc_c allocation strategy. */
  t3rt_memory_temp_clear(&mscope, ctx);
  /* Processing that make allocation using the
     t3rt_temp_alloc_c allocation strategy. */
  t3rt_memory_temp_end(&mscope, ctx);
}
```

t3rt_memory_temp_allocate

Allocates temporary memory in current memory scope.

Parameters

`size`	Size of allocated block in bytes.

```
void * t3rt_memory_temp_allocate
        (unsigned long size,
         t3rt_context_t context);
```

Description

Allocates size number of bytes in the temporary memory area.

Return Values

Returns pointer to allocated block. Returns NULL in case of error.

RTL Source Tracking Functions

RTL Source Tracking Related Type Definitions

t3rt_source_location_t

This entity represents a location in a TTCN-3 source file (or in any language source file that can be expressed with file name and line number.

A stack of objects of this type represents the call-stack during execution.

It is passed to the log mechanisms for all events to enable the logging to point to the exact location where something happened.

t3rt_scope_kind_t

These enumeration values signifies the kind of scope that the source location represents.

```
t3rt_scope_function_c
t3rt_scope_external_function_c
t3rt_scope_testcase_c
t3rt_scope_teststep_c
t3rt_scope_control_part_c
t3rt_scope_undefined_c
```

t3rt_targetcode_location_push

Pushes a new target code location on the stack.

```
void t3rt_targetcode_location_push
        (const char *scope_name,
         const char *file_name,
         unsigned long line_number,
         t3rt_context_t ctx);
```

Parameters

`scope_name`	Name of an entered scope, as a C function, for example.
`file_name`	Name of the target code file.
`line_number`	Line in the target code file.

Description

This sets the target code location and stores it within the component.

A symmetric “t3rt_targetcode_location_pop” on page 486 must be made after a push at all returning point in the scope or the location information will be inconsistent.

t3rt_targetcode_location_set_line

Updates the line number of the pushed source code object on the stack.

```
void t3rt_targetcode_location_set_line
        (unsigned long line,
         t3rt_context_t ctx);
```

Parameters

`line`	The new line number.

Description

This sets the line number of the topmost target code location object.

t3rt_targetcode_location_pop

Removes a previously pushed target code location from the stack.

```
void t3rt_targetcode_location_pop(t3rt_context_t ctx);
```

Description

This removes one target code location element from the source location stack.

A symmetric “t3rt_targetcode_location_push” on page 485 must be made prior to this or the location information will be inconsistent.

t3rt_targetcode_location_get

Returns location created by the last call to t3rt_targetcode_location_push.

```
t3rt_source_location_t t3rt_targetcode_location_get
(t3rt_context_t ctx);
```

Description

This function only retrieves the topmost, target code related, source location object.

Return Values

The topmost source location object. If not found a static object is returned, representing the “unknown” location.

t3rt_source_tracking_top

Retrieves the top element pushed on the source location stack without removing it.

```
t3rt_source_location_t t3rt_source_tracking_top
(t3rt_context_t ctx);
```

Description

Retrieves the location on the top of the source tracking stack independent of the type of the top source location object. To retrieve a specific kind of source location, use “t3rt_targetcode_location_get” on page 486.

Return Values

The topmost source location. If the stack is empty, a test case error will be generated and the execution will terminate.

t3rt_source_location_module_name

Returns the module name.

```
const char* t3rt_source_location_module_name
        (t3rt_source_location_t location,
         t3rt_context_t ctx);
```

Parameters

location	Source location object.

Return Values

Returns module name of the given source location object.

t3rt_source_location_scope_name

Retrieves the scope name.

```
const char* t3rt_source_location_scope_name
        (t3rt_source_location_t location,
         t3rt_context_t ctx);
```

Parameters

location	Source location object.

Return Values

Returns scope (usually function, testcase or altstep) name of the given source location object.

t3rt_source_location_scope_arguments

Returns the scope arguments.

```
t3rt_value_t* t3rt_source_location_scope_arguments
        (t3rt_source_location_t location,
         t3rt_context_t ctx);
```

Parameters

location	Source location object.

Return Values

Returns the (null-terminated) vector of scope arguments from a source location object.

t3rt_source_location_scope_kind

Retrieves the scope kind.

```
t3rt_scope_kind_t t3rt_source_location_scope_name
        (t3rt_source_location_t location,
         t3rt_context_t ctx);
```

Parameters

location	Source location object.

Return Values

Returns the scope kind from a given source location object.

t3rt_source_location_file_name

Retrieves the file name.

```
const char* t3rt_source_location_file_name
        (t3rt_source_location_t location,
         t3rt_context_t ctx);
```

Parameters

location	Source location object.

Return Values

Returns the file name from a given source location object.

t3rt_source_location_file_line

Retrieves the line number.

```
unsigned long t3rt_source_location_file_line
        (t3rt_source_location_t location,
         t3rt_context_t ctx);
```

Parameters

location	Source location object.

Return Values

Returns the line number in the source file from a given source location object.

t3rt_source_location_is_ttcn3

Check whether the source location is a TTCN-3 source location or not.

```
bool t3rt_source_location_is_ttcn3
        (t3rt_source_location_t location,
         t3rt_context_t ctx);
```

Parameters

location	Source location object.

Return Values

Returns true if source location represents location in a TTCN-3 file, false if it's a target code location.

RTL Symbol Table Functions

One symbol table is generated statically per TTCN-3 module. Elements can not be added dynamically during runtime. The intended usage is only to provide uniform access to the declared entities in the test suite modules.

RTL Symbol Table Related Type Definitions

t3rt_symbol_entry_t

This is an element of a module's symbol table. Its structure is only public (that is, the fields are visible and can be accessed) because the entries are statically generated by the Compiler. Use the access function below to access the information.

t3rt_symbol_entry_kind_t

This is an enumeration of the different kinds of which a symbol table entry can be. It can be any of the following:

```
t3rt_symbol_entry_kind_module
t3rt_symbol_entry_kind_imported_module
t3rt_symbol_entry_kind_group
t3rt_symbol_entry_kind_userdefined_type
t3rt_symbol_entry_kind_signature_type
t3rt_symbol_entry_kind_template_type
t3rt_symbol_entry_kind_constant
t3rt_symbol_entry_kind_external_constant
t3rt_symbol_entry_kind_module_parameter
```

```
t3rt_symbol_entry_kind_initialize_function
t3rt_symbol_entry_kind_module_params_initialize_function
t3rt_symbol_entry_kind_finalize_function
t3rt_symbol_entry_kind_external_function
t3rt_symbol_entry_kind_control_part_function
t3rt_symbol_entry_kind_function
t3rt_symbol_entry_kind_test_step
t3rt_symbol_entry_kind_testcase
```

t3rt_find_element

Finds the element in the symbol table of the specified module and returns it.

```
t3rt_symbol_entry_t t3rt_find_element
        (const char *module_name,
         const char *element_name,
         t3rt_context_t context);
```

Parameters

module_name	Search is performed in the symbol table of this module.
element_name	Object to search for.

Description

This function searches for element using its name in symbol table of the specified module. Module name may be an empty string (i.e. ““) thus telling RTS to search in the symbol table of the root module. Using NULL as the value of module name results in test case error.

Each imported object is represented in the symbol table of importing module either by one or two entries. In most cases there are two entries, one entry with fully qualified name (<name of imported module>.<name of imported object>) and one entry with only object name. If two objects that are imported from separate modules have same names then each of them is represented in the symbol table of importing module only by one entry with the name given in fully qualified format. It means that care should be taken when searching for imported objects.

Return Value

The symbol table entry if found, otherwise NULL is returned.

t3rt_root_module_name

Returns the name of the root module.

```
const char * t3rt_root_module_name (t3rt_context_t
context);
```

Description

Usually root module is set automatically by RTS according to the specified root module registration function passed to the t3rt_run_test_suite. However it may be set manually when using TCI test management by calling tciRoot-Module function.

Return Values

Returns the name of the current root module.

t3rt_symbol_table_entry_name

Access the symbol table entry name.

```
const char* t3rt_symbol_table_entry_name
        (t3rt_symbol_entry_t entry,
         t3rt_context_t ctx);
```

Parameters

entry	The symbol table entry.

Description

This is always the name of the symbol for which this is an entry.

Return Value

The name of the symbol or the empty string (“”) if the entry is invalid.

t3rt_symbol_table_entry_kind

Access the kind of symbol table entry.

```
t3rt_symbol_entry_kind_t t3rt_symbol_table_entry_kind
        (t3rt_symbol_entry_t entry,
```

```
        t3rt_context_t ctx);
```

Parameters

entry	The symbol table entry.

Description

This retrieves the kind of symbol table entry.

Return Value

See t3rt_symbol_entry_kind_t type for value set.

t3rt_symbol_table_entry_type

Access the type descriptor of the symbol table entry.

```
t3rt_type_t t3rt_symbol_table_entry_type
        (t3rt_symbol_entry_t entry,
         t3rt_context_t ctx);
```

Parameters

entry	The symbol table entry.

Description

This is only applicable to entries for user-defined types, signature types and templates.

Return Value

The type of the symbol or t3rt_undefined_type type constant if the entry is invalid.

t3rt_symbol_table_entry_value

Access the value for the symbol table entry.

```
t3rt_value_t t3rt_symbol_table_entry_value
        (t3rt_symbol_entry_t entry,
         t3rt_context_t ctx);
```

Parameters

`entry`	The symbol table entry.

Description

This is only applicable to constants and external constants.

Return Value

The associated constant value or the `t3rt_no_value_c` value constant if the entry is invalid.

t3rt_symbol_table_entry_function

Access the function information of the symbol table entry.

```
void* t3rt_symbol_table_entry_function
        (t3rt_symbol_entry_t entry,
         t3rt_context_t ctx);
```

Parameters

`entry`	The symbol table entry.

Description

This is only applicable for function-related symbols like test cases, functions, test steps, and so on. This is where the pointer to the generated C function is stored.

To be used, it has to be explicitly casted to the appropriate function. It is only intended to be used by the Rational Systems Tester Compiler.

Return Value

The ("voidified") function pointer of the symbol or NULL if the entry is invalid.

t3rt_symbol_table_entry_attribute

Access the attribute information of the symbol table entry.

```
void* t3rt_symbol_table_entry_attribute
        (t3rt_symbol_entry_t entry,
         t3rt_context_t ctx);
```

Parameters

entry	The symbol table entry.

Description

This field is currently not used and should not be necessary to access. It is for future extensibility in this area.

Return Value

Currently always NULL.

t3rt_symbol_table_entry_auxiliary

Access the auxiliary information of the symbol table entry.

```
void* t3rt_symbol_table_entry_auxiliary
        (t3rt_symbol_entry_t entry,
         t3rt_context_t ctx);
```

Parameters

entry	The symbol table entry.

Description

The auxiliary field is currently not used and should not be necessary to access. It is only intended for future extensions.

Return Value

A pointer to the auxiliary data or NULL if not present or if the entry is invalid.

RTL Miscellaneous Functions

t3rt_rtconf_get_param

Returns the value of configuration parameter name.

```
t3rt_value_t t3rt_rtconf_get_param
        (const char *param_name,
         t3rt_context_t ctx);
```

Parameters

param_name	RTConf parameter name.

Description

This function queries the value of the specified key from the RTConf configuration table.

Return Values

If the key is illegal, `t3rt_illegal_value_c` will be returned and if the value is not present, the `t3rt_no_value_c` is returned.

t3rt_rtconf_set_param

Sets the configuration parameter param_name to the provided value.

```
void t3rt_rtconf_set_param
        (const char *param_name,
         t3rt_value_t param_value,
         t3rt_context_t ctx);
```

Parameters

param_name	RTConf parameter name.
parameter_value	Parameter value to set.

Description

If the key exists, the current value will be overwritten, and no warning will be issued when this happens. If the param_value is blank (that is, NULL since it is really a pointer) the `t3rt_no_value_c` will be inserted.

During a test case execution this function will not be allowed to modify the information in runtime configuration, it will be a no-op.

t3rt_register_default_logging

Called during initialization to register the built-in log mechanism(s).

```
void t3rt_register_default_logging(void);
```

Description

This is located in a source file `t3rts_conditional.c` just to make it possible to remove the built-in log mechanism(s) at compile time when building the ETS.

If the T3RT_NO_BUILTIN_LOG symbol is set when compiling, the built-in log mechanism will be disabled.

Look in the file to see what compilation symbols can be used to accomplish this.

t3rt_register_provided_logging

Called during initialization to register the provided log mechanisms (apart from the built-in log).

```
void t3rt_register_provided_logging(void);
```

Description

This is located in a source file `t3rts_conditional.c` just to make it possible to enable the provided log mechanisms at compile time when building the ETS.

If the T3RT_MSC96_EVENT_LOG symbol is set when compiling, the MSC-96 log mechanism will be enabled.

If the T3RT_DEBUG symbol is set when compiling, the TTCN-3 real-time debugger will be enabled.

Look in the file to see what compilation symbols can be used to accomplish this.

t3rt_context_get_component_type

Retrieve the component type of the component of this runtime context.

```
t3rt_type_t t3rt_context_get_component_type (t3rt_context_t
ctx);
```

Description

This can be used to access the component type information from a context.

Return Value

The type descriptor for the component.

t3rt_context_get_component_address

Retrieve the component control port address of the component of this runtime context.

```
t3rt_binary_string_t t3rt_context_get_component_address
(t3rt_context_t ctx);
```

Description

This can be used to access the component (control port) address from a context.

This address is used to control running components and is unique for all components. It can be useful as an identifier for the component instance.

Return Value

The binary string containing the component control port address.

t3rt_context_get_component_name

Retrieve the name of the component of this runtime context.

```
const char* t3rt_context_get_component_name (t3rt_context_t
ctx);
```

Description

This can be used to access the component name from a context. This is the name provided at the component create operation. If name is not provided explicitly then system generates and assigns unique name.

This name is used in logging to identify component.

Return Value

The character string with the component name.

t3rt_set_epsilon_double

Set the constant for floating point value comparison.

```
void t3rt_set_epsilon_double
        (double epsilon,
         t3rt_context_t context);
```

Parameters

`epsilon`	The value to be used when comparing floating point value. See configuration key "t3rt.values.limits.epsilon_double" on page 130 for detailed description.

Description

Sets the constant value to be used when comparing floating point values.

Instead of calling this function from user-defined code, the configuration `t3rt.values.limits.epsilon_double` key can be applied for the execution of the ETS.

t3rt_epsilon_double

Retrieve the constant used in floating point value comparison.

```
double t3rt_epsilon_double(t3rt_context_t context);
```

Description

Sets the constant value to be used when comparing floating point values.

This value can be explicitly set by calling the function t3rt_set_epsilon_double, or by setting the configuration key t3rt.values.limits.epsilon_double.

Return Value

If the value is not configured explicitly the value returned is equal to 2*DBL_EPSILON from the <limits.h> definitions.

t3rt_value_to_string, t3rt_value_to_wide_string

Prints value into ASCII or wide string.

```
const char * t3rt_value_to_string
        (t3rt_value_t value,
         t3rt_context_t ctx);

t3rt_wide_string_t t3rt_value_to_wide_string
        (t3rt_value_t value,
         t3rt_context_t ctx);
```

Parameters

value	Value to be printed.

Description

This functions may be used to print value into ASCII or wide string. It's not assumed that ASCII representation conforms to the TTCN-3 representation of the given value. The intended use of this function is in custom log mechanisms.

Return Values

Returns free text representation of the specified value.

RTL Function for Generated Code Only

There are a number of functions that can be encountered in the public RTL interface that are only intended to be used in the code generated from the Rational Systems Tester TTCN-3 Compiler. Those functions are listed here.

Important!

Using any of these functions from your own code can cause the ETS to behave in an undefined way! Contact your Rational Systems Tester support organization if you need to use them.

- t3rt_type_instantiate_template
- t3rt_type_instantiate_dynamic_template
- t3rt_type_instantiate_named_dynamic_template
- t3rt_type_instantiate_external_value
- t3rt_type_check_builtin
- t3rt_type_check_char_range
- t3rt_type_check_universal_char_range
- t3rt_type_check_port_message
- t3rt_template_match
- t3rt_template_match_signature
- t3rt_value_set_null
- t3rt_value_set_address_value
- t3rt_value_get_address_value
- t3rt_value_set_external_value
- t3rt_value_get_external_value
- t3rt_value_set_timer_default_duration
- t3rt_value_set_timer_in_array_default_duration
- t3rt_valueof
- t3rt_valueof_signature
- t3rt_value_init
- t3rt_value_init_vector_element
- t3rt_value_init_vector_element_partial
- t3rt_value_assign_vector_element_partial
- t3rt_value_assign_and_log
- t3rt_value_try_field_by_index
- t3rt_match
- t3rt_match_signature
- t3rt_recordof_match

- t3rt_setof_match
- t3rt_subset_match
- t3rt_superset_match
- t3rt_regexp_match
- t3rt_regexp_match_substring
- t3rt_match_continue_on_fail
- t3rt_timer_start
- t3rt_timer_stop
- t3rt_timer_read
- t3rt_timer_is_running
- t3rt_timer_is_timed_out
- t3rt_timer_try_timed_out
- t3rt_component_create
- t3rt_component_execute
- t3rt_component_start
- t3rt_component_stop
- t3rt_component_kill
- t3rt_component_is_running
- t3rt_component_is_alive
- t3rt_component_try_done
- t3rt_component_try_killed
- t3rt_component_try_else
- t3rt_component_snapshot
- t3rt_component_control
- t3rt_component_wait
- t3rt_component_connect_port
- t3rt_component_map_port
- t3rt_component_disconnect_port
- t3rt_component_unmap_port
- t3rt_component_set_system_component_type
- t3rt_port_clear
- t3rt_port_start

- t3rt_port_stop
- t3rt_port_halt
- t3rt_port_is_enabled
- t3rt_port_sut_action
- t3rt_port_send
- t3rt_port_call
- t3rt_port_reply
- t3rt_port_raise
- t3rt_port_try_receive
- t3rt_port_try_trigger
- t3rt_port_try_getcall
- t3rt_port_try_getreply
- t3rt_port_try_catch
- t3rt_port_try_catch_timeout
- t3rt_builtin_encode
- t3rt_builtin_decode
- t3rt_module_register
- t3rt_call_function
- t3rt_call_external_function
- t3rt_call_altstep
- t3rt_activate
- t3rt_deactivate
- t3rt_activation_list_invoke
- t3rt_retrieve_module_parameter
- t3rt_log_list_to_all
- t3rt_log_template_mismatch_by_name_event
- t3rt_log_template_mismatch_by_index_event
- t3rt_log_template_mismatch_event
- t3rt_log_alt_entered_event
- t3rt_log_alt_left_event
- t3rt_log_alt_rejected_event
- t3rt_log_alt_else_event

- t3rt_log_alt_defaults_event
- t3rt_log_alt_repeat_event
- t3rt_log_alt_wait_event
- t3rt_log_variable_modified_event
- t3rt_source_location_push
- t3rt_source_location_pop
- t3rt_source_location_push_block
- t3rt_source_location_pop_block
- t3rt_source_location_set_line
- t3rt_source_location_get
- t3rt_get_source_location
- t3rt_source_tracking_register_value
- t3rt_source_tracking_find_value
- t3rt_template_set_value
- t3rt_template_set_value_range
- t3rt_template_set_value_list
- t3rt_template_set_length_constraint
- t3rt_template_set_string_pattern
- t3rt_template_set_permutation
- t3rt_templateof
- t3rt_int2charstr
- t3rt_int2unicharstr
- t3rt_str2int_zero
- t3rt_str2float_zero
- all functions having two undescores after “t3rt” prefix (i.e. t3rt__XYZ))

Platform Layer API

This is the interface that provides the services needed by the RTS. All functions have to be implemented when creating a PL-based integration (in difference to a TRI based integration).

A working implementation, called the “Example integration”, is provided in the distribution and can be found in the /integrations/example directory in the Rational Systems Tester installation directory. This can be copied and modified to fit your own integration needs.

Important!

The actual example integration implementation can potentially change without prior notice in future versions. The PL API will not change without notice.

PL General Functions

t3pl_general_prepare_testcase

Prepares the integration for a new test case.

```
void t3pl_general_prepare_testcase
        (const char* module,
         const char* testcase,
         t3rt_type_t mtc_type,
         t3rt_type_t system_type,
         t3rt_context_t context);
```

Parameters

`module`	Name of module that defines test case.
`testcase`	Name of the preparing test case.
`mtc_type`	Type of the mtc component.
`system_type`	Type of the system component.

Description

This function is called before the execution of each test case to enable the integration to be prepared, if necessary.

t3pl_general_postprocess_testcase

Finalizes the integration for a terminating test case.

```
void t3pl_general_postprocess_testcase
        (const char* module,
         const char* testcase,
```

```
        t3rt_type_t mtc_type,
        t3rt_type_t system_type,
        t3rt_context_t context);
```

Parameters

module	Name of module that defines test case.
testcase	Name of the preparing test case.
mtc_type	Type of the mtc component.
system_type	Type of the system component.

Description

This function is called when the execution of each test case is going to terminate.

t3pl_general_testcase_terminated

Cleans up after a test case has finished.

```
void t3pl_general_testcase_terminated
        (const char* module,
         const char* testcase,
         t3rt_type_t mtc_type,
         t3rt_type_t system_type,
         t3rt_context_t context);
```

Parameters

module	Name of module that defines test case.
testcase	Name of the terminated test case.
mtc_type	Type of the mtc component.
system_type	Type of the system component.

Description

This function is called when the execution of a test case has been finished to enable the integration to take the actions it finds necessary.

t3pl_general_control_terminated

Cleans up after a control part has finished.

```
void t3pl_general_control_terminated
        (const char* module,
         t3rt_context_t context);
```

Parameters

module	Name of module that defines test case.

Description

This function is called when the execution of a control part has been finished to enable the integration to take actions it finds necessary. This function is also called after test case directly started with tciStartTestcase command terminates.

t3pl_call_external_function

Carries out the execution of an external function.

```
void t3pl_call_external_function
        (t3rt_type_t signature_type,
         t3rt_value_t parameters[],
         t3rt_value_t return_value,
         t3rt_context_t context);
```

Parameters

signature_type	The signature type for the external function.
parameters	Actual argument vector for the function call.
return_value	The (pre-allocated) inout value container for the return value. If this is set to the t3rt_no_value_c constant, no return value is expected (according to the signature type).

Description

This function is called as a result of an external function call in the test suite. This function may block.

PL Timer Functions

t3pl_time_pre_initialize

Performs timer module pre-initialization.

```
void t3pl_time_pre_initialize
        (int argc,
         char *argv[],
         t3rt_context_t context);
```

Parameters

argc	Number of elements in the argv character string array.
argv	String array of command line parameters.

Description

This function is called to perform pre-initialization of timer module. It's called before RTConf table is filled with user-provided values.

This function is called only once.

t3pl_time_initialize

Initializes/Resets timer module.

```
void t3pl_time_initialize (t3rt_context_t ctx);
```

Description

This function is called to initialize/reset timer module. It's called after RT-Conf and root module initialization.

When using TCI or GUI test management this function is called at initialization of every directly started test case and/or control part.

t3pl_time_finalize

Finalizes timer module.

```
void t3pl_time_finalize (t3rt_context_t ctx);
```

Description

This function is called to finalize timer module. No timer handling routines will be called after it.

t3pl_timer_create

Creates a timer instance into the “stopped” state.

```
void t3pl_timer_create
        (t3rt_timer_handle_t *handle,
         t3rt_context_t context);
```

Parameters

handle	Pointer to t3rt_timer_handle_t object that receives handle of created timer

Description

This function is called to create new timer instance. The initial state of created timer is “stopped“.

t3pl_timer_delete

Deletes the timer instance. It will never be used again.

```
void t3pl_timer_delete
        (const t3rt_timer_handle_t *handle,
         t3rt_context_t context);
```

Parameters

handle	Pointer to t3rt_timer_handle_t object that stores handle of timer to be deleted

Description

This function is called to delete timer instance. It's not assumed that timer should be stopped prior to calling this function.

t3pl_timer_start

Starts a created timer instance.

```
void t3pl_timer_start
        (const t3rt_timer_handle_t handle,
         double duration,
         t3rt_context_t context);
```

Parameters

handle	Timer handle
duration	Timeout period specified in seconds

Description

This function is used to start or restart the timer and may be applied to running timer. This should set the timer into the "running" state.

t3pl_timer_stop

Stops a timer instance.

```
void t3pl_timer_stop
        (const t3rt_timer_handle_t handle,
         t3rt_context_t context);
```

Parameters

handle	Timer handle

Description

This should set the timer into the "stopped" state.

t3pl_timer_read

Reads the current value of a timer.

```
t3rt_timer_state_t t3pl_timer_read
        (const t3rt_timer_handle_t handle,
         double *elapsed_time,
         t3rt_context_t context);
```

Parameters

handle	Timer handle
elapsed_timer	Number of seconds elapsed since timer start (‘out’ parameter)

Description

This function is used to query timer state. Second parameter may be NULL.

Return Value

Returns timer state. If elapsed timer parameter is not NULL then it’s set to number of seconds elapsed since timer start.

t3pl_timer_decode

Obtains TRI timer id from the timer handle.

```
t3rt_binary_string_t t3pl_timer_decode
        (t3rt_timer_handle_t handle,
         t3rt_alloc_strategy_t strategy,
         t3rt_context_t ctx);
```

Parameters

handle	Timer handle
strategy	Memory allocation strategy for the return value

Description

This function is used to convert runtime system timer handle into TRI timer id. Note that timer id is returned as t3rt_binary_string_t object, not as TriTimerId (i.e. TRI BinaryString).

Return Value

Returns TRI timer id as binary string of t3rt_binary_string_t type.

PL Communication Functions

t3pl_communication_pre_initialize

Performs communication module pre-initialization.

```
void t3pl_communication_pre_initialize
        (int argc,
         char *argv[],
         t3rt_context_t context);
```

Parameters

argc	Number of elements in the argv character string array.
argv	String array of command line parameters.

Description

This function is called to perform pre-initialization of communication module. It's called before RTConf table is filled with user-provided values.

This function is called only once.

t3pl_communication_initialize

Initializes/Resets communication module.

```
void t3pl_communication_initialize(t3rt_context_t context);
```

Description

This function is called to initialize/reset communication module. It's called after RTConf and root module initialization.

When using TCI or GUI test management this function is called at initialization of every directly started test case and/or control part.

t3pl_communication_finalize

Finalizes communication module.

```
void t3pl_communication_finalize (t3rt_context_t ctx);
```

Description

This function is called to finalize communication module. No communication handling routines will be called after it.

t3pl_port_create

Creates and initializes a port.

```
void t3pl_port_create
        (t3rt_value_t port_value,
         t3rt_binary_string_t address,
         const char* name, long index,
         t3rt_context_t ctx);
```

Parameters

port_value	Port value, may be used to extract port type.
address	Address of created port (‘out’ parameter).
name	Port or port array name.
index	Index in port array, -1 in case of scalar port.

Description

Creates and initializes new port. It includes any platform dependent communications mechanism, address, and queue initialization. This function supports only scalar ports and one-dimensional port arrays. Address of created port should be returned through ‘address’ out parameter.

t3pl_port_create_control_port_for_cpc

Creates control port for CPC (control) component

```
void t3pl_port_create_control_port_for_cpc
        (t3rt_binary_string_t address,
         t3rt_context_t ctx);
```

Parameters

address	Address of created port (‘out’ parameter).

Description

Creates and initializes CPC control port. It includes any platform dependent communications mechanism, address, and queue initialization. This is the first port created in the test suite.

t3pl_port_start

Starts a port.

```
void t3pl_port_start
        (t3rt_binary_string_t port_address,
         t3rt_context_t ctx);
```

Parameters

port_address	Address of port to be started.

Description

This is the direct mapping of the TTCN-3 port ‘start’ statement. After it port becomes active and should be able to transmit data through it.

t3pl_port_stop

Stops a port.

```
void t3pl_port_stop
        (t3rt_binary_string_t port_address,
         t3rt_context_t ctx);
```

Parameters

port_address	Address of port to be stopped.

Description

This is the direct mapping of the TTCN-3 port ‘stop’ statement. After it port becomes inactive and should not transmit any data through it.

t3pl_port_halt

Halts a port.

```
void t3pl_port_halt
        (t3rt_binary_string_t port_address,
         t3rt_context_t ctx);
```

Parameters

port_address	Address of port to be halted.

Description

This is the direct mapping of the TTCN-3 port ‘halt’ statement. After it port stops transmitting messages and receiving new messages. All data already in port queue is processed accordingly. After all message are extracted from port queue port becomes inactive.

t3pl_port_destroy

Destroys a port.

```
void t3pl_port_destroy
        (t3rt_binary_string_t port_address,
         t3rt_context_t ctx);
```

Parameters

port_address	Address of port to be destroyed.

Description

This function destroys port deallocating all port data structures. The port with given address will never be used more.

t3pl_port_clear

Discards any data contents of the port.

```
void t3pl_port_clear
        (t3rt_binary_string_t port_address,
         t3rt_context_t ctx);
```

Parameters

port_address	Address of port to be cleared.

Description

This is the direct mapping of the TTCN-3 port ‘clear’ statement. Any received data in port buffers that has not been passed to runtime system is deleted.

t3pl_port_component_send

Send data to another running test component.

```
void t3pl_port_component_send
        (t3rt_binary_string_t dest_component,
         t3rt_binary_string_t port_address,
         t3rt_binary_string_t data,
         t3rt_context_t ctx);
```

Parameters

dest_component	Destination component address.
port_address	Destination port address.
data	Encoded data.

Description

This function is called whenever one of the components (including control component) needs to communicate with another component. It may be called as a result of TTCN-3 ‘send’, ‘call’, ‘reply’, ‘raise’ operations as well as to perform service communication through control port. It’s used only in internal test suite communication, communication with SUT is done using other functions.

t3pl_port_sut_send

Send encoded data to the SUT.

```
void t3pl_port_sut_send
        (t3rt_binary_string_t port_address,
         t3rt_binary_string_t sut_address,
         t3rt_binary_string_t data,
         t3rt_binary_string_t sender_port_address,
         t3rt_context_t ctx);
```

Parameters

port_address	Address of destination previously mapped port.
sut_address	The encoded SUT address value.
data	Encoded data
sender_port_address	The port address of the sender.

Description

This function is called by the RTS when it executes a TTCN-3 unicast ‘send’ operation on a component port that has been mapped to a system port. port address parameter identifies port that has been previously mapped.

t3pl_port_sut_send_mc

Send encoded data to the multiple entities within SUT.

```
void t3pl_port_sut_send_mc
        (t3rt_binary_string_t port_address,
         t3rt_binary_string_t sut_address_list[],
         t3rt_binary_string_t data,
         t3rt_binary_string_t sender_port_address,
         t3rt_context_t ctx);
```

Parameters

port_address	Address of destination previously mapped port.
sut_address_list	NULL-terminated list of the encoded SUT address values.
data	Encoded data
sender_port_address	The port address of the sender.

Description

This function is called by the RTS when it executes a TTCN-3 multicast ‘send’ operation on a component port that has been mapped to a system port. port address parameter identifies port that has been previously mapped.

t3pl_port_sut_send_bc

Send encoded data to all entities within SUT.

```
void t3pl_port_sut_send_bc
        (t3rt_binary_string_t port_address,
         t3rt_binary_string_t data,
         t3rt_binary_string_t sender_port_address,
         t3rt_context_t ctx);
```

Parameters

port_address	Address of destination previously mapped port.
data	Encoded data
sender_port_address	The port address of the sender.

Description

This function is called by the RTS when it executes a TTCN-3 broadcast ‘send’ operation on a component port that has been mapped to a system port. port address parameter identifies port that has been previously mapped.

t3pl_port_sut_call

Request SUT to call specified remote function.

```
void* t3pl_port_sut_call
        (t3rt_binary_string_t port_to_call,
         t3rt_binary_string_t sut_address,
         t3rt_binary_string_t caller_port,
         t3rt_binary_string_t address,
         t3rt_type_t signature_type,
         t3rt_binary_string_t parameters[],
         t3rt_context_t ctx);
```

Parameters

port_to_call	Address of destination previously mapped port
sut_address	The encoded SUT address value
caller_port	Address of caller’s port

address	Reserved parameter, should be NULL
signature_type	The signature type, this specifies which function to be called
parameters	An array of encoded parameter values – not NULL terminated

Description

This function is called by the RTS when it executes a TTCN-3 unicast ‘call’ operation on a component port that has been mapped to a system port.

The parameters are in encoded form, that is, binary data, but output parameters are replaced with NULL values in the array.

This function is expected to not block, which means it should somehow request another thread or process to perform the actual call on behalf of the requesting component.

This function returns an opaque handle that is assumed to represent the requested call operation. This handle is later supplied when calling either the t3pl_port_sut_call_done or the t3pl_port_sut_call_abort function.

Return Values

This function returns an opaque handle that is assumed to represent the requested call operation. If such a handle is not needed, NULL may be returned instead.

t3pl_port_sut_call_mc

Request SUT to call specified remote function.

```
void* t3pl_port_sut_call_mc
        (t3rt_binary_string_t port_to_call,
         t3rt_binary_string_t sut_address_list[],
         t3rt_binary_string_t caller_port,
         t3rt_binary_string_t address,
         t3rt_type_t signature_type,
         t3rt_binary_string_t parameters[],
         t3rt_context_t ctx);
```

Parameters

port_to_call	Address of destination previously mapped port
sut_address_list	NULL-terminated list of the encoded SUT address values
caller_port	Address of caller's port
address	Reserved parameter, should be NULL
signature_type	The signature type, this specifies which function to be called
parameters	An array of encoded parameter values – not NULL terminated

Description

This function is called by the RTS when it executes a TTCN-3 multicast 'call' operation on a component port that has been mapped to a system port.

The parameters are in encoded form, that is, binary data, but output parameters are replaced with NULL values in the array.

This function is expected to not block, which means it should somehow request another thread or process to perform the actual call on behalf of the requesting component.

This function returns an opaque handle that is assumed to represent the requested call operation. This handle is later supplied when calling either the t3pl_port_sut_call_done or the t3pl_port_sut_call_abort function.

Return Values

This function returns an opaque handle that is assumed to represent the requested call operation. If such a handle is not needed, NULL may be returned instead.

t3pl_port_sut_call_bc

Request SUT to call specified remote function.

```
void* t3pl_port_sut_call_bc
        (t3rt_binary_string_t port_to_call,
         t3rt_binary_string_t caller_port,
```

```
t3rt_binary_string_t address,
t3rt_type_t signature_type,
t3rt_binary_string_t parameters[],
t3rt_context_t ctx);
```

Parameters

`port_to_call`	Address of destination previously mapped port
`caller_port`	Address of caller's port
`address`	Reserved parameter, should be NULL
`signature_type`	The signature type, this specifies which function to be called
`parameters`	An array of encoded parameter values – not NULL terminated

Description

This function is called by the RTS when it executes a TTCN-3 broadcast 'call' operation on a component port that has been mapped to a system port.

The parameters are in encoded form, that is, binary data, but output parameters are replaced with NULL values in the array.

This function is expected to not block, which means it should somehow request another thread or process to perform the actual call on behalf of the requesting component.

This function returns an opaque handle that is assumed to represent the requested call operation. This handle is later supplied when calling either the t3pl_port_sut_call_done or the t3pl_port_sut_call_abort function.

Return Values

This function returns an opaque handle that is assumed to represent the requested call operation. If such a handle is not needed, NULL may be returned instead.

t3pl_port_sut_call_done

Request SUT to release the handle to the finished call operation.

```
void t3pl_port_sut_call_done
```

```
        (void* handle,
         t3rt_context_t ctx);
```

Parameters

handle	the handle previously returned by the t3pl_port_sut_call function

Description

This function is expected to release the opaque handle that is assumed to represent a previously requested call operation.

t3pl_port_sut_call_abort

Request SUT to release the handle to the timed out call operation.

```
void t3pl_port_sut_call_abort
        (void* handle,
         t3rt_context_t ctx);
```

Parameters

handle	the handle previously returned by the t3pl_port_sut_call function

Description

This function is expected to release the opaque handle that is assumed to represent a previously requested call operation. Note the risk of encountering a race condition here, as the call operation was not “officially” terminated when this function is decided to be called, but may have managed to terminate before this function is called anyway.

t3pl_port_sut_reply

Return a reply value to a previously received call.

```
void t3pl_port_sut_reply
        (t3rt_binary_string_t destination_port,
         t3rt_binary_string_t sut_address,
         t3rt_binary_string_t caller_port,
         t3rt_binary_string_t address,
         t3rt_type_t signature_type,
```

```
        t3rt_binary_string_t parameters[],
        t3rt_binary_string_t return_value,
        t3rt_context_t ctx);
```

Parameters

destination_port	Address of destination previously mapped port
sut_address	Encoded SUT address value
caller_port	Address of caller's port
address	Reserved parameter, should be NULL
signature_type	The signature type, this specifies which function was called
parameters	An array of encoded parameter values – not NULL terminated
return_value	The encoded return value, if any

Description

This function is called by the RTS when it executes a TTCN-3 unicast 'reply' operation on a component port that has been mapped to a system port.

The parameters are in encoded form, that is, binary data, but input parameters are replaced with NULL values in the array.

t3pl_port_sut_reply_mc

Return a reply value to a previously received call.

```
void t3pl_port_sut_reply_mc
        (t3rt_binary_string_t destination_port,
         t3rt_binary_string_t sut_address_list[],
         t3rt_binary_string_t caller_port,
         t3rt_binary_string_t address,
         t3rt_type_t signature_type,
         t3rt_binary_string_t parameters[],
         t3rt_binary_string_t return_value,
         t3rt_context_t ctx);
```

Parameters

destination_port	Address of destination previously mapped port
sut_address_list	NULL-terminated list of the encoded SUT address values
caller_port	Address of caller's port
address	Reserved parameter, should be NULL
signature_type	The signature type, this specifies which function was called
parameters	An array of encoded parameter values – not NULL terminated
return_value	The encoded return value, if any

Description

This function is called by the RTS when it executes a TTCN-3 multicast 'reply' operation on a component port that has been mapped to a system port.

The parameters are in encoded form, that is, binary data, but input parameters are replaced with NULL values in the array.

t3pl_port_sut_reply_bc

Return a reply value to a previously received call.

```
void t3pl_port_sut_reply_bc
        (t3rt_binary_string_t destination_port,
         t3rt_binary_string_t caller_port,
         t3rt_binary_string_t address,
         t3rt_type_t signature_type,
         t3rt_binary_string_t parameters[],
         t3rt_binary_string_t return_value,
         t3rt_context_t ctx);
```

Parameters

destination_port	Address of destination previously mapped port
caller_port	Address of caller's port
address	Reserved parameter, should be NULL

signature_type	The signature type, this specifies which function was called
parameters	An array of encoded parameter values – not NULL terminated
return_value	The encoded return value, if any

Description

This function is called by the RTS when it executes a TTCN-3 broadcast ‘reply’ operation on a component port that has been mapped to a system port.

The parameters are in encoded form, that is, binary data, but input parameters are replaced with NULL values in the array.

t3pl_port_sut_raise

Return an exception value to a previously received call.

```
void t3pl_port_sut_raise
        (t3rt_binary_string_t destination_port,
         t3rt_binary_string_t sut_address,
         t3rt_binary_string_t caller_port,
         t3rt_binary_string_t address,
         t3rt_type_t signature_type,
         t3rt_binary_string_t exception_value,
         t3rt_context_t ctx);
```

Parameters

destination_port	Address of destination previously mapped port
sut_address	Encoded SUT address value
caller_port	Address of caller’s port
address	Reserved parameter, should be NULL
signature_type	The signature type, this specifies which function was called
exception_value	The encoded exception value

Description

This function is called by the RTS when it executes a TTCN-3 unicast ‘raise’ operation on a component port that has been mapped to a system port.

The exception value is provided in encoded form, that is, binary data.

t3pl_port_sut_raise_mc

Return an exception value to a previously received call.

```
void t3pl_port_sut_raise_mc
        (t3rt_binary_string_t destination_port,
         t3rt_binary_string_t sut_address_list[],
         t3rt_binary_string_t caller_port,
         t3rt_binary_string_t address,
         t3rt_type_t signature_type,
         t3rt_binary_string_t exception_value,
         t3rt_context_t ctx);
```

Parameters

destination_port	Address of destination previously mapped port
sut_address_list	NULL-terminated list of the encoded SUT address values
caller_port	Address of caller’s port
address	Reserved parameter, should be NULL
signature_type	The signature type, this specifies which function was called
exception_value	The encoded exception value

Description

This function is called by the RTS when it executes a TTCN-3 multicast ‘raise’ operation on a component port that has been mapped to a system port.

The exception value is provided in encoded form, that is, binary data.

t3pl_port_sut_raise_bc

Return an exception value to a previously received call.

```
void t3pl_port_sut_raise_bc
        (t3rt_binary_string_t destination_port,
         t3rt_binary_string_t caller_port,
         t3rt_binary_string_t address,
         t3rt_type_t signature_type,
         t3rt_binary_string_t exception_value,
         t3rt_context_t ctx);
```

Parameters

destination_port	Address of destination previously mapped port
caller_port	Address of caller's port
address	Reserved parameter, should be NULL
signature_type	The signature type, this specifies which function was called
exception_value	The encoded exception value

Description

This function is called by the RTS when it executes a TTCN-3 broadcast 'raise' operation on a component port that has been mapped to a system port.

The exception value is provided in encoded form, that is, binary data.

t3pl_port_sut_action

Performs the SUT action (implicit send in TTCN-2).

```
void t3pl_port_sut_action
        (t3rt_value_t string_or_template,
         t3rt_context_t ctx);
```

Parameters

string_or_template	Character string or template value

Description

This function is called by the RTS when it executes a TTCN-3 "SUT action" operation. Depending on what type of action is performed specified value may represent character string or template value.

t3pl_port_retrieve_system_port

Retrieves the system port address.

```
void t3pl_port_retrieve_system_port
        (const char * system_port_name,
         long index,
         t3rt_binary_string_t system_port_address,
         t3rt_context_t ctx);
```

Parameters

system_port_name	Name of system (TSI) port (or port array)
index	Index in port array (or -1 in case of scalar port)
system_port_address	Output parameter for system (TSI) port address

Description

This function is called when mapping and unmapping a local port to a named system port. It locates system (TSI) component port using given port name and port array index and returns port address through ‘system_port_address’ output parameter.

t3pl_port_release_system_port

Releases the system port address.

```
void t3pl_port_release_system_port
        (t3rt_binary_string_t system_port_address,
         t3rt_context_t ctx);
```

Parameters

system_port_address	TSI port address

Description

This function is called when unmapping a system port.

t3pl_port_map, t3pl_port_unmap

Maps and unmaps port

```
void t3pl_port_map
        (t3rt_binary_string_t port_address,
         const char* system_port_name,
         long index,
         t3rt_context_t ctx);

void t3pl_port_unmap
        (t3rt_binary_string_t port_address,
         t3rt_context_t ctx);
```

Parameters

port_address	Container for mapped/unmapped port address
system_port_name	Name of system port (or name of port array)
index	Index in port array (or -1 for scalar port)

Description

These functions are direct mappings of the TTCN-3 ‘map’ and ‘unmap’ operations. ‘port_address’ represents port that is going to be mapped or unmapped. Multidimensional port arrays are not supported.

t3pl_component_get_system_control_port

Returns the port for controlling the system (TSI) component.

```
t3rt_binary_string_t t3pl_component_get_system_control_port
(t3rt_context_t ctx);
```

Description

This function is called whenever there is a need to get address of system (TSI) component control port.

Return Value

Address of system (TSI) component control port.

t3pl_component_set_system_component_type

Sets the component type of the system component.

```
void t3pl_component_set_system_component_type
        (t3rt_type_t component_type,
         t3rt_context_t ctx);
```

Parameters

component_type	Type of system (TSI) component

Description

This function is called when starting test case. It's the right place to create and initialize system component. It should create control port and all communication ports. Use given component type to process all component fields and perform necessary initialization. After leaving this function system component should be ready for map and unmap operations.

t3pl_component_wait

Retrieve any input from the environment to put in ports or detect timeout.

```
t3rt_snapshot_return_t t3pl_component_wait
        (double* real_time_wait,
         double* time_to_soonest_timeout,
         t3rt_context_t ctx);
```

Parameters

real_time_wait	An inout timeout value stating the maximum time we want to wait before the waiting should be interrupted. It should be modified and set to the "time left" after data has arrived. This timeout value comes from real-time related time-outs in difference to the declared timer.
time_to_soonest_timeout	An inout timeout value stating the maximum time the integration should to wait before the TTCN-3 timer will time out. It should be modified and set to the "time left" after data has arrived.

Description

Blocks the current component waiting for some external stimuli (that is, some message received or timer timing out). This function will return information if there was data received and/or a timeout occurred. In either case the timeout values, real_time_wait and time_to_soonest_timeout must be updated according to how long it really took.

The actual wait should not be longer than the least of the timeout values. The reason for having two timeout values is that if the time scale for TTCN-3 timers is not equal to the real-time clock (for example, time is slowed down or sped up), the integration is the only place where this is known and this function must make adjustments to the time waited.

`t3rt_duration_forever_c` is a valid timeout value for both parameters and if both parameters have this value the function should wait indefinitely.

If one (or both) of the timeout values is set to `t3rt_duration_nowait_c`, the function should have polling semantics, just checking for existing data/time-outs, not waiting.

Return Values

Returns information if data was received, and/or a timeout occurred.

t3pl_component_control

Processes and dispatches control messages.

```
t3pl_component_control(t3rt_context_t ctx);
```

Description

This function is called whenever it's necessary to process control messages in the incoming event queue without touching data messages. It differs from the t3pl_component_wait in two ways: it processes only control messages (i.e. messages received through control port) and it doesn't block if there are no messages.

PL Memory Functions

t3pl_memory_pre_initialize

Performs memory module pre-initialization.

```
void t3pl_memory_pre_initialize(int argc, char *argv[])
```

Parameters

`argc`	Number of elements in the argv character string array.
`argv`	String array of command line parameters.

Description

This function is called to perform pre-initialization of memory module. It's called before RTConf table initialization.

This function is called only once.

After a call to t3pl_memory_pre_initialize, the t3pl_memory_allocate function must be working.

t3pl_memory_initialize

Initializes/Resets memory module

```
void t3pl_memory_initialize (t3rt_context_t context);
```

Description

This function is called to initialize/reset memory module. It's called after RTConf and root module initialization.

When using TCI or GUI test management this function is called at initialization of every directly started test case and/or control part.

After a call to t3pl_memory_initialize, all memory primitives must be available.

t3pl_memory_finalize

Finalizes memory module

```
void t3pl_memory_finalize (t3rt_context_t context);
```

Description

This function is called to finalize memory module. No memory handling routines will be called after it.

t3pl_memory_allocate

Allocated memory block

```
void* t3pl_memory_allocate
        (const t3rt_alloc_strategy_t strategy,
         const unsigned long size,
         t3rt_context_t context);
```

Parameters

strategy	Memory allocation strategy.
size	Size of allocated memory in bytes.

Description

This function allocates ‘size’ number of bytes of memory.

Return value

Returns pointer to allocated memory or NULL if memory cannot be allocated.

t3pl_memory_deallocate

Deallocates given memory block

```
void t3pl_memory_deallocate
        (const t3rt_alloc_strategy_t strategy,
         void* mem,
         t3rt_context_t context);
```

Parameters

`strategy`	Memory allocation strategy of the given memory block.
`mem`	Pointer to the memory block to be deallocated.

Description

This function deallocates given memory block. It's not assumed that integration stores memory allocation strategy for each allocated block thus 'strategy' parameter is used to tell integration which strategy has been used to allocate given memory block.

t3pl_memory_reallocate

Reallocates given memory block

```
void* t3pl_memory_reallocate
        (const t3rt_alloc_strategy_t strategy,
         void* mem, const unsigned long new_size,
         t3rt_context_t context);
```

Parameters

`strategy`	Memory allocation strategy of the given memory block.
`mem`	Pointer to the memory block to be reallocated.
`new_size`	Size in bytes for the new memory block

Description

This function is called to resize existing memory block. The contents of the result are unchanged up to the shorter of new and old sizes. New block may be in a different location, i.e. it's not guaranteed that pointer returned by t3pl_memory_reallocate is the same as passed through 'mem' parameter.

PL Concurrency Functions

t3pl_concurrency_pre_initialize

Pre-initializes the concurrency module.

```
void t3pl_concurrency_pre_initialize
        (int argc,
         char *argv[],
         t3rt_context_t ctx);
```

Parameters

argc	Number of elements in the argv character string array.
argv	String array of command line parameters.

Description

This function is called to perform pre-initialization of concurrency module. It's called before RTConf table is filled with user-provided values thus it cannot rely on RTS configuration information.

This function is called only once.

t3pl_concurrency_initialize

Initializes/Resets the concurrency functionality.

```
void t3pl_concurrency_initialize (t3rt_context_t ctx);
```

Description

These should set the concurrency implementation of the integration into a state where it is fully functional.

This function can rely on the contents of the RTS configuration information.

When using TCI or GUI test management this function is called at initialization of every directly started test case and/or control part.

t3pl_concurrency_finalize

Finalizes concurrency module.

```
void t3pl_concurrency_finalize (t3rt_context_t ctx);
```

Description

This function is called to finalize all concurrency handling functionality. No concurrency handling routines will be called after it.

t3pl_concurrency_start_separate_component

Called when a component has been started in a separate process.

```
void t3pl_concurrency_start_separate_component
        (int argc,
         const char* argv[],
         t3rt_context_t ctx);
```

Parameters

argc	Number of arguments in the argument vector ‘argv’.
argv	The argument vector passed from the command line when the process was created

Description

Start the first (non-CPC) component of the current process. This should communicate with the creator of this component and hand over the newly created control port address of this component.

This function is supposed to end by calling the t3rt_component_main function with the newly created control port address of this component along with the provided context.

t3pl_task_create

Create a port to control the task and a thread of execution executing the function t3rt_component_main.

```
void t3pl_task_create
        (int argc,
```

```
        const char * argv[],
        t3rt_value_t component_value,
        t3rt_binary_string_t address,
        t3rt_context_t context);
```

Parameters

argc	Number of arguments in the argument vector ‘argv’.
argv	The argument vector passed from the command line when the process was created
component_value	Inout value for the component.
address	Inout parameter to be set to the address of the created tasks control port.

Description

This is a direct mapping from the TTCN-3 create operation. This function creates the thread of execution, the control port of this component and initiates the component value with the control port address. After this operation, the created component is fully initialized and in a state where it is listening to its control port.

t3pl_task_setup

Initializes new component

```
void t3pl_task_setup
        (t3rt_binary_string_t compaddr,
         t3rt_context_t context);
```

Parameters

compaddr	Task control port address.

Description

This function is called from the t3rt_component_main function in an attempt to setup whatever is necessary for the component to communicate through its ports. It is created after the control port has been created but before any other port is created and before any communication between components is made.

t3pl_task_id

Lookups task identifier

```
unsigned long t3pl_task_id();
```

Description

This function is used to lookup system dependent task identifier (e.g. process or thread id or whatever). Note that this function does not receive reference to context. Task identifier may be reused by other component after it's termination, i.e. two components may have same id if they do not execute concurrently.

Return Value

Returns integer value that uniquely identifies task in the test suite. This function should return one and the same value each time component calls it.

t3pl_task_register_context

Registers context of the new task

```
void t3pl_task_register_context (t3rt_context_t context);
```

Description

This provides integration with the task context. It may be used to call RTS routines that require context reference from inside the TRI functions that doesn't know about RTS context (e.g. triExternalFunction).

t3pl_task_kill

Force the task to stop executing.

```
void t3pl_task_kill
        (t3rt_binary_string_t address,
         const bool shutdown_acknowledged,
         t3rt_context_t context);
```

Parameters

address	Address to the control port of the component executing in this task that should be killed.
shutdown_acknowledged	Set to “true” if the component shut down according to the normal shutdown procedure.

Description

This is called by a component in a thread that wants to kill a task running in another thread. This function is called even if the task did shutdown according to the normal shutdown procedure. This is done just to enable the implementation to make necessary clean up.

t3pl_task_exit

Called to terminate this task.

```
void t3pl_task_exit
        (t3rt_context_cleanup_function_t context_cleanup_f,
         t3rt_context_t context);
```

Parameters

context_cleanup_f	Address to the function that performs context finalization.

Description

This is called by a component thread to exit normally with finalizing all task objects. ‘context_cleanup_f’ function should be called right before terminating task.

t3pl_sem_create

Creates new semaphore object

```
void* t3pl_sem_create
        (unsigned int value,
         t3rt_context_t ctx);
```

Parameters

`value`	Amount of available resources guarded by semaphore.

Description

This function creates new semaphore that guards ‘value’ instances of some object. It means that ‘value’ threads may simultaneously acquire (lock) semaphore. ‘value’ is the initial value for semaphore counter. It may be equal to zero.

Return Value

Returns handle to the created semaphore or NULL if it cannot be created.

t3pl_sem_wait

Performs unlimited time waiting on semaphore

```
bool t3pl_sem_wait
        (void *sem,
         t3rt_context_t ctx);
```

Parameters

`sem`	Handle to the semaphore.

Description

This function acquires (locks) semaphore. Each time this function is called semaphore counter (that initially equals to ‘value’ parameter of t3pl_sem_create function) is decremented. If the value of semaphore counter equals to zero (before decremented) then the thread is put into sleep state until one of other threads release semaphore by calling t3pl_sem_post.

Return Value

Returns true if semaphore has been acquired, false in case of error.

t3pl_sem_trywait

Tries to acquire (lock) semaphore without waiting

```
bool t3pl_sem_trywait
```

```
        (void *sem,
         t3rt_context_t ctx);
```

Parameters

sem	Handle to the semaphore.

Description

This function tries to acquire (lock) semaphore. If semaphore counter is greater than zero then behavior of this function is the same as t3pl_sem_wait. If semaphore counter equals to zero then function returns immediately without putting thread into sleep state. Semaphore is not acquired in latter case.

Return Value

Returns true if semaphore has been acquired, false otherwise.

t3pl_sem_timedwait

Performs limited time waiting on semaphore.

```
bool t3pl_sem_trywait
        (void *sem,
         double wait_seconds,
         t3rt_context_t ctx);
```

Parameters

sem	Handle to the semaphore.
wait_seconds	Wait limit

Description

This function tries to acquire (lock) semaphore. If semaphore counter is greater than zero then behavior of this function is the same as t3pl_sem_wait. If semaphore counter equals to zero then function waits specified amount of time for the semaphore to be released. If semaphore cannot be acquired during the specified time then function aborts returning false.

Return Value

Returns true if semaphore has been acquired, false otherwise.

t3pl_sem_post

Releases semaphore

```
bool t3pl_sem_post
        (void *sem,
         t3rt_context_t ctx);
```

Parameters

sem	Handle to the semaphore.

Description

This function releases (unlocks) semaphore. Each time this function is called semaphore counter (that initially equals to ‘value’ parameter of t3pl_sem_create function) increments. If there were threads waiting for semaphore in sleep state then one of them is awaken (thus decreasing semaphore counter).

Return Value

Returns true if semaphore has been released, false in case of error.

t3pl_sem_destroy

Destroys semaphore object

```
bool t3pl_sem_destroy
        (void *sem,
         t3rt_context_t ctx);
```

Parameters

sem	Handle to the semaphore.

Description

This function destroys given semaphore. All waiting threads (if any) are released.

Return Value

Returns true if semaphore has been successfully destroyed, false in case of error.

User Defined Functions

t3ud_register_codecs

Function called in the initiation phase to enable registering of codecs systems.

```
void t3ud_register_codecs
        (int argc,
         char * argv[],
         t3rt_context_t ctx);
```

Parameters

argc	Number of arguments in the argument vector ‘argv’.
argv	The argument vector passed from the command line when the process was created

Description

This function should call the t3rt_codecs_register function to register a codecs system. More than one codecs system can be registered.

t3ud_register_log_mechanisms

Register a log mechanism.

```
void t3ud_register_log_mechanisms(int argc, char * argv[]);
```

Parameters

argc	Number of arguments in the argument vector ‘argv’.
argv	The argument vector passed from the command line when the process was created

Description

This function should registers all user-defined log mechanisms by, for each such mechanism, calling the t3rt_log_register_listener function.

t3ud_read_module_param

Function to read a given test suite parameter for a module.

```
bool t3ud_read_module_param
        (const char * module_name,
         const char * param_name,
         t3rt_value_t value,
         t3rt_context_t ctx);
```

Parameters

module_name	The name of the module.
param_name	Name of the module parameter
value	Inout value container for the value to be read. This is an instantiated value of the correct (expected) type and the read value should be set (or assigned) to it.

Description

The read value should be stored in the provided value parameter by using the provided inout value container. If the type of the value is needed (or any value or type information), it can be accessed using the normal value and type access functions.

If this function defines the requested module parameter no attempts to set the parameters default value will be made.

The intended way to set module parameters is by using the command-line switches -par and -parfile. This function is only necessary to implement when a module parameter must be retrieved from a source where the command-line way is not sufficient.

Return Values

Returns true if the module parameter value was defined (set) by this function, false otherwise.

t3ud_retreive_configuration

Retrieves any environment information and stores this in the RTS configuration.

```
void t3ud_retrieve_configuration(t3rt_context_t ctx);
```

Description

This is a place to set up any configuration information in the environment into the RTS configuration storage using the function "t3rt_rtconf_set_param" on page 496.

t3ud_make_timestamp

Function to build user-defined timestamp for event logging.

```
void t3ud_make_timestamp
        (t3rt_binary_string_t timestamp,
         t3rt_context_t ctx);
```

Parameters

timestamp	Inout binary string container for the ASCII timestamp. This is an allocated binary string which should be filled with the valid ASCII timestamp.

Description

This function should prepare ASCII-based timestamp exactly in the same way as it should appear in execution log. Run-time system doesn't perform any transformations of the prepared timestamp and prints it as is. Binary string is used as the container for the arbitrary length character string only.

Example 25

```
void t3ud_make_timestamp(t3rt_binary_string_t timestamp,
t3rt_context_t ctx)
{
   struct timeb timebuffer;
   char *timeline;
   ftime( &timebuffer );
   timeline = ctime( & ( timebuffer.time ) );
   t3rt_binary_string_append_nbytes(timestamp, timeline,
   strlen(timeline)+1, ctx);
}
```

Return Values

None.

TRI API

This interface is defined according to TRI (ETSI ES 201 873-5 V3.2.1). See this document for further details.

The TRI interface functions are divided into four parts as defined in ETSI ES 201 873-5 V3.2.1, depending how and where they are used. They can be implemented by Rational Systems Tester (TE), the System Adaptor (SA) or the Platform Adaptor (PA) and used from (that is, called by) the same parts. So, the categories are:

- SA->TE
- PA->TE
- TE->SA
- TE->PA.

TRI Type Definitions

BinaryString

This is used for storing binary data, when handling encoded values, for example.

The data field is an array of bytes, not a null-terminated (ASCII) string. `bits` is the number of bits stored in the array and the `aux` field is for future extensibility of TRI functionality.

```
struct
{
  unsigned char*    data;
  long int          bits;
  void*             aux;
};
```

QualifiedName

A value of this type is used for any named object declared in the context of a component, a type or a timer, and so on.

The moduleName and objectName fields are the TTCN-3 identifiers literally and the `aux` field is for future extensibility of TRI functionality.

```
struct
```

```
{
  char* moduleName;
  char* objectName;
  void* aux;
};
```

TriActionTemplate

An alias type for BinaryString representing an action template.

TriAddress

An alias type for BinaryString representing an address.

TriAddressList

The representation of a list of TriAddress. This type is used for multicast communication in TRI.

No special values mark the end of addrList[]. The length field shall be used to traverse this array properly.

```
typedef struct _TriAddressList
{
  TriAddress    **addrList;
  long int      length;
};
```

TriException

An alias type for BinaryString representing an exception.

TriFunctionId

An alias type for QualifiedName representing a function identifier.

TriMessage

An alias type for BinaryString representing an encoded value.

TriSignatureId

An alias type for QualifiedName representing a signature identifier.

TriTestCaseId

An alias type for QualifiedName representing a test case identifier.

TriTimerDuration

A double value representing a time duration.

TriTimerId

An alias type for BinaryString representing a unique timer identifier.

Note

Pending ETSI statement on timer and snapshot semantics may influence future representation.

TriStatus

This is the status returned by all TRI functions that says whether the function succeeded or failed. The value of the type is either TRI_Error or TRI_OK.

Note

This is an unsigned integer type and all negative values are reserved for future extension of TRI functionality.

TriComponentId

The representation of a component instance.

The compInst field is a unique “handle” for the component instance, compName is the name of the component as provided in the “start” component operation and compType is the name of the component type.

```
typedef struct _TriComponentId
{
  BinaryString  compInst;
  char*         compName;
  QualifiedName compType;
};
```

TriComponentIdList

The representation of a list of TriComponentId. This type is used for multicast communication in TRI.

No special values mark the end of compIdList[]. The length field shall be used to traverse this array properly.

```
typedef struct _TriComponentIdList
{
  TriComponentId **compIdList;
  long int       length;
};
```

TriParameterPassingMode

```
typedef enum
{
  TRI_IN    = 0,
  TRI_INOUT = 1,
  TRI_OUT   = 2
};
```

TriParameter

The representation of an encoded parameter to functions.

```
typedef struct _TriParameter
{
  BinaryString            par;
  TriParameterPassingMode mode;
};
```

TriParameterList

No special values mark the end of parList[]. The length field shall be used to traverse this array properly.

```
typedef struct _TriParameterList
{
  TriParameter **parList;
  long int     length;
};
```

TriPortId

compInst is for component instance. For a singular (non-array) declaration, the `portIndex` value should be -1. The `aux` field is for future extensibility of TRI functionality.

```
typedef struct _TriPortId
{
  TriComponentId compInst;
  char*          portName;
  long int       portIndex;
  QualifiedName  portType;
  void*          aux;
};
```

TriPortIdList

No special values mark the end of portIdList[]. The length field shall be used to traverse this array properly.

```
typedef struct _TriPortIdList
{
  TriPortId **portIdList;
  long int  length;
};
```

SA->TE Functions

These functions are provided by the TRI integration to be called from the SA.

triEnqueueMsg

Enqueues a message in the input queue of the given port.

```
void triEnqueueMsg
        (const TriPortId *tsiPortId,
         const TriAddress *sutAddress,
         const TriComponentId *componentId,
         const TriMessage *receivedMessage);
```

Parameters

`tsiPortId`	identifier of the test system interface port via which the message is enqueued by the SUT Adapter
`sutAddress`	(optional) source address within the SUT
`componentId`	identifier of the receiving test component
`receivedMessage`	the encoded received message

Description

This operation is called by the SA after it has received a message from the SUT. It can only be used when tsiPortId has been either previously mapped to a port of componentId or has been referenced in the previous triExecuteTestCase statement.

In the invocation of a triEnqueueMessage operation receivedMessage shall contain an encoded value.

This operation shall pass the message to the TE indicating the port of the component componentId to which the TSI port tsiPortId is mapped.

The decoding of receivedMessage has to be done in the TE.

triEnqueueCall

Enqueues a call request in the input queue of the given procedure port.

```
void triEnqueueCall
        (const TriPortId* tsiPortId,
         const TriAddress* sutAddress,
         const TriComponentId* componentId,
         const TriSignatureId* signatureId,
         const TriParameterList*  parameterList);
```

Parameters

tsiPortId	identifier of the test system interface port via which the message is enqueued by the SUT Adapter
sutAddress	(optional) source address within the SUT
componentId	identifier of the receiving test component
signatureId	identifier of the signature of the procedure call
parameterList	A list of encoded parameters which are part of the indicated signature. The parameters in parameterList are ordered as they appear in the TTCN-3 signature declaration.

Description

This operation can be called by the SA after it has received a procedure call from the SUT. It can only be used when tsiPortId has been either previously mapped to a port of componentId or referenced in the previous triExecute statement.

In the invocation of a triEnqueueCall operation all in and inout procedure parameters contain encoded values. All out procedure parameters shall contain the distinct value of null since they are only relevant in the reply on the procedure call but not in the procedure call itself.

The TE can enqueue this procedure call with the signature identifier signatureId at the port of the component componentId to which the TSI port tsiPortId is mapped. The decoding of procedure parameters has to be done in the TE.

No error shall be indicated by the TE in case the value of any out parameter is non-null.

triEnqueueReply

Enqueues a reply to a call in the input queue of the given procedure port.

```
void TriEnqueueReply
        (const TriPortId* tsiPortId,
         const TriAddress* sutAddress,
         const TriComponentId* componentId,
         const TriSignatureId* signatureId,
         const TriParameterList* parameterList,
```

```
const TriParameter* returnValue);
```

Parameters

`tsiPortId`	identifier of the test system interface port via which the message is enqueued by the SUT Adapter
`sutAddress`	(optional) source address within the SUT
`componentId`	identifier of the receiving test component
`signatureId`	identifier of the signature of the procedure call
`parameterList`	A list of encoded parameters which are part of the indicated signature. The parameters in parameterList are ordered as they appear in the TTCN-3 signature declaration.
`returnValue`	(optional) encoded return value of the procedure call

Description

This operation can be called by the SA after it has received a reply from the SUT. It can only be used when tsiPortId has been either previously mapped to a port of componentId or referenced in the previous triExecute statement.

In the invocation of a triEnqueueReply operation all out and inout procedure parameters and the return value contain encoded values. All in procedure parameters shall contain the distinct value of null since they are only of relevance to the procedure call but not in the reply to the call.

If no return type has been defined for the procedure signature in the TTCN-3 ATS, the distinct value null shall be used for the return value.

The TE can enqueue this reply to the procedure call with the signature identifier signatureId at the port of the component componentId to which the TSI port tsiPortId is mapped. The decoding of the procedure parameters has to be done within the TE.

No error shall be indicated by the TE in case the value of any in parameter or a non-defined return value is non-null.

triEnqueueException

Enqueues an exception (raised during a call operation) in the input queue of the given procedure port.

```
void TriEnqueueException
        (const TriPortId* tsiPortId,
         const TriAddress* sutAddress,
         const TriComponentId* componentId,
         const TriSignatureId* signatureId,
         const TriException* exception);
```

Parameters

tsiPortId	identifier of the test system interface port via which the message is enqueued by the SUT Adapter
sutAddress	(optional) source address within the SUT
componentId	identifier of the receiving test component
signatureId	identifier of the signature of the procedure call
exception	the encoded exception

Description

This operation can be called by the SA after it has received a reply from the SUT. It can only be used when tsiPortId has been either previously mapped to a port of componentId or referenced in the previous triExecuteTestCase statement.

In the invocation of a triEnqueueException operation exception shall contain an encoded value.

The TE can enqueue this exception for the procedure call with the signature identifier signatureId at the port of the component componentId to which the TSI port tsiPortId is mapped.

The decoding of the exception has to be done within the TE.

PA->TE Functions

These functions are provided by TRI integration to be called from the PA.

triTimeout

This operation is called by the PA when a timer has expired.

```
void triTimeout(const TriTimerId *timerId);
```

Parameters

`timerId`	Identifier of the timer instance.

Description

This operation is called by the PA after a timer, which has previously been started using the triStartTimer operation, has expired, that is, it has reached its maximum duration value.

The timeout with the timerId can be added to the timeout list in the TE. The implementation of this operation in the TE has to be done in such a manner that it addresses the different TTCN-3 semantics for timers defined in TTCN-3.

TE->SA Functions

These functions are implemented in the SA part of the TRI implementation and will be called from the TE.

triSAReset

This operation can be called by the TE at any time to reset the SA.

```
TriStatus triSAReset(void);
```

Description

The SA shall reset all communication means which it is maintaining, that is reset static connections to the SUT, close dynamic connections to the SUT, discard any pending messages or procedure calls, for example.

The triSAReset operation returns TRI_OK in case the operation has been successfully performed, TRI_Error otherwise.

Return Values

The return status of the triSAReset operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.

triExecuteTestcase

Called to prepare the TRI implementation that a test case is about to be executed.

```
TriStatus triExecuteTestcase
        (const TriTestCaseId *testCaseId,
         const TriPortIdList *tsiPortList);
```

Parameters

`testCaseId`	Identifier of the test case that is going to be executed.
`tsiPortList`	A list of test system interface ports defined for the test system.

Description

This operation is called by the TE immediately before the execution of any test case. The test case that is going to be executed is indicated by the testCaseId. `tsiPortList` contains all ports that have been declared in the definition of the system component for the test case, that is, the TSI ports. If a system component has not been explicitly defined for the test case in the TTCN-3 ATS then the tsiPortList contains all communication ports of the MTC test component. The ports in tsiPortList are ordered as they appear in the respective TTCN-3 component declaration.

The SA can set up any static connections to the SUT and initialize any communication means for TSI ports.

The triExecuteTestcase operation returns TRI_OK in case the operation has been successfully performed, TRI_Error otherwise.

Return Values

The return status of the triExecuteTestcase operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.

triEndTestcase

Called immediately after the execution of any test case.

```
TriStatus triEndTestcase(void);
```

Description

This operation is called by the TE immediately after the execution of any test case.

The SA can free resources, cease communication at system ports and to test components.

The triEndTestCase operation returns TRI_OK in case the operation has been successfully performed, TRI_Error otherwise.

Return Values

The return status of the triEndTestcase operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.

triMap

Called when a port needs to be mapped.

```
TriStatus triMap
        (const TriPortId *compPortId,
         const TriPortId *tsiPortId);
```

Parameters

`compPortId`	Identifier of the test component port to be mapped.
`tsiPortId`	Identifier of the test system interface port to be mapped.

Description

This operation is called by the TE when it executes a TTCN-3 map operation.

The SA can establish a dynamic connection to the SUT for the referenced TSI port.

The triMap operation returns TRI_Error in case a connection could not be established successfully, TRI_OK otherwise. The operation should return TRI_OK in case no dynamic connection needs to be established by the test system.

Return Values

The return status of the triMap operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.

triUnmap

This operation is called by the TE when it executes any TTCN-3 `unmap` operation.

```
TriStatus triUnmap
        (const TriPortId *compPortId,
         const TriPortId *tsiPortId);
```

Parameters

`compPortId`	Identifier of the test component port to be unmapped.
`tsiPortId`	Identifier of the test system interface port to be unmapped.

Description

The SA shall close a dynamic connection to the SUT for the referenced TSI port.

The triUnmap operation returns TRI_Error in case a connection could not be closed successfully or no such connection has been established previously, TRI_OK otherwise. The operation should return TRI_OK in case no dynamic connections have to be established by the test system.

Return Values

The return status of the triUnmap operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.

triSend

Called when a message needs to be sent on a port to the single recipient.

```
TriStatus triSend
        (const TriComponentId *componentId,
         const TriPortId *tsiPortId,
         const TriAddress *sutAddress,
         const TriMessage *sendMessage);
```

Parameters

componentId	Identifier of the sending test component.
tsiPortId	Identifier of the test system interface port via which the message is sent to the SUT Adapter.
sutAddress	(Optional) destination address within the SUT.
sendMessage	The encoded message to be send.

Description

This operation is called by the TE when it executes a TTCN-3 unicast send operation on a component port, which has been mapped to a TSI port. This operation is called by the TE for all TTCN-3 send operations if no system component has been specified for a test case, that is, only an MTC test component is created for a test case.

The encoding of sendMessage has to be done in the TE prior to this TRI operation call.

The SA can send the message to the SUT.

The triSend operation returns TRI_OK in case it has been completed successfully. Otherwise TRI_Error shall be returned. Notice that the return value TRI_OK does not imply that the SUT has received sendMessage.

Return Values

The return status of the triSend operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.

triSendMC

Called when a message needs to be sent on a port to the multiple recipients.

```
TriStatus triSendMC
        (const TriComponentId *componentId,
         const TriPortId *tsiPortId,
         const TriAddressList *sutAddresses,
         const TriMessage *sendMessage);
```

Parameters

componentId	Identifier of the sending test component.
tsiPortId	Identifier of the test system interface port via which the message is sent to the SUT Adapter.
sutAddresses	(Optional) destination addresses within the SUT.
sendMessage	The encoded message to be send.

Description

This operation is called by the TE when it executes a TTCN-3 multicast send operation on a component port, which has been mapped to a TSI port. This operation is called by the TE for all TTCN-3 send operations if no system component has been specified for a test case, that is, only an MTC test component is created for a test case.

The encoding of sendMessage has to be done in the TE prior to this TRI operation call.

The SA can send the message to the SUT.

The triSendMC operation returns TRI_OK in case it has been completed successfully. Otherwise TRI_Error shall be returned. Notice that the return value TRI_OK does not imply that the SUT has received sendMessage.

Return Values

The return status of the triSendMC operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.

triSendBC

Called when a message needs to be sent on a port to all recipients in a SUT.

```
TriStatus triSendBC
        (const TriComponentId *componentId,
         const TriPortId *tsiPortId,
         const TriMessage *sendMessage);
```

Parameters

componentId	Identifier of the sending test component.
tsiPortId	Identifier of the test system interface port via which the message is sent to the SUT Adapter.
sendMessage	The encoded message to be send.

Description

This operation is called by the TE when it executes a TTCN-3 broadcast send operation on a component port, which has been mapped to a TSI port. This operation is called by the TE for all TTCN-3 send operations if no system component has been specified for a test case, that is, only an MTC test component is created for a test case.

The encoding of sendMessage has to be done in the TE prior to this TRI operation call.

The SA can send the message to the SUT.

The triSendBC operation returns TRI_OK in case it has been completed successfully. Otherwise TRI_Error shall be returned. Notice that the return value TRI_OK does not imply that the SUT has received sendMessage.

Return Values

The return status of the triSendBC operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.

triCall

Called when a procedure call needs to be made on a port to the single recipient.

```
TriStatus triCall
        (const TriComponentId* componentId,
         const TriPortId* tsiPortId,
         const TriAddress* sutAddress,
         const TriSignatureId* signatureId,
         const TriParameterList* parameterList);
```

Parameters

componentId	identifier of the test component issuing the procedure call
tsiPortId	identifier of the test system interface port via which the procedure call is sent to the SUT Adapter
sutAddress	(Optional) destination address within the SUT.
signatureId	identifier of the signature of the procedure call
parameterList	A list of encoded parameters which are part of the indicated signature. The parameters in parameterList are ordered as they appear in the TTCN-3 signature declaration.

Description

This operation is called by the TE when it executes a TTCN-3 unicast call operation on a component port, which has been mapped to a TSI port. This operation is called by the TE for all TTCN-3 call operations if no system component has been specified for a test case, that is, only an MTC test component is created for a test case.

All in and inout procedure parameters contain encoded values. All out procedure parameters shall contain the distinct value of null since they are only of relevance in a reply to the procedure call but not in the procedure call itself.

The procedure parameters are the parameters specified in the TTCN-3 signature template. Their encoding has to be done in the TE prior to this TRI operation call.

On invocation of this operation, the SA can initiate the procedure call corresponding to the signature identifier signatureId and the TSI port tsiPortId.

The triCall operation shall return without waiting for the return of the issued procedure call. This might be achieved for example by spawning a new thread or process. This handling of this procedure call is, however, dependent on implementation of the TE.

This TRI operation returns TRI_OK on successful initiation of the procedure call, TRI_Error otherwise. No error shall be indicated by the SA in case the value of any out parameter is non-null. Notice that the return value of this TRI operation does not make any statement about the success or failure of the procedure call.

Note

An optional timeout value, which can be specified in the TTCN-3 ATS for a call operation, is not included in the triCall operation signature. The TE is responsible for addressing this issue by starting a timer for the TTCN-3 call operation in the PA with a separate TRI operation call, that is, triStart-Timer.

Return Values

The return status of the triCall operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.

triCallMC

Called when a procedure call needs to be made on a port to the multiple recipients.

```
TriStatus triCall
        (const TriComponentId* componentId,
         const TriPortId* tsiPortId,
         const TriAddressList* sutAddresses,
         const TriSignatureId* signatureId,
         const TriParameterList* parameterList);
```

Parameters

componentId	identifier of the test component issuing the procedure call
tsiPortId	identifier of the test system interface port via which the procedure call is sent to the SUT Adapter

sutAddresses	(Optional) destination addresses within the SUT.
signatureId	identifier of the signature of the procedure call
parameterList	A list of encoded parameters which are part of the indicated signature. The parameters in parameterList are ordered as they appear in the TTCN-3 signature declaration.

Description

This operation is called by the TE when it executes a TTCN-3 multicast call operation on a component port, which has been mapped to a TSI port. This operation is called by the TE for all TTCN-3 call operations if no system component has been specified for a test case, that is, only an MTC test component is created for a test case.

All in and inout procedure parameters contain encoded values. All out procedure parameters shall contain the distinct value of null since they are only of relevance in a reply to the procedure call but not in the procedure call itself.

The procedure parameters are the parameters specified in the TTCN-3 signature template. Their encoding has to be done in the TE prior to this TRI operation call.

On invocation of this operation, the SA can initiate the procedure call corresponding to the signature identifier signatureId and the TSI port tsiPortId.

The triCallMC operation shall return without waiting for the return of the issued procedure call. This might be achieved for example by spawning a new thread or process. This handling of this procedure call is, however, dependent on implementation of the TE.

This TRI operation returns TRI_OK on successful initiation of the procedure call, TRI_Error otherwise. No error shall be indicated by the SA in case the value of any out parameter is non-null. Notice that the return value of this TRI operation does not make any statement about the success or failure of the procedure call.

Note

An optional timeout value, which can be specified in the TTCN-3 ATS for a call operation, is not included in the triCallMC operation signature. The TE is responsible for addressing this issue by starting a timer for the TTCN-3 call operation in the PA with a separate TRI operation call, that is, triStartTimer.

Return Values

The return status of the triCallMC operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.

triCallBC

Called when a procedure call needs to be made on a port to all recipients in a SUT.

```
TriStatus triCall
        (const TriComponentId* componentId,
         const TriPortId* tsiPortId,
         const TriSignatureId* signatureId,
         const TriParameterList* parameterList);
```

Parameters

`componentId`	identifier of the test component issuing the procedure call
`tsiPortId`	identifier of the test system interface port via which the procedure call is sent to the SUT Adapter
`signatureId`	identifier of the signature of the procedure call
`parameterList`	A list of encoded parameters which are part of the indicated signature. The parameters in parameterList are ordered as they appear in the TTCN-3 signature declaration.

Description

This operation is called by the TE when it executes a TTCN-3 broadcast call operation on a component port, which has been mapped to a TSI port. This operation is called by the TE for all TTCN-3 call operations if no system component has been specified for a test case, that is, only an MTC test component is created for a test case.

All in and inout procedure parameters contain encoded values. All out procedure parameters shall contain the distinct value of null since they are only of relevance in a reply to the procedure call but not in the procedure call itself.

The procedure parameters are the parameters specified in the TTCN-3 signature template. Their encoding has to be done in the TE prior to this TRI operation call.

On invocation of this operation, the SA can initiate the procedure call corresponding to the signature identifier signatureId and the TSI port tsiPortId.

The triCallBC operation shall return without waiting for the return of the issued procedure call. This might be achieved for example by spawning a new thread or process. This handling of this procedure call is, however, dependent on implementation of the TE.

This TRI operation returns TRI_OK on successful initiation of the procedure call, TRI_Error otherwise. No error shall be indicated by the SA in case the value of any out parameter is non-null. Notice that the return value of this TRI operation does not make any statement about the success or failure of the procedure call.

Note

An optional timeout value, which can be specified in the TTCN-3 ATS for a call operation, is not included in the triCallBC operation signature. The TE is responsible for addressing this issue by starting a timer for the TTCN-3 call operation in the PA with a separate TRI operation call, that is, triStartTimer.

Return Values

The return status of the triCallBC operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.

triReply

Called when a reply (to a call operation) needs to be made on a port to the single recipient.

```
TriStatus triReply
        (const TriComponentId* componentId,
         const TriPortId* tsiPortId,
         const TriAddress* sutAddress,
         const TriSignatureId* signatureId,
         const TriParameterList* parameterList,
         const TriParameter* returnValue);
```

Parameters

componentId	identifier of the replying test component
tsiPortId	iidentifier of the test system interface port via which the reply is sent to the SUT Adapter
sutAddress	(Optional) destination address within the SUT.
signatureId	identifier of the signature of the procedure call
parameterList	A list of encoded parameters which are part of the indicated signature. The parameters in parameter-List are ordered as they appear in the TTCN-3 signature declaration.
returnValue	(optional) encoded return value of the procedure call

Description

This operation is called by the TE when it executes a TTCN-3 unicast reply operation on a component port which has been mapped to a TSI port. This operation is called by the TE for all TTCN-3 reply operations if no system component has been specified for a test case, that is, only an MTC test component is created for a test case.

All out and inout procedure parameters and the return value contain encoded values. All in procedure parameters shall contain the distinct value of null since they are only of relevance to the procedure call but not in the reply to the call.

The parameterList contains procedure call parameters. These parameters are the parameters specified in the TTCN-3 signature template. Their encoding has to be done in the TE prior to this TRI operation call.

If no return type has been defined for the procedure signature in the TTCN-3 ATS, the distinct value null shall be passed for the return value.

On invocation of this operation, the SA can issue the reply to a procedure call corresponding to the signature identifier signatureId and the TSI port tsiPortId.

The triReply operation will return TRI_OK on successful execution of this operation, TRI_Error otherwise. No error shall be indicated by the SA in case the value of any in parameter or a non-defined return value is non-null.

Return Values

The return status of the triReply operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.

triReplyMC

Called when a reply (to a call operation) needs to be made on a port to the multiple recipients.

```
TriStatus triReplyMC
        (const TriComponentId* componentId,
         const TriPortId* tsiPortId,
         const TriAddressList* sutAddresses,
         const TriSignatureId* signatureId,
         const TriParameterList* parameterList,
         const TriParameter* returnValue);
```

Parameters

componentId	identifier of the replying test component
tsiPortId	iidentifier of the test system interface port via which the reply is sent to the SUT Adapter
sutAddresses	(Optional) destination addresses within the SUT.

`signatureId`	identifier of the signature of the procedure call
`parameterList`	A list of encoded parameters which are part of the indicated signature. The parameters in parameterList are ordered as they appear in the TTCN-3 signature declaration.
`returnValue`	(optional) encoded return value of the procedure call

Description

This operation is called by the TE when it executes a TTCN-3 multicast reply operation on a component port which has been mapped to a TSI port. This operation is called by the TE for all TTCN-3 reply operations if no system component has been specified for a test case, that is, only an MTC test component is created for a test case.

All out and inout procedure parameters and the return value contain encoded values. All in procedure parameters shall contain the distinct value of null since they are only of relevance to the procedure call but not in the reply to the call.

The parameterList contains procedure call parameters. These parameters are the parameters specified in the TTCN-3 signature template. Their encoding has to be done in the TE prior to this TRI operation call.

If no return type has been defined for the procedure signature in the TTCN-3 ATS, the distinct value null shall be passed for the return value.

On invocation of this operation, the SA can issue the reply to a procedure call corresponding to the signature identifier signatureId and the TSI port tsiPortId.

The triReplyMC operation will return TRI_OK on successful execution of this operation, TRI_Error otherwise. No error shall be indicated by the SA in case the value of any in parameter or a non-defined return value is non-null.

Return Values

The return status of the triReplyMC operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.

triReplyBC

Called when a reply (to a call operation) needs to be made on a port to all recipients in a SUT.

```
TriStatus triReply
        (const TriComponentId* componentId,
         const TriPortId* tsiPortId,
         const TriSignatureId* signatureId,
         const TriParameterList* parameterList,
         const TriParameter* returnValue);
```

Parameters

componentId	identifier of the replying test component
tsiPortId	iidentifier of the test system interface port via which the reply is sent to the SUT Adapter
signatureId	identifier of the signature of the procedure call
parameterList	A list of encoded parameters which are part of the indicated signature. The parameters in parameterList are ordered as they appear in the TTCN-3 signature declaration.
returnValue	(optional) encoded return value of the procedure call

Description

This operation is called by the TE when it executes a TTCN-3 broadcast reply operation on a component port which has been mapped to a TSI port. This operation is called by the TE for all TTCN-3 reply operations if no system component has been specified for a test case, that is, only an MTC test component is created for a test case.

All out and inout procedure parameters and the return value contain encoded values. All in procedure parameters shall contain the distinct value of null since they are only of relevance to the procedure call but not in the reply to the call.

The parameterList contains procedure call parameters. These parameters are the parameters specified in the TTCN-3 signature template. Their encoding has to be done in the TE prior to this TRI operation call.

If no return type has been defined for the procedure signature in the TTCN-3 ATS, the distinct value null shall be passed for the return value.

On invocation of this operation, the SA can issue the reply to a procedure call corresponding to the signature identifier signatureId and the TSI port tsiPortId.

The triReplyBC operation will return TRI_OK on successful execution of this operation, TRI_Error otherwise. No error shall be indicated by the SA in case the value of any in parameter or a non-defined return value is non-null.

Return Values

The return status of the triReplyBC operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.

triRaise

Called to raise an exception (during a call operation) on a port to the single recipient.

```
TriStatus triRaise
        (const TriComponentId* componentId,
         const TriPortId* tsiPortId,
         const TriAddress* sutAddress,
         const TriSignatureId* signatureId,
         const TriException* exception);
```

Parameters

componentId	identifier of the replying test component
tsiPortId	identifier of the test system interface port via which the reply is sent to the SUT Adapter
sutAddress	(Optional) destination address within the SUT.
signatureId	identifier of the signature of the procedure call
exception	the encoded exception

Description

This operation is called by the TE when it executes a TTCN-3 unicast raise operation on a component port which has been mapped to a TSI port. This operation is called by the TE for all TTCN-3 raise operations if no system component has been specified for a test case, that is, only an MTC test component is created for a test case.

The encoding of the exception has to be done in the TE prior to this TRI operation call.

On invocation of this operation the SA can raise an exception to a procedure call corresponding to the signature identifier signatureId and the TSI port tsiPortId.

The triRaise operation returns TRI_OK on successful execution of the operation, TRI_Error otherwise.

Return Values

The return status of the triRaise operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.

triRaiseMC

Called to raise an exception (during a call operation) on a port to the multiple recipient.

```
TriStatus triRaiseMC
        (const TriComponentId* componentId,
         const TriPortId* tsiPortId,
         const TriAddressList* sutAddresses,
         const TriSignatureId* signatureId,
         const TriException* exception);
```

Parameters

componentId	identifier of the replying test component
tsiPortId	identifier of the test system interface port via which the reply is sent to the SUT Adapter

sutAddresses	(Optional) destination addresses within the SUT.
signatureId	identifier of the signature of the procedure call
exception	the encoded exception

Description

This operation is called by the TE when it executes a TTCN-3 multicast raise operation on a component port which has been mapped to a TSI port. This operation is called by the TE for all TTCN-3 raise operations if no system component has been specified for a test case, that is, only an MTC test component is created for a test case.

The encoding of the exception has to be done in the TE prior to this TRI operation call.

On invocation of this operation the SA can raise an exception to a procedure call corresponding to the signature identifier signatureId and the TSI port tsiPortId.

The triRaiseMC operation returns TRI_OK on successful execution of the operation, TRI_Error otherwise.

Return Values

The return status of the triRaiseMC operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.

triRaiseBC

Called to raise an exception (during a call operation) on a port to all recipients in a SUT.

```
TriStatus triRaiseBC
        (const TriComponentId* componentId,
         const TriPortId* tsiPortId,
         const TriSignatureId* signatureId,
         const TriException* exception);
```

Parameters

componentId	identifier of the replying test component
tsiPortId	identifier of the test system interface port via which the reply is sent to the SUT Adapter
signatureId	identifier of the signature of the procedure call
exception	the encoded exception

Description

This operation is called by the TE when it executes a TTCN-3 broadcast raise operation on a component port which has been mapped to a TSI port. This operation is called by the TE for all TTCN-3 raise operations if no system component has been specified for a test case, that is, only an MTC test component is created for a test case.

The encoding of the exception has to be done in the TE prior to this TRI operation call.

On invocation of this operation the SA can raise an exception to a procedure call corresponding to the signature identifier signatureId and the TSI port tsiPortId.

The triRaiseBC operation returns TRI_OK on successful execution of the operation, TRI_Error otherwise.

Return Values

The return status of the triRaiseBC operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.

triSUTActionInformal

This operation is called by the TE when it executes a TTCN-3 SUT action operation, which only contains a string.

```
TriStatus triSUTActionInformal(char* description);
```

Parameters

`description`	An informal description of an action to be taken on the SUT.

Description

On invocation of this operation the SA shall initiate the described actions to be taken on the SUT, that is turn on, initialize, or send a message to the SUT, for example.

The `triSUTactionInformal` operation returns TRI_OK on successful execution of the operation, TRI_Error otherwise. Notice that the return value of this TRI operation does not make any statement about the success or failure of the actions to be taken on the SUT.

Return Values

The return status of the `triSUTactionInformal` operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.

triSUTActionTemplate

This operation is called by the TE when it executes a TTCN-3 SUT action operation, which uses a template.

```
TriStatus triSUTActionTemplate(const TriActionTemplate*
templateValue);
```

Parameters

`templateValue`	the encoded value of the action template

Description

The encoding of the action template value has to be done in the TE prior to this TRI operation call.

On invocation of this operation the SA shall initiate the actions to be taken on the SUT using the passed template value, turn on, initialize, or send a message to the SUT, for example.

The `triSUTactionTemplate` operation returns TRI_OK on successful execution of the operation, TRI_Error otherwise. Notice that the return value of this TRI operation does not make any statement about the success or failure of the actions to be taken on the SUT.

Return Values

The return status of the `triSUTactionTemplate` operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.

TE->PA Functions

These functions are implemented in the SA part of the TRI implementation and will be called from the TE.

triPAReset

This operation can be called by the TE at any time to reset the PA.

```
TriStatus triPAReset(void);
```

Description

The PA shall reset all timing activities which it is currently performing, stop all running timers, discard any pending time-outs of expired timers, for example.

The triPAReset operation returns TRI_OK in case the operation has been performed successfully, TRI_Error otherwise.

Return Values

The return status of the triPAReset operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.

triStartTimer

This operation is called by the TE when a timer needs to be started.

```
TriStatus triStartTimer
        (const TriTimerId *timerId,
         TriTimerDuration timerDuration);
```

Parameters

`timerId`	Identifier of the timer instance.
`timerDuration`	Duration of the timer in seconds.

Description

On invocation of this operation the PA shall start the indicated timer with the indicated duration. The timer runs from the value zero (0.0) up to the maximum specified by timerDuration. Should the timer indicated by timerId already be running it is to be restarted. When the timer expires the PA will call the triTimeout() operation with timerId.

The triStartTimer operation returns TRI_OK if the timer has been started successfully, TRI_Error otherwise.

Return Values

The return status of the triStartTimer operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.

triStopTimer

This operation is called by the TE when a timer is to be stopped.

```
TriStatus triStopTimer(const TriTimerId *timerId);
```

Parameters

`timerId`	Identifier of the timer instance.

Description

On invocation of this operation the PA shall use the timerId to stop the indicated timer instance. The stopping of an inactive timer, that is, a timer which has not been started or has already expired, should have no effect.

The triStopTimer operation returns TRI_OK if the operation has been performed successfully, TRI_Error otherwise. Notice that stopping an inactive timer is a valid operation. In this case TRI_OK shall be returned.

Return Values

The return status of the triStopTimer operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.

triReadTimer

This operation may be called by the TE when a TTCN-3 read timer operation is to be executed on the indicated timer.

```
TriStatus triReadTimer
        (const TriTimerId *timerId,
         TriTimerDuration *elapsedTime);
```

Parameters

timerId	Identifier of the timer instance.
elapsedTime	Value of the time elapsed since the timer has been started in seconds.

Description

On invocation of this operation the PA shall use the timerId to access the time that elapsed since this timer was started. The return value elapsedTime shall be provided in seconds. The reading of an inactive timer, that is, a timer which has not been started or already expired, shall return an elapsed time value of zero.

The triReadTimer operation returns TRI_OK if the operation has been performed successfully, TRI_Error otherwise.

Return Values

The return status of the triReadTimer operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.

triTimerRunning

This operation may be called by the TE when a TTCN-3 running timer operation is to be executed on the indicated timer.

```
TriStatus triTimerRunning
        (const TriTimerId *timerId,
```

```
        unsigned char *running);
```

Parameters

`timerId`	Identifier of the timer instance.
`running`	Status of the timer.

Description

On invocation of this operation the PA shall use the timerId to access the status of the timer. The operation sets running to the boolean value true if and only if the timer is currently running.

The triTimerRunning operation returns TRI_OK if the status of the timer has been successfully determined, TRI_Error otherwise.

Return Values

The return status of the triTimerRunning operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.

triExternalFunction

This operation is called by the TE when it executes a function which is defined to be TTCN-3 external (that is, all non-external functions are implemented within the TE).

```
TriStatus triExternalFunction
        (const TriFunctionId *functionId,
         TriParameterList *parameterList,
         TriParameter *returnValue);
```

Parameters

`functionId`	Identifier of the external function.
`parameterList`	A list of encoded parameters for the indicated function. The parameters in parameterList are ordered as they appear in the TTCN-3 function declaration.
`returnValue`	(Optional) encoded return value.

Description

In the invocation of a triExternalFunction operation by the TE, all in and inout function parameters contain encoded values. All out function parameters shall contain the distinct value of null since they are only of relevance in the return from the external function but not in its invocation. No error shall be indicated by the PA in case the value of any out parameter is non-null.

For each external function specified in the TTCN-3 ATS, the PA shall implement the behavior. On invocation of this operation the PA shall invoke the function indicated by the identifier functionId. It shall access the specified in and inout function parameters in parameterList, evaluate the external function using the values of these parameters, and compute values for inout and out parameters in parameterList. The operation shall then return encoded values for all inout and out function parameters, the distinct value of null for all in parameters, and the encoded return value of the external function.

If no return type has been defined for this external function in the TTCN-3 ATS, the distinct value null shall be used for the latter.

The triExternalFunction operation returns TRI_OK if the PA completes the evaluation of the external function successfully, TRI_Error otherwise.

Note

Whereas all other TRI operations are considered to be non-blocking, the triExternalFunction operation is considered to be blocking. That means that the operation shall not return before the indicated external function has been fully evaluated. External functions have to be implemented carefully as they could cause deadlock of test component execution or even the entire test system implementation.

Return Values

The return status of the triExternalFunction operation. The return status indicates the local success (TRI_OK) or failure (TRI_Error) of the operation.

TCI API

TCI type declarations

String

String is a synonym for `char*`

TciVerdictValue

TciVerdictValue is a synonym for `unsigned long int`.

May take the value equal to the one of the following constants:

```
TCI_VERDICT_NONE;
TCI_VERDICT_PASS;
TCI_VERDICT_INCONC;
TCI_VERDICT_FAIL;
TCI_VERDICT_ERROR;
```

`TciObjidElemValue`

This type is used for string and integer representations of object identifier element.

```
typedef struct _TciObjidElemValue
{
  String elem_as_ascii;
  long int elem_as_number;
  void* aux;
};
```

TciObjidValue

This type is used to represent a list of `objid` elements. No special values mark the end of elements. The length field shall be used to traverse this array properly.

```
typedef struct _TciObjidValue
{
  long int length;
  TciObjidElemValue *elements;
}*;
```

TciCharStringValue

This type is used to represent character string.

No special values mark the end of string. The length field shall be used to traverse this array properly.

```
typedef struct _TciCharStringValue
{
  unsigned long int length;
  char* string;
}*;
```

TciUCValue

Synonym for `unsigned char[4]`. Represents group, plane, row and cell of universal character as defined in char (group, plane, row, cell) format.

TciUCReturnValue

Synonym for `unsigned char*`. This type is used instead of TciUCValue for return values in functions.

TciUCStringValue

This type is used for textual representation of universal character string.

No special values mark the end of string. The length field shall be used to traverse this array properly.

```
typedef struct _TciUCStringValue
{
  unsigned long int length;
  TciUCValue *string;
}*;
```

TciModuleIdType

Synonym for QualifiedName. This type is used to represent module name.

TciModuleIdListType

This type is used to represent the list of modules.

No special values mark the end of an `idList`. The length field shall be used to traverse this array properly.

```
typedef struct _TciModuleIdListType
{
  long int length;
  TciModuleIdType *idList;
}*;
```

TciModuleParameterIdType

Synonym for QualifiedName. This type is used to represent the qualified name of module parameter as defined in TTCN-3 ATS.

TciModuleParameterType

This type is used to represent the parameter name and the default value of a module parameter.

```
typedef struct _TciModuleParameterType
{
  TciModuleParameterIdType parName;
  TciValue defaultValue;
};
```

TciModuleParameterListType

This type is used to represent the module parameters of a TTCN-3 module.

No special values mark the end of a `modParList`. The length field shall be used to traverse this array properly.

```
typedef struct _TciModuleParameterListType
{
  long int length;
  TciModuleParameterType *modParList;
}*;
```

TciTestCaseIdType

Synonym for QualifiedName. This type is used to represent the qualified name of test case as defined in TTCN-3 ATS.

TciTestCaseIdListType

This type is used to represent the list of test cases.

No special values mark the end of `idList`. The length field shall be used to traverse this array properly.

```
typedef struct _TciTestCaseIdListType
{
  long int length;
  TciTestCaseIdType *idList;
}*;
```

TciTestCaseParameterIdType

Synonym for String. This type is used to represent the name of test case formal parameter as defined in TTCN-3 ATS.

TciTestCaseParameterIdListType

This type is used to represent the list of test case formal parameter names.

No special values mark the end of `idList`. The length field shall be used to traverse this array properly.

```
typedef struct _TciTestCaseParameterIdListType
{
  long int length;
  TciTestCaseParameterIdType *idList;
}*;
```

TciParameterPassingModeType

This type is used to represent the passing mode of a test case parameter: in, inout or out.

```
typedef enum
{
  TCI_IN_PAR,
  TCI_INOUT_PAR,
  TCI_OUT_PAR
};
```

TciParameterTypeType

This type is used to represent the type of a test case parameter as well as its passing mode.

```
typedef struct _TciParameterTypeType
{
  TciParameterPassingModeType parPassMode;
  TciType parType;
};
```

TciParameterTypeListType

This type is used to represent the types and passing modes of all test case formal parameters.

No special values mark the end of parList. The length field shall be used to traverse this array properly.

```
typedef struct _TciParameterTypeListType
{
  long int length;
  TciParameterTypeType *parList;
}*;
```

TciParameterType

This type is used to represent the actual value of a test case parameter as well as its passing mode.

```
typedef struct _TciParameterType
{
  String parName;
  TciParameterPassingModeType parPassMode;
  TciValue parValue;
};
```

TciParameterListType

This type is used to represent the actual values of all test case parameters.

No special values mark the end of parList. The length field shall be used to traverse this array properly.

```
typedef struct _TciParameterListType
```

```
{
  long int length;
  TciParameterType *parList;
}*;
```

TciTestComponentKindType

This type is used to represent component kind: internal control component, main test component, parallel test component or system component.

```
typedef enum
{
  TCI_CTRL_COMP,
  TCI_MTC_COMP,
  TCI_PTC_COMP,
  TCI_SYS_COMP,
  TCI_ALIVE_COMP
};
```

TciTypeClassType

This type is used to represent the all possible kinds of values that may by handled in runtime.

```
typedef enum
{
  TCI_ADDRESS_TYPE,
  TCI_ANYTYPE_TYPE,
  TCI_BITSTRING_TYPE,
  TCI_BOOLEAN_TYPE,
  TCI_CHAR_TYPE,
  TCI_CHARSTRING_TYPE,
  TCI_COMPONENT_TYPE,
  TCI_ENUMERATED_TYPE,
  TCI_FLOAT_TYPE,
  TCI_HEXSTRING_TYPE,
  TCI_INTEGER_TYPE,
  TCI_OBJID_TYPE,
  TCI_OCTETSTRING_TYPE,
  TCI_RECORD_TYPE,
  TCI_RECORD_OF_TYPE,
  TCI_SET_TYPE,
  TCI_SET_OF_TYPE,
  TCI_UNION_TYPE,
  TCI_UNIVERSAL_CHAR_TYPE,
  TCI_UNIVERSAL_CHARSTRING_TYPE,
  TCI_VERDICT_TYPE,
  TCI_DEFAULT_TYPE,
  TCI_PORT_TYPE,
  TCI_TIMER_TYPE,
  TCI_TEMPLATE_TYPE
};
```

ComponentStatus

This type is used to represent the component status.

```
typedef enum
{
  inactiveC,
  runningC,
  stoppedC,
  killedC
};
```

TimerStatus

This type is used to represent the timer status.

```
typedef enum
{
  runningT,
  inactiveT,
  expiredT
};
```

PortStatus

This type is used to represent the port status.

```
typedef enum
{
  startedP,
  haltedP,
  stoppedP
};
```

TciSignatureIdType

Synonym for QualifiedName. This type is used to represent the qualified name of a procedure signature as defined in TTCN-3 ATS.

TciBehaviourIdType

Synonym for QualifiedName. This type is used to represent the qualified name of a function or `altstep` as defined in TTCN-3 ATS.

TciValueList

This type is used to represent the list of values.

No special values mark the end of `valueList`. The length field shall be used to traverse this array properly.

```
typedef struct _TciValueList
{
  long int length;
  TciValue *valueList;
}*;
```

TciValueDifference

This type is used to represent the difference between the value and template.

It contains value, template and a meaningful description for the reason of this difference

```
typedef struct _TciValueDifference
{
  TciValue val;
  TciValueTemplate tmpl;
  String desc;
};
```

TciValueDifferenceList

This type is used to represent the list of difference between the value and template.

No special values mark the end of `diffList`. The length field shall be used to traverse this array properly.

```
typedef struct _TciValueDifferenceList
{
  long int length;
  TciValueDifference *diffList;
}*;
```

TCI Type Interface API

tciGetDefiningModule

Lookups the module identifier that defines a specified type.

```
TciModuleIdType tciGetDefiningModule(TciType typeId);
```

Parameters

`typeId`	Identifier of the type instance.

Description

This operation may be called to lookup module identifier of the module in which type is defined. If type is a TTCN-3 base type, e.g. boolean, integer, etc. then distinct NULL value is returned.

Return Values

Returns the module identifier for the user-defined types, NULL for built-in types.

tciGetName

Lookups the name of the specified type

```
String tciGetName(TciType typeId);
```

Parameters

`typeId`	Identifier of the type instance.

Description

This operation may be called to lookup the name of the type as defined in TTCN-3

abstract test suite specification.

Return Values

Returns the name of the type as defined in the TTCN-3 module

tciGetTypeClass

Lookups type class of the specified type

```
TciTypeClassType tciGetTypeClass(TciType typeId);
```

Parameters

`typeId`	Identifier of the type instance.

Description

This operation may be called to lookup the type class of the type. Array types are represented as types from RECORD_OF type class.

Some of the type classes (DEFAULT, PORT, TIMER, TEMPLATE - for formal template parameters) are not specified in the standard. These classes were intentionally added in Rational Systems Tester since during testsuite execution TCI TL may provide the values of above-mentioned type classes. However there are no functions to process such values.

Return Values

Returns the type class of the respective type.

tciNewInstance

Creates new value of specified type

```
TciValue tciNewInstance(TciType typeId);
```

Parameters

`typeId`	Identifier of the type instance.

Description

This operation may be called to instantiate new value of the specified type.

The initial value of the created value is undefined.

This function may be called only for types from the following type classes:

```
BOOLEAN
CHAR
FLOAT
UNIVERSAL_CHAR
VERDICT
ENUMERATED
UNIVERSAL_CHARSTRING
OBJID
ADDRESS
RECORD
SET
RECORD_OF
SET_OF
UNION
ANYTYPE
INTEGER
BITSTRING
CHARSTRING
HEXSTRING
```

```
OCTETSTRING
```

When called for type from other type classes tciNewInstance will produce error and return NULL value.

Return Values

Returns a freshly created (not-initialized) value of the given type.

Returns NULL in case of error.

tciGetTypeEncoding

Lookups the encoding attribute of specified type

```
String tciGetTypeEncoding(TciType typeId);
```

Parameters

`typeId`	Identifier of the type instance.

Description

This operation returns the type encoding attribute as defined in TTCN-3, if any. If no encoding attribute is defined the distinct value NULL is returned.

Return Values

Returns the encoding attribute as defined in the TTCN-3 module.

Returns NULL if attribute was not specified.

tciGetTypeEncodingVariant

Lookups the encoding attribute of specified type

```
String tciGetTypeEncodingVariant(TciType typeId);
```

Parameters

`typeId`	Identifier of the type instance.

Description

This operation returns the type encoding variant attribute as defined in TTCN-3, if any. If no encoding variant attribute is defined the distinct value NULL is returned.

Return Values

Returns the encoding variant attribute as defined in the TTCN-3 module.

Returns NULL if attribute was not specified.

tciGetTypeExtension

Lookups the extension attribute of specified type

```
String* tciGetTypeExtension(TciType typeId);
```

Parameters

`typeId`	Identifier of the type instance.

Description

This operation returns the type encoding extension attribute as defined in TTCN-3, if any. If no extension variant attribute is defined the distinct value NULL is returned.

This function returns NULL terminated string array. It contains more than one element for compound types. The first element in the array represents extension attribute for the type itself while subsequent elements represent extension attributes for the fields. Empty string (i.e. ““) denotes absence of extension attribute.

Return Values

Returns the extension attribute as defined in the TTCN-3 module.

Returns NULL if attribute was not specified.

TCI Value Interface API

Generic operations

tciGetType

Lookups type identifier of the specified value

```
TciType tciGetType(TciValue valueId);
```

Parameters

`valueId`	Identifier of the value instance.

Description

This operation may be called to lookup type identifier of the respective value.

Return Values

Returns the type of the specified value.

Returns NULL in case of error.

tciNotPresent

Checks whether specified value is 'omit'

```
unsigned char tciNotPresent(TciValue valueId);
```

Parameters

`valueId`	Identifier of the value instance.

Description

This operation may be called to check whether respective value is omitted or not. If `valueId` equals to NULL then value is also treated as omitted.

Return Values

Returns true if the specified value is 'omit' or NULL, false otherwise

tciGetValueEncoding

Lookups the encoding attribute of specified value

```
String tciGetValueEncoding(TciValue valueId);
```

Note

This function is not supported!

Parameters

`valueId`	Identifier of the value instance.

Description

This operation returns the value encoding attribute as defined in TTCN-3, if any. If no encoding attribute is defined the distinct value NULL is returned.

Return Values

Returns the encoding attribute as defined in the TTCN-3 module.

Returns NULL if attribute was not specified.

tciGetValueEncodingVariant

Lookups the encoding attribute of specified value.

```
String tciGetValueEncodingVariant(TciValue valueId);
```

Note

This function is not supported!

Parameters

`valueId`	Identifier of the value instance.

Description

This operation returns the value encoding variant attribute as defined in TTCN-3, if any. If no encoding variant attribute is defined the distinct value NULL is returned.

Return Values

Returns the encoding variant attribute as defined in the TTCN-3 module.

Returns NULL if attribute was not specified.

Integer Value Interface

tciGetIntAbs

Lookups absolute value of an integer as an ASCII string

```
String tciGetIntAbs(TciValue valueId);
```

Parameters

`valueId`	Identifier of the value instance.

Description

This operation may be called to obtain absolute value of an integer value.

Absolute value is returned as an 10-base ASCII string. Since integer value may be filled digit-by-digit there exist intermediate states in which integer value has invalid contents. Using this function for such undefined values will lead to error and NULL string will be returned.

Return Values

Returns the (10-base) integer absolute value as an ASCII string.

Returns NULL if specified value is not a valid integer value.

tciGetIntNumberOfDigits

Lookups the number of digits (length) of an integer value

```
unsigned long int tciGetIntNumberOfDigits(in TciValue
valueId);
```

Parameters

`valueId`	Identifier of the value instance.

Description

This operation may be called to obtain the number of digits (the length in other words) of an integer value. Number of digits of an integer value may be changed by calling tciSetIntNumberOfDigits or setting digit using tciSetIntDigit beyond the length of an integer.

Return Values

Returns the number of digits in an integer value.

tciGetIntSign

Lookups the sign (+/-) of an integer value

```
unsigned char tciGetIntSign(in TciValue valueId);
```

Parameters

`valueId`	Identifier of the value instance.

Description

This operation may be called to obtain the sign of an integer value.

True corresponds to positive values and the zero.

False corresponds to the negative values.

Return Values

Returns true if the number is positive or zero, false otherwise

tciGetIntDigit

Lookups the digit of an integer value at specified position

```
unsigned char tciGetIntDigit(in TciValue valueId, unsigned
long int position);
```

Parameters

`valueId`	Identifier of the value instance.
`position`	Zero based offset from the least significant digit of an integer

Description

This operation may be called to obtain the value of a digit at specified position. Position '0' corresponds to the least significant digit of an integer. For example, in a value '12345' position '0' corresponds to the digit '5'.

Return Values

Returns the value of the digit at specified position

tciSetIntAbs

Sets absolute value of an integer using ASCII string

```
void tciSetIntAbs(TciValue valueId, String absValue);
```

Parameters

`valueId`	Identifier of the value instance.
`absValue`	10-base ASCII string representing absolute integer value

Description

This operation may be called to set absolute value of an integer value.

Absolute value is specified using 10-base ASCII string.

Due to the limitations in Rational Systems Tester runtime system it's possible to

specify only those values that fit into 64-bit signed integer.

Return Values

None

tciSetIntNumberOfDigits

Sets the number of digits (length) in an integer value

```
void tciSetIntNumberOfDigits(in TciValue valueId, unsigned
long int nDigits);
```

Parameters

`valueId`	Identifier of the value instance.
`nDigits`	Number of digits to be set in an integer value

Description

This operation may be called to set the number of digits (the length in other words) of an integer value. If specified number of digits is greater than current length then integer is expanded and digits are marked as not-initialized. If specified number of digits is lower then the current length of a value then the value is truncated and all digits that lie beyond new length are lost.

Return Values

None

tciSetIntSign

Sets the sign (+/-) of an integer value

```
void tciSetIntSign(in TciValue valueId, unsigned char
sign);
```

Parameters

`valueId`	Identifier of the value instance.
`sign`	boolean value representing either '+' or '-

Description

This operation may be called to set the sign of an integer value.

True corresponds to positive values and the zero.

False corresponds to the negative values.

Return Values

None

tciSetIntDigit

Lookups the digit of an integer value at specified position

```
void tciSetIntDigit(in TciValue valueId, unsigned long int
position, unsigned char digit);
```

Parameters

`valueId`	Identifier of the value instance.
`position`	Zero based offset from the least significant digit of an integer
`digit`	The value to be set to the digit at the specified position

Description

This operation may be called to set the value of a digit at specified position. Position '0' corresponds to the least significant digit of an integer. For example, in a value '12345' position '0' corresponds to the digit '5'.

Return Values

None

Float Value Interface

tciGetFloatValue

Returns the float value of a specified value

```
double tciGetFloatValue(TciValue valueId);
```

Parameters

`valueId`	Identifier of the value instance.

Description

This operation may be called to obtain the float value of a specified value.

Return Values

Returns the float value of this TTCN-3 float

tciSetFloatValue

Sets the value to a specified float value

```
void tciSetFloatValue(TciValue valueId, double floatValue);
```

Parameters

`valueId`	Identifier of the value instance.
`floatValue`	The float value to assign to the specified value

Description

This operation may be called to set the value to a specified float value.

Return Values

None

Boolean Value Interface

tciGetBooleanValue

Returns the boolean value of a specified value

```
unsigned char tciGetBooleanValue(TciValue valueId);
```

Parameters

`valueId`	Identifier of the value instance.

Description

This operation may be called to obtain the boolean value of a specified value.

Return Values

Returns the boolean value of this TTCN-3 boolean

tciSetBooleanValue

Sets the value to a specified boolean value

```
void tciSetBooleanValue(TciValue valueId, unsigned char
booleanValue);
```

Parameters

`valueId`	Identifier of the value instance.
`booleanValue`	The boolean value to assign to the specified value

Description

This operation may be called to set the value to a specified boolean value.

Return Values

None

Object Identifier Value Interface

tciGetTciObjidValue

Returns the `objid` value of a specified value

```
TciObjidValue tciGetTciObjidValue(TciValue valueId);
```

Parameters

`valueId`	Identifier of the value instance.

Description

This operation may be called to obtain the objid value of a specified value.

Return Values

Returns the objid value of this TTCN-3 object identifier

tciSetObjidValue

Sets the value to a specified objid value

```
void tciSetObjidValue(TciValue valueId, TciObjidValue
objidValue);
```

Parameters

`valueId`	Identifier of the value instance.
`objidValue`	The objid value that should be assigned to specified value

Description

This operation may be called to set the value to a specified objid value.

Return Values

None

Char Value Interface

tciGetCharValue

Returns the character value of a specified value

```
unsigned char tciGetCharValue(TciValue valueId);
```

Parameters

`valueId`	Identifier of the value instance.

Description

This operation may be called to obtain the character value of a specified value.

Return Values

Returns the character value of this TTCN-3 char

tciSetCharValue

Sets the value to a specified character value

```
void tciSetCharValue(TciValue valueId, unsigned char
charValue);
```

Parameters

`valueId`	Identifier of the value instance.
`charValue`	The character value that should be assigned to specified value

Description

This operation may be called to set the value to a specified character value.

Return Values

None

Universal Char Value Interface

tciGetUniversalCharValue

Returns the universal character value of a specified value

```
TciUCReturnValue tciGetUniversalCharValue(TciValue
```

```
valueId);
```

Parameters

`valueId`	Identifier of the value instance.

Description

This operation may be called to obtain the universal character value of a specified value. TciUCReturnValue type was introduced in Rational Systems Tester due to semantics of C programming language. Originally this functions return the value of TciUCValue type but TciUCValue is defined as fixed size array and C cannot return such arrays.

Return Values

Returns the universal character value of this TTCN-3 universal char

tciSetUniversalCharValue

Sets the value to a specified character value

```
void tciSetCharValue(TciValue valueId, TciUCValue
uniCharValue);
```

Parameters

`valueId`	Identifier of the value instance.
`uniCharValue`	Universal char value that should be assigned to specified value

Description

This operation may be called to set the value to a specified universal character value.

Return Values

None

Charstring Value Interface

tciGetCStringValue

Returns the character string of a specified value

```
TciCharStringValue tciGetCStringValue(TciValue valueId);
```

Parameters

`valueId`	Identifier of the value instance.

Description

This operation may be called to obtain the character string of a specified value.

Return Values

Returns the character string of this TTCN-3 charstring

tciSetCStringValue

Sets the value to a specified character string

```
void tciSetCStringValue(TciValue valueId,
TciCharStringValue charStrValue);
```

Parameters

`valueId`	Identifier of the value instance.
`charStrValue`	character string that should be assigned to specified value

Description

This operation may be called to set the value to a specified character string.

Return Values

None

tciGetCharstringValue

Returns the NULL terminated character string of a specified value

```
String tciGetCharstringValue(TciValue valueId);
```

Parameters

`valueId`	Identifier of the value instance.

Description

This operation may be called to obtain the NULL terminated character string of a specified value.

Return Values

Returns the NULL terminated character string of this TTCN-3 charstring

tciSetCharstringValue

Sets the value to a specified NULL terminated character string

```
void tciSetCharstringValue(TciValue valueId, String
charStrValue);
```

Parameters

`valueId`	Identifier of the value instance.
`charStrValue`	NULL terminated character string that should be assigned to specified value

Description

This operation may be called to set the value to a specified NULL terminated character string.

Return Values

None

tciGetCStringCharValue

Returns the character at specified position of charstring

```
unsigned char tciGetCStringCharValue(TciValue valueId,
unsigned long int position);
```

Parameters

`valueId`	Identifier of the value instance.
`position`	Zero based offset from the character string

Description

This operation may be called to obtain the character at specified position of TTCN-3 charstring. Position 0 denotes the first char of the character string. Valid values for position are from 0 to "`length-1`". Characters are numbered from left to right.

Return Values

Returns the character at specified position of the TTCN-3 charstring

tciSetCStringCharValue

Sets the character at specified position of charstring

```
void tciSetCStringCharValue(TciValue valueId, unsigned long
int position, unsigned char charValue);
```

Parameters

`valueId`	Identifier of the value instance.
`position`	Zero based offset from the character string
`charvalue`	character to be set

Description

This operation may be called to set the character at specified position of TTCN-3 charstring. Position 0 denotes the first char of the character string.Valid values for position are from 0 to "`length-1`". Characters are numbered from left to right.

Return Values

None

tciGetCStringLength

Returns the length of the specified charstring value

```
unsigned long int tciGetCStringLength(TciValue valueId);
```

Parameters

`valueId`	Identifier of the value instance.

Description

This operation may be called to obtain the length of the TTCN-3 charstring. If specified value is omitted then zero is returned.

Return Values

Returns the length of the specified charstring value in chars.

Returns zero if value is 'omit'.

tciSetCStringLength

Sets the length of the specified charstring value

```
void tciSetCStringLength(TciValue valueId, unsigned long
int length);
```

Parameters

`valueId`	Identifier of the value instance.
`length`	New length to be set to the specified charstring value

Description

This operation may be called to change the length of the TTCN-3 charstring. If new length is greater than current one then string is padded with '\0' characters. If new length is lower than current one then string is truncated.

Return Values

None

Universal Charstring Value Interface

tciGetUCStringValue

Returns the universal character string of a specified value

```
TciUCStringValue tciGetUCStringValue(TciValue valueId);
```

Parameters

`valueId`	Identifier of the value instance.

Description

This operation may be called to obtain the textual representation of the universal character string of a specified value.

Return Values

Returns the textual representation of the universal character string of the specified value.

tciSetUCStringValue

Sets the value to a specified universal character string according to its textual representation

```
void tciSetUCStringValue(TciValue valueId, TciUCStringValue
uniCharStrValue);
```

Parameters

`valueId`	Identifier of the value instance.
`uniCharStrValue`	Universal character string value that should be assigned to specified value

Description

This operation may be called to set the value to a specified universal character string, which is defined by means of textual representation.

Return Values

None

tciGetUCStringCharValue

Returns the universal character at specified position of universal charstring.

```
TciUCReturnValue tciGetUCStringCharValue(TciValue valueId,
unsigned long int position);
```

Parameters

`valueId`	Identifier of the value instance.
`position`	Zero based offset from the character string

Description

This operation may be called to obtain the universal character at specified position of TTCN-3 universal charstring. Position 0 denotes the first universal char of the universal character string. Valid values for position are from 0 to "`length-1`". Universal characters are numbered from left to right. TciUCReturnValue type was introduced in Rational Systems Tester due to seman-

tics of C programming language. Originally this functions return the value of TciUCValue type but a TciUCValue is defined as a fixed size array and C cannot return such arrays.

Return Values

Returns the universal character at specified position of the TTCN-3 universal charstring

tciSetUCStringCharValue

Sets the universal character at specified position of universal charstring

```
void tciSetUCStringCharValue(TciValue valueId, unsigned
long int position, TciUCValue uniCharValue);
```

Parameters

`valueId`	Identifier of the value instance.
`position`	Zero based offset from the start of the string
`uniCharValue`	Universal character to be set

Description

This operation may be called to set the universal character at specified position of TTCN-3 universal charstring. Position 0 denotes the first universal char of the universal character string. Valid values for position are from 0 to "`length-1`". Universal characters are numbered from left to right.

Return Values

None

tciGetUCStringLength

Returns the length of the specified universal charstring value

```
unsigned long int tciGetUCStringLength(TciValue valueId);
```

Parameters

`valueId`	Identifier of the value instance.

Description

This operation may be called to obtain the length of the TTCN-3 universal charstring in universal characters. If specified value is omitted then zero is returned.

Return Values

Returns the length of the specified universal charstring value in universal chars.

Returns zero if value is 'omit'.

tciSetUCStringLength

Sets the length of the specified universal charstring value

```
void tciSetUCStringLength(TciValue valueId, unsigned long
int length);
```

Parameters

`valueId`	Identifier of the value instance.
`length`	New length to be set to the specified universal charstring value

Description

This operation may be called to change the length of the TTCN-3 charstring. If new length is greater than current one then string is padded with 'char (255,255,255,255)' universal characters. If new length is lower than current one then string is truncated.

Return Values

None

Bitstring Value Interface

tciGetBStringValue

Returns the textual representation of the bit string of a specified value

```
String tciGetBStringValue(TciValue valueId);
```

Parameters

`valueId`	Identifier of the value instance.

Description

This operation may be called to obtain the textual representation of the bit string of a specified value. E.g. the textual representation of 0101 is '0101'B. The textual representation of the empty TTCN-3 bitstring is `''B`, with length zero.

Return Values

Returns the textual representation of the bit string of this TTCN-3 bitstring

tciSetBStringValue

Sets the value to a specified bit string according to its textual representation.

```
void tciSetBStringValue(TciValue valueId, String
bitStrValue);
```

Parameters

`valueId`	Identifier of the value instance.
`bitStrValue`	textual representation of the bit string that should be assigned to specified value

Description

This operation may be called to set the value to a specified bit string according to its textual representation. E.g. to assign bitstring 0101 the value of `bitStrValue` formal parameter should be '0101'B

Return Values

None

tciGetBStringBitValue

Returns the bit (0 or 1) at specified position of bitstring

```
long int tciGetBStringBitValue(TciValue valueId, unsigned
long int position);
```

Parameters

`valueId`	Identifier of the value instance.
`position`	Zero based offset from the start of the string

Description

This operation may be called to obtain the bit (0 or 1) at specified position of TTCN-3 bitstring. Position 0 denotes the first bit of the bit string. Valid values for position are from 0 to "`length-1`". Bits are numbered from left to right.

Return Values

Returns the bit (0 or 1) at specified position of the TTCN-3 bitstring

tciSetBStringBitValue

Sets the bit (0 or 1) at specified position of bitstring

```
void tciSetBStringBitValue(TciValue valueId, unsigned long
int position, long int bitValue);
```

Parameters

`valueId`	Identifier of the value instance.
`position`	Zero based offset from the start of the hex string
`bitValue`	bit (0 or 1) to be set

Description

This operation may be called to set the bit (0 or 1) at specified position of TTCN-3 bitstring. Position 0 denotes the first bit of the bit string. Valid values for position are from 0 to "`length-1`". Bits are numbered from left to right.

Return Values

None

tciGetBStringLength

Returns the length of the specified bitstring value

```
unsigned long int tciGetBStringLength(TciValue valueId);
```

Parameters

`valueId`	Identifier of the value instance.

Description

This operation may be called to obtain the length of the TTCN-3 bitstring.

If specified value is omitted then zero is returned.

Return Values

Returns the length of the specified bitstring value in bits.

Returns zero if value is 'omit'.

tciSetBStringLength

Sets the length of the specified bitstring value

```
void tciSetBStringLength(TciValue valueId, unsigned long
int length);
```

Parameters

`valueId`	Identifier of the value instance.
`length`	New length to be set to the specified universal charstring value

Description

This operation may be called to change the length of the TTCN-3 bitstring.

If new length is greater than current one then string is expanded and bits are marked as not-initialized. If new length is lower than current one then string is truncated.

Return Values

None

Octetstring Value Interface

tciGetOStringValue

Returns the textual representation of the octet string of a specified value

```
String tciGetOStringValue(TciValue valueId);
```

Parameters

`valueId`	Identifier of the value instance.

Description

This operation may be called to obtain the textual representation of the octet string of a specified value. E.g. the textual representation of 0xCAFFEE is 'CAFFEE'O. The textual representation of the empty TTCN-3 octetstring is "O, while its length is zero.

Return Values

Returns the textual representation of the octet string of this TTCN-3 octetstring

tciSetOStringValue

Sets the value to a specified octet string according to its textual representation.

```
void tciSetOStringValue(TciValue valueId, String
octStrValue);
```

Parameters

`valueId`	Identifier of the value instance.
`position`	Zero based offset from the start of the hex string
`octStrValue`	textual representation of the octet string that should be assigned to specified value

Description

This operation may be called to set the value to a specified octet string according to its textual representation. E.g. to assign octetstring 0xABCD the value of `octStrValue` formal parameter should be 'ABCD'O

Return Values

None

tciGetOStringOctetValue

Returns the octet (integer in range 0..255) at specified position of octetstring

```
long int tciGetOStringOctetValue(TciValue valueId, unsigned
long int position);
```

Parameters

`valueId`	Identifier of the value instance.
`position`	Zero based offset from the start of the string

Description

This operation may be called to obtain the octet (integer in range 0..255) at specified position of TTCN-3 octetstring. Position 0 denotes the first octet of the octet string. Valid values for position are from 0 to "`length-1`". Octets are numbered from left to right.

Return Values

Returns the octet (integer in range 0..255) at specified position of the TTCN-3 octetstring

tciSetOStringOctetValue

Sets the octet (integer in range 0..255) at specified position of octetstring

```
void tciSetOStringOctetValue(TciValue valueId, unsigned
long int position, long int octValue);
```

Parameters

`valueId`	Identifier of the value instance.
`position`	Zero based offset from the start of the hex string
`octValue`	octet (integer in range 0..255) to be set

Description

This operation may be called to set the octet (integer in range 0..255) at specified position of TTCN-3 octetstring. Position 0 denotes the first octet of the octet string. Valid values for position are from 0 to "`length-1`". Octets are numbered from left to right.

Return Values

None

tciGetOStringLength

Returns the length of the specified octetstring value

```
unsigned long int tciGetOStringLength(TciValue valueId);
```

Parameters

`valueId`	Identifier of the value instance.

Description

This operation may be called to obtain the length of the TTCN-3 octetstring. If specified value is omitted then zero is returned.

Return Values

Returns the length of the specified octetstring value in octets.

Returns zero if value is 'omit'.

tciSetOStringLength

Sets the length of the specified octetstring value

```
void tciSetOStringLength(TciValue valueId, unsigned long
int length);
```

Parameters

`valueId`	Identifier of the value instance.
`length`	New length to be set to the specified universal char-string value

Description

This operation may be called to change the length of the TTCN-3 octetstring. If new length is greater than current one then string is expanded and octets are marked as not-initialized. If new length is lower than current one then string is truncated.

Return Values

None

Hexstring Value Interface

tciGetHStringValue

Returns the textual representation of the hex string of a specified value

```
String tciGetHStringValue(TciValue valueId);
```

Parameters

`valueId`	Identifier of the value instance.

Description

This operation may be called to obtain the textual representation of the hex string of a specified value. E.g. the textual representation of 0xAFFEE is 'AFFEE'H. The textual representation of the empty TTCN-3 hexstring is ''H, while its length is zero.

Return Values

Returns the textual representation of the hex string of this TTCN-3 hexstring

tciSetHStringValue

Sets the value to a specified hex string according to its textual representation.

```
void tciSetHStringValue(TciValue valueId, String
hexStrValue);
```

Parameters

`valueId`	Identifier of the value instance.
`hexStrValue`	textual representation of the hex string that should be assigned to specified value

Description

This operation may be called to set the value to a specified hex string according to its textual representation. E.g. to assign hexstring 0xABC the value of `hexStrValue` formal parameter should be 'ABC'H

Return Values

None

tciGetHStringHexValue

Returns the hexadecimal digit (integer in range 0..15) at specified position of hexstring

```
long int tciGetHStringHexValue(TciValue valueId, unsigned
long int position);
```

Parameters

`valueId`	Identifier of the value instance.
`position`	zero based offset from the start of the hex string

Description

This operation may be called to obtain the hex digit (integer in range 0..15) at specified position of TTCN-3 hexstring. Position 0 denotes the first hex of the hex string. Valid values for position are from 0 to "`length-1`". Hex digits are numbered from left to right.

Return Values

Returns the hexadecimal digit (integer in range 0..15) at specified position of the TTCN-3 hexstring

tciSetHStringHexValue

Sets the hex digit (integer in range 0..15) at specified position of hexstring

```
void tciSetHStringHexValue(TciValue valueId, unsigned long
int position, long int hexValue);
```

Parameters

`valueId`	Identifier of the value instance.
`position`	Zero based offset from the start of the hex string
`hexValue`	Hex digit (integer in range 0..15) to be set

Description

This operation may be called to set the hex digit (integer in range 0..15) at specified position of TTCN-3 hexstring. Position 0 denotes the first hex digit of the hex string. Valid values for position are from 0 to "`length-1`". Hex digits are numbered from left to right.

Return Values

None

tciGetHStringLength

Returns the length of the specified hexstring value

```
unsigned long int tciGetHStringLength(TciValue valueId);
```

Parameters

`valueId`	Identifier of the value instance.

Description

This operation may be called to obtain the length of the TTCN-3 hexstring. If specified value is omitted then zero is returned.

Return Values

Returns the length of the specified hexstring value in hex digits.

Returns zero if value is 'omit'.

tciSetHStringLength

Sets the length of the specified hexstring value

```
void tciSetHStringLength(TciValue valueId, unsigned long
int length);
```

Parameters

`valueId`	Identifier of the value instance.
`length`	New length to be set to the specified universal charstring value

Description

This operation may be called to change the length of the TTCN-3 hexstring. If new length is greater than current one then string is expanded and hex digits are marked as not-initialized. If new length is lower than current one then string is truncated.

Return Values

None

Record/Set Value Interface

tciGetRecFieldValue

Returns the field value of specified record/set

```
TciValue tciGetRecFieldValue(TciValue valueId, String
fieldName);
```

Parameters

`valueId`	Identifier of the value instance.
`fieldName`	Name of the record/set field

Description

This operation may be called to obtain the record/set field value by the record/set field name. If no field with such name exists in record/set then error is reported and NULL value is returned. It's allowed to use this function against undefined record/set fields. In this case omitted value will be created and returned.

Return Values

Returns the field value of specified record/set.

tciSetRecFieldValue

Sets the field value of specified record/set

```
void tciSetRecFieldValue(TciValue valueId, String
fieldName, TciValue fieldValueId);
```

Parameters

`valueId`	Identifier of the value instance.
`fieldName`	Name of the record/set field
`fieldValueId`	identifier or the value instance to be assigned to record/set field

Description

This operation may be called to set the record/set field value by the record/set field name. If no field with such name exists in record/set then error is reported. When assigning field value runtime system creates copy of the passed value (3rd parameter), thus it's possible to reuse passed field value in chain of assignments (e.g. in a loop).

Return Values

None.

tciSetFieldOmitted

Marks the referenced optional field in a record/set as being omitted.

```
void tciSetFieldOmitted(TciValue valueId, String
fieldName);
```

Parameters

`valueId`	Identifier of the value instance.
`fieldName`	Name of the record/set field

Description

This operation may be called to omit the optional field in a record or set. If no field with such name exists in record/set then error is reported. Calling this operation for a mandatory field also results in error.

Return Values

None.

tciGetRecFieldNames

Returns the NULL terminated array of record/set field names

```
String* tciGetRecFieldNames(TciValue valueId);
```

Parameters

`valueId`	Identifier of the record/set value instance.

Description

This operation may be called to obtain the array of record/set field names. The end of array is identified by NULL element. If record/set has no fields then NULL is returned.

Return Values

Returns the NULL terminated array of record/set field names.

Returns NULL if record/set has no fields.

RecordOf/SetOf Value Interface

tciGetRecOfFieldValue

Returns the element value of `record_of/set_of` at specified position

```
TciValue tciGetRecOfFieldValue(TciValue valueId, unsigned
long int position);
```

Parameters

valueId	Identifier of the record_of/set_of the value instance.
position	position of the element to return

Description

This operation may be called to obtain the value of record_of/set_of element at specified position. Valid position is between zero and length -1, for other positions the distinct value NULL is returned. recordOf and SetOf values will not be automatically expanded if position exceeds its lengths. It's allowed to use this function against undefined record_of/set_of elements. In this case an uninitialized value will be created and returned.

Return Values

Returns the element value of record_of/set_of at specified position.

tciSetRecOfFieldValue

Sets the element value of record_of/set_of at specified position

```
void tciSetRecOfFieldValue(TciValue vecValueId, unsigned
long int position, TciValue elemValueId);
```

Parameters

vecValueId	Identifier of the record_of/set_of the value instance.
position	position of the element to set
elemValueId	identifier of the value instance to be assigned to record_of/set_of element

Description

This operation may be called to set the record_of/set_of element value at specified position. If position is greater than (length -1) the record of is extended to have the length (position + 1). The record of elements between the original position at length and position - 1 are set to omit. When assigning

element value runtime system creates copy of the passed value (3rd parameter), thus it's possible to reuse passed element value in chain of assignments (e.g. in a loop).

Return Values

None.

tciAppendRecOfFieldValue

Appends specified value to the `record_of/set_of`

```
void tciAppendRecOfFieldValue(TciValue vecValueId, TciValue
elemValueId);
```

Parameters

`vecValueId`	Identifier of the `record_of/set_of` the value instance.
`elemValueId`	identifier of the value instance to be appended to `record_of/set_of`

Description

This operation may be called to append the element to the end of `record_of/set_of`, i.e. to set element value at position 'length'. When assigning element value runtime system creates copy of the passed value (2nd parameter), thus it's possible to reuse passed element value in chain of assignments (e.g. in a loop).

Return Values

None.

tciGetRecOfElementType

Returns the type identifier of the element of the specified `record_of/set_of`

```
TciType tciGetRecOfElementType(TciValue valueId);
```

Parameters

`valueId`	Identifier of the value instance.

Description

This operation may be called to lookup the element type of the `record_of/set_of`.

Return Values

Returns the type identifier of the elements of the specified `record_of/set_of`

tciGetRecOfLength

Returns the actual length of the specified `record_of/set_of` value

```
unsigned long int tciGetRecOfLength(TciValue valueId);
```

Parameters

`valueId`	Identifier of the value instance.

Description

This operation may be called to lookup the actual length of the `record_of/set_of` value.

Return Values

Returns the actual length of the specified `record_of/set_of` value

tciSetRecOfLength

Sets the length of the specified `record_of/set_of` value

```
void tciSetRecOfLength(TciValue valueId, unsigned long int
length);
```

Parameters

`valueId`	Identifier of the value instance.
`length`	New length to be set to the specified universal char-string value

Description

This operation may be called to change the length of the `record_of/set_of` value. If length is greater than the original length then `record_of/set_of` is expanded and newly created elements are set as undefined. If length is less than the original length then `record_of/set_of` is truncated to the specified length.

Return Values

Returns the actual length of the specified `record_of/set_of` value

Union/Anytype Value Interface

tciGetUnionVariant

Returns the variant value of specified union/anytype

```
TciValue tciGetUnionVariant(TciValue valueId, String
variantName);
```

Parameters

`valueId`	Identifier of the union/anytype value instance.
`variantName`	name of the union/anytype variant

Description

This operation may be called to obtain the union/anytype variant value that is denoted by the variant name. If no variant was previously set or `variantName` is not equal to the result of tciGetUnionPresentVariantName() then `variantName` is selected as present variant and fresh uninitialized value

is returned. The type of returned value corresponds to the union variant with name equal to `variantName`. If no variant with such name exists in union/anytype then error is reported and NULL value is returned.

Return Values

Returns the variant value of specified union/anytype denoted by `variantName`

tciSetUnionVariant

Sets the variant value of specified union/anytype and assigns specified value to it

```
void tciSetUnionVariant(TciValue valueId, String
variantName, TciValue variantValueId);
```

Parameters

`valueId`	Identifier of the union/anytype value instance.
`variantName`	name of the union/anytype variant
`variantValueId`	identifier of the value instance to be assigned to union/anytype variant

Description

This operation may be called to set the union/anytype variant and assign a value to it. Union/anytype variant is denoted by the specified variant name. If no variant with such name exists in union/anytype then error is reported and function returns without changing union/anytype state. When assigning variant value runtime system creates copy of the passed value (3rd parameter), thus it's possible to reuse passed variant value in chain of assignments (e.g. in a loop).

Return Values

None.

tciGetUnionPresentVariantName

Returns the name of the currently selected variant of specified union/anytype

```
String tciGetUnionPresentVariantName(TciValue valueId);
```

Parameters

`valueId`	Identifier of the union/anytype value instance.

Description

This operation may be called to lookup the name of currently selected union/anytype variant. If no variant was previously set then NULL is returned.

Return Values

Returns the name of currently selected union/anytype variant.

Returns NULL if no variant selected.

tciGetUnionVariantNames

Returns the NULL terminated array of union/anytype variant names

```
String* tciGetUnionVariantNames(TciValue inst);
```

Parameters

`valueId`	Identifier of the union/anytype value instance.

Description

This operation may be called to obtain the array of union/anytype variant names. The end of array is identified by NULL element. If union has no variants then NULL is returned. If the `valueId` represents the TTCN-3 anytype, i.e. the type class of the type obtained by tciGetType is ANYTYPE, then the array of all built-in and user-defined TTCN-3 type names is returned

Return Values

Returns the NULL terminated array of union/anytype variant names.

Returns NULL if union has no variants.

Enumerated Value Interface

tciGetEnumValue

Returns the string identifier of the specified enumerated value

```
String tciGetEnumValue(TciValue valueId);
```

Parameters

`valueId`	Identifier of the value instance.

Description

This operation may be called to obtain the string identifier of the enumerated value. This identifier equals the identifier in the TTCN-3 specification.

Return Values

Returns the string identifier of the specified enumerated value.

Returns NULL if enumerated value is not assigned with one of the enumeration choices.

tciSetEnumValue

Sets the enumerated value to a specified string identifier

```
void tciSetEnumValue(TciValue valueId, String enumValue);
```

Parameters

`valueId`	Identifier of the value instance.
`enumValue`	string identifier, one of enumerated choices

Description

This operation may be called to set the enumerated value to a specified string identifier. String identifier should be equal to the one of the possible enumerated choices. If `enumValue` is not an allowed value for this enumeration then the operation is ignored.

Return Values

None

Verdict Value Interface

tciGetVerdictValue

Returns verdict stored in the specified value

```
TciVerdictValue tciGetVerdictValue(TciValue valueId);
```

Parameters

`valueId`	Identifier of the value instance.

Description

This operation may be called to obtain the verdict from the specified value. Verdict is presented as an integer, which value denotes one of the possible TTCN-3 verdicts. Returned integer equals to the one of the following constants:

TCI_VERDICT_NONE, TCI_VERDICT_PASS,
TCI_VERDICT_INCONC, TCI_VERDICT_FAIL,
TCI_VERDICT_ERROR

Return Values

Returns the integer value that denotes one of the verdicts stored in the specified value

tciSetVerdictValue

Sets the verdict value to a specified verdict constant

```
void tciSetVerdictValue(TciValue valueId, TciVerdictValue
verdict);
```

Parameters

`valueId`	Identifier of the value instance.
`verdict`	integer value that denotes one of the TTCN-3 verdicts

Description

This operation may be called to set the verdict value to the one of possible TTCN-3 verdicts. Verdict is presented as an integer, which value should be equal to the one of the following constants:

TCI_VERDICT_NONE, TCI_VERDICT_PASS, TCI_VERDICT_INCONC, TCI_VERDICT_FAIL, TCI_VERDICT_ERROR

Return Values

None

Address Value Interface

tciGetAddressValue

Returns underlying value of the specified address value

```
TciValue tciGetAddressValue(TciValue valueId);
```

Parameters

`valueId`	Identifier of the value instance.

Description

This operation may be called to obtain the underlying value of the specified address value. Returned value is no longer of type class ADDRESS, but rather of the actual type used for 'address' type representation.

Return Values

Returns the underlying value of the specified address value. The type of the returned value is the type that is used in user-defined address type specifiication.

tciSetAddressValue

Sets the underlying value of the specified address value

```
void tciSetAddressValue(TciValue addrValueId, TciValue
undValueId);
```

Parameters

`addrValueId`	identifier of the address value instance
`undValueId`	identifier of the value instance to be assigned to address value

Description

This operation may be called to set the underlying address value to the specified value. The type of `undValueId` should be the type that is used in user-defined address type specification.

Return Values

None

TCI TE->CD Interface API

tciGetTypeForName

Lookups type identifier using specified type name.

```
TciType tciGetTypeForName(String typeName);
```

Parameters

`typeName`	name of type to look up

Description

This operation may be called to lookup type identifier using specified type name. Built-in TTCN-3 types can be retrieved from the TE by using the TTCN-3 keywords for the predefined types. In this case `typeName` denotes to the basic TTCN-3 type like `charstring`, `bitstring` etc. User-defined types as well as address and anytype types should be specified using fully qualified names.

Return Values

Returns the type identifier for the built-in and user-defined types.

Returns the distinct value null if the requested type can not be returned.

tciGetIntegerType

Lookups type identifier for the predefined type `integer`.

```
TciType tciGetIntegerType();
```

Parameters

None

Description

This operation may be called to lookup type identifier for the predefined type `integer`.

Return Values

Returns the type identifier for the predefined type `integer`.

tciGetFloatType

Lookups type identifier for the predefined type `float`.

```
TciType tciGetFloatType();
```

Parameters

None

Description

This operation may be called to lookup type identifier for the predefined type `float`.

Return Values

Returns the type identifier for the predefined type `float`.

tciGetBooleanType

Lookups type identifier for the predefined type `boolean`.

```
TciType tciGetBooleanType();
```

Parameters

None

Description

This operation may be called to lookup type identifier for the predefined type `boolean`.

Return Values

Returns the type identifier for the predefined type `boolean`.

tciGetCharType

Lookups type identifier for the predefined type `char`.

```
TciType tciGetCharType();
```

Parameters

None

Description

This operation may be called to lookup type identifier for the predefined type `char`.

Return Values

Returns the type identifier for the predefined type `char`.

tciGetUniversalCharType

Lookups type identifier for the predefined type `universal char`.

```
TciType tciGetUniversalCharType();
```

Parameters

None

Description

This operation may be called to lookup type identifier for the predefined type `universal char`.

Return Values

Returns the type identifier for the predefined type `universal char`.

tciGetObjidType

Lookups type identifier for the predefined type `objid`.

```
TciType tciGetObjidType();
```

Parameters

None

Description

This operation may be called to lookup type identifier for the predefined type `objid`.

Return Values

Returns the type identifier for the predefined type `objid`.

tciGetCharstringType

Lookups type identifier for the predefined type `charstring`.

```
TciType tciGetCharstringType();
```

Parameters

None

Description

This operation may be called to lookup type identifier for the predefined type `charstring`.

Return Values

Returns the type identifier for the predefined type `charstring`.

tciGetUniversalCharstringType

Lookups type identifier for the predefined type `universal charstring`.

```
TciType tciGetUniversalCharstringType();
```

Parameters

None

Description

This operation may be called to lookup type identifier for the predefined type `universal charstring`.

Return Values

Returns the type identifier for the predefined type `universal charstring`.

tciGetHexstringType

Lookups type identifier for the predefined type `hexstring`.

```
TciType tciGetHexstringType();
```

Parameters

None

Description

This operation may be called to lookup type identifier for the predefined type `hexstring`.

Return Values

Returns the type identifier for the predefined type `hexstring`.

tciGetBitstringType

Lookups type identifier for the predefined type `bitstring`.

```
TciType tciGetBitstringType();
```

Parameters

None

Description

This operation may be called to lookup type identifier for the predefined type `bitstring`.

Return Values

Returns the type identifier for the predefined type `bitstring`.

tciGetOctetstringType

Lookups type identifier for the predefined type `octetstring`.

```
TciType tciGetOctetstringType();
```

Parameters

None

Description

This operation may be called to lookup type identifier for the predefined type `octetstring`.

Return Values

Returns the type identifier for the predefined type `octetstring`.

tciGetVerdictType

Lookups type identifier for the predefined type `verdicttype`.

```
TciType tciGetVerdictType();
```

Parameters

None

Description

This operation may be called to lookup type identifier for the predefined type `verdicttype`.

Return Values

Returns the type identifier for the predefined type `verdicttype`..

tciErrorReq

Notifies the TE about non-recoverable error while encoding/decoding the data

```
void tciErrorReq(String message);
```

Parameters

`message`	character string containing description of error

Description

This operation may be called to notify TE about an unrecoverable error situation within the CD and forward the error indication to the test management

Return Values

None.

TCI CD->TE Interface API

tciDecode

Decodes received data.

```
TciValue tciDecode(BinaryString message, TciType
decodingHypothesis);
```

Parameters

`message`	encoded data
`decodingHypothesis`	Type identifier of expected type

Description

This operations decodes message according to the encoding rules and returns a TTCN-3 value. The `decodingHypothesis` shall be used to determine whether the encoded value can be decoded. If an encoding rule is not self-sufficient, i.e. if the encoded message does not inherently contain its type `decodingHypothesis` shall be used. If the encoded value can be decoded without the decoding hypothesis, the distinct NULL value shall be returned if the type determined from the encoded message is not compatible with the decoding hypothesis.

Return Values

Returns decoded value or NULL if decoding is not possible.

tciEncode

Encodes value to be sent.

```
BinaryString tciEncode(TciValue valueId);
```

Parameters

`valueId`	Value to be encoded

Description

This operations encodes value according to encoding rules.

Return Values

Returns binary string containing encoded representation of the specified value.

TCI TE->TM Interface API

tciRootModule

Selects specified module as root module.

```
void tciRootModule(String moduleId);
```

Parameters

`moduleId`	the name of module to be set as root

Description

This operation selects the indicated module for execution through a subsequent call using tciStartTestCase or tciStartControl. A `tciError` will be issued by the TE if no such module exists. This operation shall be used only if neither the control part nor a test case is currently being executed.

Return Values

None.

tciGetModules

Lookups the list of all modules defined in the testsuite.

```
TciModuleIdListType tciGetModules();
```

Parameters

None

Description

This operation returns the list of all modules defined in the testsuite.

The modules are ordered as they appear in the TTCN-3 module.

Return Values

Returns the list of module names.

tciGetImportedModules

Lookups the list of all modules imported by the root module.

```
TciModuleIdListType tciGetImportedModules();
```

Parameters

None

Description

This operation returns the list of imported modules of the root module.

The modules are ordered as they appear in the TTCN-3 module.

If no imported module exist, an empty module list is returned.

If the TE cannot provide a list, the distinct NULL value is returned.

This operation shall be used only if a root module has been set before.

Return Values

Returns the list of module names.

tciGetModuleParameters

Lookups the list of module parameters of a specified module.

```
TciModuleParameterListType
```

```
tciGetModuleParameters(TciModuleIdType moduleName);
```

Parameters

`moduleName`	module name for which to return module parameters

Description

This operation returns the list of module parameters of the identified module. The parameters are ordered as they appear in the TTCN-3 module. If no module parameters exist, an empty module parameter list is returned. If the TE cannot provide a list, the distinct NULL value is returned. This operation shall be used only if a root module has been set before.

Return Values

Returns the list of module parameters.

tciGetModuleParameterType

Returns the type identifier of a specified module parameter.

```
TciType tciGetModuleParameterType(TciModuleParameterIdType
modParId);
```

Parameters

`modParId`	fully qualified name of a module parameter

Description

This operation returns the type of the specified module parameter. This may be required if module parameter does not have default value. If the TE cannot provide type, the distinct NULL value is returned. This operation shall be used only if a root module has been set before.

Return Values

Returns the type identifier of a specified module parameter.

tciGetTestCases

Lookups the list of test cases either defined or imported in root module.

```
TciTestCaseIdListType T3TCI(tciGetTestCases) ();
```

Parameters

None

Description

This operation returns the list of test cases that are either defined in or imported into the root module. If no test cases exist, an empty test case list is returned. If the TE cannot provide a list, the distinct NULL value is returned. This operation shall be used only if a root module has been set before.

Return Values

Returns the list of test cases.

tciGetTestCaseParameters

Lookups the list of formal parameters types of a specified test case.

```
TciParameterTypeListType
tciGetTestCaseParameters(TciTestCaseIdType testCaseId);
```

Parameters

`testCaseId`	fully qualified test case name

Description

This operation returns the list of parameter types of the given test case. The parameter types are ordered as they appear in the TTCN-3 signature of the test case. If no test case parameters exist, an empty parameter type list is returned. If the TE cannot provide a list, the distinct NULL value is returned. This operation shall be used only if a root module has been set before.

Return Values

Returns the list of test case formal parameters types.

tciGetTestCaseParametersNames

Lookups the list of formal parameters names of a specified test case.

```
TciTestCaseParameterIdListType
tciGetTestCaseParametersNames(TciTestCaseIdType
testCaseId);
```

Parameters

`testCaseId`	fully qualified test case name

Description

This operation returns the list of parameter names of the given test case. The parameter names are ordered as they appear in the TTCN-3 signature of the test case. If no test case parameters exist, an empty parameter name list is returned. If the TE cannot provide a list, the distinct NULL value is returned. This operation shall be used only if a root module has been set before.

Return Values

Returns the list of test case formal parameters names.

tciGetTestCaseTSI

Returns the list of system ports of a specified test case.

```
TriPortIdList tciGetTestCaseTSI(TciTestCaseIdType
testCaseId);
```

Parameters

`testCaseId`	fully qualified test case name

Description

This operation returns the list of system ports of the given test case that have been declared in the definition of the system component for the test case, i.e. the TSI ports. If a system component has not been explicitly defined for the test case, then the list contains all communication ports of the MTC test com-

ponent. The ports are ordered as they appear in the respective TTCN-3 component type declaration. If no system ports exist, an empty port list is returned. If the TE cannot provide a list, the distinct NULL value is returned.

This operation shall be used only if a root module has been set before.

Note

This operation is not supported

Return Values

Returns the list of test case system ports.

tciStartTestCase

Starts test case with the specified actual parameters.

```
void tciStartTestCase(TciTestCaseIdType testCaseId,
TciParameterListType parameterList);
```

Parameters

`testCaseId`	fully qualified test case name
`parameterList`	A list of actual test case parameters

Description

This operation starts a test case in the currently selected module with the given parameters. A `tciError` will be issued by the TE if no such test case exists. All in and inout test case parameters in parameterList shall contain defined values. All out test case parameters in parameterList shall contain the distinct NULL value since they are only of relevance when the test case terminates. This operation shall be used only if a root module has been set before. It is only a `testCaseId` for a test case that is declared in the currently selected TTCN-3 module that shall pass. Test cases that are imported in a referenced module can not be started. To start imported test cases the referenced (imported) module must be selected first using the `tciRootModule` operation

Return Values

None.

tciStopTestCase

Stops currently running test case.

```
void tciStopTestCase();
```

Parameters

None

Description

This operation stops the test case currently being executed. If the TE is not executing a test case, the operation is ignored. If the control part is being executed, tciStopTestCase will stop execution of the currently executed test case, i.e. the execution of the test case that has recently been indicated using the provided operation tciTestCaseStarted. A possible executing control part will continue execution as if the test case has stopped normally and returned with verdict ERROR. This operation shall be used only if a root module has been set before.

Return Values

None.

tciStartControl

Starts control part of the selected module.

```
TriComponentId tciStartControl();
```

Parameters

None

Description

This operation starts the control part of the selected module. The control part starts TTCN-3 test cases as described in TTCN-3. While executing the control part the TE calls the provided operation tciTestCaseStarted and tciTestCaseTerminated for every test case that has been started and that has termi-

nated. After termination of the control part the TE calls the provided operation tciControlTerminated. This operation shall be used only if a root module has been set before.

Return Values

Returns the id of a component that executes started control part

tciStopControl

Stops currently executing control part.

```
void tciStopControl();
```

Parameters

None

Description

This operation stops execution of the control part. If no control part is currently being executed the operation is ignored. If a test case has been started directly this will stop execution of the current test case as if tciStopTestCase has been called. This operation shall be used only if a root module has been set before.

Return Values

None.

TCI TM->TE Interface API

tciTestCaseStarted

Notifies that test case has been started.

```
void tciTestCaseStarted(TciTestCaseIdType testCaseId,
TciParameterListType parameterList, double timeout);
```

Parameters

`testCaseId`	fully qualified test case name
`parameterList`	A list of actual test case parameters
`timeout`	double value of test case timeout

Description

This operation indicates to the test management that a test case with testCaseId has been started. It will not be distinguished whether the test case has been started explicitly using the required operation tciStartTestCase or implicitly while executing the control part. Zero value in timeout indicates that test case has been started without timeout.

Return Values

None.

tciTestCaseTerminated

Notifies that test case has been ended.

```
void tciTestCaseTerminated(TciValue verdict,
TciParameterListType parameterlist);
```

Parameters

`verdict`	final test case verdict
`parameterList`	list of test case parameters (inout and out have non NULL values)

Description

This operation indicates to the test management that a test case that has been currently executed on the MTC has terminated with specified final verdict.All out and inout test case parameters contain non NULL values. All in test case parameters contain the distinct NULL value.

Return Values

None.

tciControlTerminated

Notifies that control part has been ended.

```
void tciControlTerminated();
```

Parameters

None.

Description

This operation indicates to the test management that the control part of the selected module has just terminated execution.

Return Values

None.

tciGetModulePar

Returns the value of a specified module parameter.

```
TciValue tciGetModulePar(TciModuleParameterIdType
parameterId);
```

Parameters

`parameterId`	fully qualified name of a module parameter.

Description

The test management provides to the TE a value for the indicated module parameter. Every call of tciGetModulePar() should return the same value throughout the execution of an explicitly started test case or throughout the execution of a control part. If the management cannot provide a TTCN-3 value, the distinct NULL value should be returned.

Return Values

Returns the value of a specified module parameter.

Returns NULL if value cannot be determined.

tciError

Notifies about runtime error in TE.

```
void tciError(String message);
```

Parameters

`message`	description of a runtime error

Description

This operation indicates the occurrence of an unrecoverable error situation. message contains a reason phrase that might be communicated to the test system user. It is up to the test management to terminate execution of test cases or control parts if running. The test management has to take explicit measures to terminate test execution immediately.

Return Values

None.

Service Functions to TCI Interface

tciInit

Initializes TCI interface.

```
int tciInit(int argc, char *argv[]);
```

Parameters

`argc`	number of command line Parameters
`argv`	string array of command line Parameters

Description

This operation performs general initialization of the TCI interface.Command line parameters that have been passed to the main() function should be passed to tciInit() as well. tciInit() should be called outside of any TTCN-3 RTS functions. No TCI functions should be called prior to tciInit()

Return Values

Returns true on success, false otherwise.

tciMemoryAllocate

Allocates specified number of bytes in temporary memory area.

```
void *tciMemoryAllocate(unsigned long bytes);
```

Parameters

`bytes`	amount of memory in bytes to be allocated

Description

This function allocates memory block in temporary memory area. It is just a wrapper to t3rt_memory_temp_allocate() function that cannot be used inside TCI functions due to the lack of access to context.

Return Values

Returns pointer to newly allocated memory.

Returns NULL if memory cannot be allocated.

tciStartTestsuiteServer

Main function that is called when test suite execution is controlled using Rational Systems Tester GUI.

```
int tciStartTestsuiteServer(int argc, char *argv[]);
```

Parameters

argc	number of command line Parameters
argv	string array of command line Parameters

Description

This function initializes TCI interface and starts all internal servers that are necessary to control test suite execution from Rational Systems Tester GUI. This is a blocking function. It will return only when test suite (not certain test case or control part) will be terminated. tciStartTestsuiteServer() should be called outside of any TTCN-3 RTS functions. No TCI functions should be called prior to tciStartTestsuiteServer()

Return Values

Returns true on success, false otherwise.

TCI TL->TE Interface

tliTcExecute

Logs execute test case request.

```
void tliTcExecute(String am, long int ts, String src, long
int line, TriComponentId c, TciTestCaseIdType tcId,
TciParameterListType pars, TriTimerDuration dur);
```

Parameters

am	An additional message
ts	The time when the event is produced (in millisec-onds from process start)
src	The source file of the test specification
line	The line number where the request is performed
c	The component which produces this event

tcId	The test case to be executed
pars	The list of parameters required by the test case.
dur	Duration of the execution

Description

This operation is called by TE to log the execute test case request.

Return Values

None.

tliTcStart

Logs start of a test case.

```
void tliTcStart(String am, long int ts, String src, long
int line, TriComponentId c, TciTestCaseIdType tcId,
TciParameterListType pars, TriTimerDuration dur);
```

Parameters

am	An additional message
ts	The time when the event is produced (in milliseconds from process start)
src	The source file of the test specification
line	The line number where the request is performed
c	The component which produces this event
tcId	The test case to be executed
pars	The list of parameters required by the test case.
dur	Duration of the execution

Description

This operation is called by TE to log the start of a test case. This event occurs before the test case is started.

Return Values

None.

tliTcStop

Logs stop of a test case.

```
void tliTcStop(String am, long int ts, String src, long int
line, TriComponentId c);
```

Parameters

`am`	An additional message
`ts`	The time when the event is produced (in milliseconds from process start)
`src`	The source file of the test specification
`line`	The line number where the request is performed
`c`	The component which produces this event

Description

This operation is called by TE to log the stop of a test case.

Return Values

None.

tliTcStarted

Logs start of a test case.

```
void tliTcStarted(String am, long int ts, String src, long
int line, TriComponentId c, TciTestCaseIdType tcId,
TciParameterListType pars, TriTimerDuration dur);
```

Parameters

`am`	An additional message
`ts`	The time when the event is produced (in milliseconds from process start)
`src`	The source file of the test specification
`line`	The line number where the request is performed
`c`	The component which produces this event
`tcId`	The test case to be executed
`pars`	The list of parameters required by the test case.
`dur`	Duration of the execution

Description

This operation is called by TM to log the start of a test case. This event occurs after the test case was started.

Return Values

None.

tliTcTerminated

Logs termination of a test case.

```
void tliTcTerminated(String am, long int ts, String src,
long int line, TriComponentId c, TciTestCaseIdType tcId,
TciParameterListType pars, TciValue outcome);
```

Parameters

`am`	An additional message
`ts`	The time when the event is produced (in milliseconds from process start)
`src`	The source file of the test specification
`line`	The line number where the request is performed

c	The component which produces this event
tcId	The test case to be executed
pars	The list of parameters required by the test case.
outcome	The verdict of the test case

Description

This operation is called by TM to log the termination of a test case. This event occurs after the test case terminated.

Return Values

None.

tliCtrlStart

Logs start of the control part.

```
void tliCtrlStart(String am, long int ts, String src, long
int line, TriComponentId c);
```

Parameters

am	An additional message
ts	The time when the event is produced (in milliseconds from process start)
src	The source file of the test specification
line	The line number where the request is performed
c	The component which produces this event

Description

This operation is called by TE to log the start of the control part. This event occurs before the control is started. If the control is not represented by a TRI component, c is null.

Return Values

None.

tliCtrlStop

Logs stop of the control part.

```
void tliCtrlStop(String am, long int ts, String src, long
int line, TriComponentId c);
```

Parameters

`am`	An additional message
`ts`	The time when the event is produced (in milliseconds from process start)
`src`	The source file of the test specification
`line`	The line number where the request is performed
`c`	The component which produces this event

Description

This operation is called by TE to log the stop of the control part. This event occurs before the control is stopped. If the control is not represented by a TRI component, c is null.

Return Values

None.

tliCtrlTerminated

Logs termination of the control part.

```
void tliCtrlTerminated (String am, long int ts, String src,
long int line, TriComponentId c);
```

Parameters

`am`	An additional message
`ts`	The time when the event is produced (in milliseconds from process start)
`src`	The source file of the test specification
`line`	The line number where the request is performed
`c`	The component which produces this event

Description

This operation is called by TM to log the termination of the control part. This event occurs after the control has terminated. If the control is not represented by a TRI component, c is null.

Return Values

None.

tliMSend_m

Logs unicast (point-to-point communication) send operation.

```
void tliMSend_m(String am, long int ts, String src, long
int line, TriComponentId c, TriPortId atPort, TriPortId
toPort, TciValue msgValue, TriAddress address, TriStatus
encoderFailure, TriMessage msg, TriStatus
transmissionFailure);
```

Parameters

`am`	An additional message
`ts`	The time when the event is produced (in milliseconds from process start)
`src`	The source file of the test specification
`line`	The line number where the request is performed
`c`	The component which produces this event

atPort	The port via which the message is sent
toPort	The port to which the message is sent
msgValue	The value to be encoded and sent
address	The address of the destination within the SUT
encoderFailure	The failure message which might occur at encoding
msg	The encoded message
transmissionFailure	The failure message which might occur at transmission

Description

This operation is called by SA to log a unicast send operation. This event occurs after sending. This event is used for logging the communication with the SUT.

Return Values

None.

tliMSend_m_BC

Logs broadcast send operation.

```
void tliMSend_m_BC(String am, long int ts, String src, long
int line, TriComponentId c, TriPortId atPort, TriPortId
toPort, TciValue msgValue, TriStatus encoderFailure,
TriMessage msg, TriStatus transmissionFailure);
```

Parameters

am	An additional message
ts	The time when the event is produced (in milliseconds from process start)
src	The source file of the test specification

line	The line number where the request is performed
c	The component which produces this event
atPort	The port via which the message is sent
toPort	The port to which the message is sent
msgValue	The value to be encoded and sent
address	The address of the destination within the SUT
encoderFailure	The failure message which might occur at encoding
msg	The encoded message
transmissionFailure	The failure message which might occur at transmission

Description

This operation is called by SA to log a broadcast send operation. This event occurs after sending. This event is used for logging the communication with the SUT.

Return Values

None.

tliMSend_m_MC

Logs multicast send operation.

```
void tliMSend_m_MC(String am, long int ts, String src, long
int line, TriComponentId c, TriPortId atPort, TriPortId
toPort, TciValue msgValue, TriAddressList addresses,
TriStatus encoderFailure, TriMessage msg, TriStatus
transmissionFailure);
```

Parameters

`am`	An additional message
`ts`	The time when the event is produced (in milliseconds from process start)
`src`	The source file of the test specification
`line`	The line number where the request is performed
`c`	The component which produces this event
`atPort`	The port via which the message is sent
`toPort`	The port to which the message is sent
`msgValue`	The value to be encoded and sent
`addresses`	The addresses of the destination within the SUT
`encoderFailure`	The failure message which might occur at encoding
`msg`	The encoded message
`transmissionFailure`	The failure message which might occur at transmission

Description

This operation is called by SA to log a multicast send operation. This event occurs after sending. This event is used for logging the communication with the SUT.

Return Values

None.

tliMSend_c

Logs unicast send operation.

```
void tliMSend_c(String am, long int ts, String src, long
int line, TriComponentId c, TriPortId atPort, TriPortId
toPort, TciValue msgValue, TriStatus transmissionFailure);
```

Parameters

`am`	An additional message
`ts`	The time when the event is produced (in milliseconds from process start)
`src`	The source file of the test specification
`line`	The line number where the request is performed
`c`	The component which produces this event
`atPort`	The port via which the message is sent
`toPort`	The port to which the message is sent
`msgValue`	The value to be encoded and sent
`transmissionFailure`	The failure message which might occur at transmission

Description

This operation is called by CH to log a unicast send operation. This event occurs after sending. This event is used for logging the inter-component communication.

Return Values

None.

tliMSend_c_BC

Logs broadcast send operation.

```
void tliMSend_c_BC(String am, long int ts, String src, long
int line, TriComponentId c, TriPortId atPort, TriPortIdList
toPortList, TciValue msgValue, TriStatus
transmissionFailure);
```

Parameters

`am`	An additional message
`ts`	The time when the event is produced (in milliseconds from process start)
`src`	The source file of the test specification
`line`	The line number where the request is performed
`c`	The component which produces this event
`atPort`	The port via which the message is sent
`toPortList`	The ports to which the message is sent
`msgValue`	The value to be encoded and sent
`transmissionFailure`	The failure message which might occur at transmission

Description

This operation is called by CH to log a broadcast send operation. This event occurs after sending. This event is used for logging the inter-component communication.

Return Values

None.

tliMSend_c_MC

Logs multicast send operation.

```
void tliMSend_c_MC(String am, long int ts, String src, long
int line, TriComponentId c, TriPortId atPort, TriPortIdList
toPortList, TciValue msgValue, TriStatus
transmissionFailure);
```

Parameters

`am`	An additional message
`ts`	The time when the event is produced (in milliseconds from process start)
`src`	The source file of the test specification
`line`	The line number where the request is performed
`c`	The component which produces this event
`atPort`	The port via which the message is sent
`toPortList`	The ports to which the message is sent
`msgValue`	The value to be encoded and sent
`transmissionFailure`	The failure message which might occur at transmission

Description

This operation is called by CH to log a multicast send operation. This event occurs after sending. This event is used for logging the inter-component communication.

Return Values

None.

tliMDetected_m

Logs enqueuing of a message.

```
void tliMDetected_m(String am, long int ts, String src,
long int line, TriComponentId c, TriPortId atPort,
TriPortId fromPort, TriMessage msg, TriAddress address);
```

Parameters

`am`	An additional message
`ts`	The time when the event is produced (in milliseconds from process start)
`src`	The source file of the test specification
`line`	The line number where the request is performed
`c`	The component which produces this event
`atPort`	The port at which the message is detected
`fromPort`	The port via which the message is sent
`msg`	The encoded value enqueued into port
`address`	The address of the source within the SUT

Description

This operation is called by SA to log the enqueuing of a message. This event occurs after the message is enqueued. This event is used for logging the communication with the SUT.

Return Values

None.

tliMDetected_c

Logs enqueuing of a message.

```
void tliMDetected_c(String am, long int ts, String src,
long int line, TriComponentId c, TriPortId atPort,
TriPortId fromPort, TciValue msgValue);
```

Parameters

`am`	An additional message
`ts`	The time when the event is produced (in milliseconds from process start)
`src`	The source file of the test specification
`line`	The line number where the request is performed
`c`	The component which produces this event
`atPort`	The port at which the message is detected
`fromPort`	The value enqueued into port
`msgValue`	The value to be encoded and sent

Description

This operation is called by CH to log the enqueuing of a message. This event occurs after the message is enqueued. This event is used for logging the inter-component communication.

Return Values

None.

tliMMismatch_m

Logs mismatch of a template.

```
void tliMMismatch_m(String am, long int ts, String src,
long int line, TriComponentId c, TriPortId port, TciValue
msgValue, TciValueTemplate msgTmpl, TciValueDifferenceList
diffs, TriAddress address, TciValueTemplate addressTmpl);
```

Parameters

`am`	An additional message
`ts`	The time when the event is produced (in milliseconds from process start)
`src`	The source file of the test specification

line	The line number where the request is performed
c	The component which produces this event
port	The port via which the message is received
msgValue	The message which is checked against the template
msgTmpl	The template used to check the message match
diffs	The difference/the mismatch between message and template
address	The address of the source within the SUT
addressTmpl	The expected address of the source within the SUT

Description

This operation is called by TE to log the mismatch of a template. This event occurs after checking a template match. This event is used for logging the communication with the SUT.

Return Values

None.

tliMMismatch_c

Logs mismatch of a template.

```
void tliMMismatch_c(String am, long int ts, String src,
long int line, TriComponentId c, TriPortId port, TciValue
msgValue, TciValueTemplate msgTmpl, TciValueDifferenceList
diffs, TriComponentId from, TciNonValueTemplate fromTmpl);
```

Parameters

`am`	An additional message
`ts`	The time when the event is produced (in milliseconds from process start)
`src`	The source file of the test specification
`line`	The line number where the request is performed
`c`	The component which produces this event
`port`	The port via which the message is received
`msgValue`	The message which is checked against the template
`msgTmpl`	The template used to check the message match
`diffs`	The difference/the mismatch between message and template
`from`	The component which sent the message
`fromTmpl`	The expected sender component

Description

This operation is called by TE to log the mismatch of a template. This event occurs after checking a template match. This event is used for logging the inter-component communication.

Return Values

None.

tliMReceive_m

Logs receive of a message.

```
void tliMReceive_m(String am, long int ts, String src, long
int line, TriComponentId c, TriPortId port, TciValue
msgValue, TciValueTemplate msgTmpl, TriAddress address,
TciValueTemplate addressTmpl);
```

Parameters

`am`	An additional message
`ts`	The time when the event is produced (in milliseconds from process start)
`src`	The source file of the test specification
`line`	The line number where the request is performed
`c`	The component which produces this event
`port`	The port via which the message is received
`msgValue`	The message which is checked against the template
`msgTmpl`	The template used to check the message match
`address`	The address of the source within the SUT
`addressTmpl`	The expected address of the source within the SUT

Description

This operation is called by TE to log the receive of a message. This event occurs after checking a template match. This event is used for logging the communication with SUT.

Return Values

None.

tliMReceive_c

Logs receive of a message.

```
void tliMReceive_c(String am, long int ts, String src, long
int line, TriComponentId c, TriPortId port, TciValue
msgValue, TciValueTemplate msgTmpl, TriComponentId from,
TciNonValueTemplate fromTmpl);
```

Parameters

`am`	An additional message
`ts`	The time when the event is produced (in milliseconds from process start)
`src`	The source file of the test specification
`line`	The line number where the request is performed
`c`	The component which produces this event
`port`	The port via which the message is received
`msgValue`	The message which is checked against the template
`msgTmpl`	The template used to check the message match
`from`	The component which sent the message
`fromTmpl`	The expected sender component

Description

This operation is called by TE to log the receive of a message. This event occurs after checking a template match. This event is used for logging the inter-component communication.

Return Values

None.

tliPrCall_m

Logs unicast call operation.

```
void tliPrCall_m(String am, long int ts, String src, long
int line, TriComponentId c, TriPortId atPort, TriPortId
toPort, TriSignatureId signature, TciParameterListType
parsValue, TriAddress address, TriStatus encoderFailure,
TriParameterList pars, TriStatus transmissionFailure);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.
atPort	The port via which the call is invoked.
toPort	The port for which the call is invoked.
signature	The signature of the called operation.
parsValue	The parameters of the called operation.
address	The address of the destination within the SUT.
encoderFailure	The failure message which might occur at encoding.
pars	The encoded parameters.
transmissionFailure	The failure message which might occur at transmission.

Description

This operation is called by SA to log a unicast call operation. This event occurs after call execution. This event is used for logging the communication with the SUT.

Return Values

None.

tliPrCall_m_BC

Logs broadcast call operation.

```
void tliPrCall_m_BC(String am, long int ts, String src,
long int line, TriComponentId c, TriPortId atPort,
TriPortId toPort, TriSignatureId signature,
TciParameterListType parsValue, TriStatus encoderFailure,
TriParameterList pars, TriStatus transmissionFailure);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.
`atPort`	The port via which the call is invoked.
`toPort`	The port for which the call is invoked.
`signature`	The signature of the called operation.
`parsValue`	The parameters of the called operation.
`encoderFailure`	The failure message which might occur at encoding.
`pars`	The encoded parameters.
`transmissionFailure`	The failure message which might occur at transmission.

Description

This operation is called by SA to log a broadcast call operation. This event occurs after call execution. This event is used for logging the communication with the SUT.

Return Values

None.

tliPrCall_m_MC

Logs multicast call operation.

```
void tliPrCall_m_MC(String am, long int ts, String src,
long int line, TriComponentId c, TriPortId atPort,
TriPortId toPort, TriSignatureId signature,
TciParameterListType parsValue, TriAddressList addresses,
TriStatus encoderFailure, TriParameterList pars, TriStatus
```

```
transmissionFailure);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.
`atPort`	The port via which the call is invoked.
`toPort`	The port for which the call is invoked.
`signature`	The signature of the called operation.
`parsValue`	The parameters of the called operation.
`addresses`	The addresses of the destinations within the SUT.
`encoderFailure`	The failure message which might occur at encoding.
`pars`	The encoded parameters.
`transmissionFailure`	The failure message which might occur at transmission.

Description

This operation is called by SA to log a multicast call operation. This event occurs after call execution. This event is used for logging the communication with the SUT.

Return Values

None.

tliPrCall_c

Logs unicast call operation.

```
void tliPrCall_c(String am, long int ts, String src, long
```

```
int line, TriComponentId c, TriPortId atPort, TriPortId
toPort, TriSignatureId signature, TciParameterListType
parsValue, TriStatus transmissionFailure);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.
`atPort`	The port via which the call is invoked.
`toPort`	The port for which the call is invoked.
`signature`	The signature of the called operation.
`parsValue`	The parameters of the called operation.
`transmissionFailure`	The failure message which might occur at transmission.

Description

This operation is called by CH to log a unicast call operation. This event occurs after call execution. This event is used for logging the inter-component communication.

Return Values

None.

tliPrCall_c_BC

Logs broadcast call operation.

```
void tliPrCall_c_BC(String am, long int ts, String src,
long int line, TriComponentId c, TriPortId atPort,
TriPortIdList toPortList, TriSignatureId signature,
TciParameterListType parsValue, TriStatus
transmissionFailure);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.
`atPort`	The port via which the call is invoked.
`toPortList`	List of ports for which the call is invoked.
`signature`	The signature of the called operation.
`parsValue`	The parameters of the called operation.
`transmissionFailure`	The failure message which might occur at transmission.

Description

This operation is called by CH to log a broadcast call operation. This event occurs after call execution. This event is used for logging the inter-component communication.

Return Values

None.

tliPrCall_c_MC

Logs multicast call operation.

```
void tliPrCall_c_MC(String am, long int ts, String src,
long int line, TriComponentId c, TriPortId atPort,
TriPortIdList toPortList, TriSignatureId signature,
TciParameterListType parsValue, TriStatus
transmissionFailure);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.
atPort	The port via which the call is invoked.
toPortList	List of ports for which the call is invoked.
signature	The signature of the called operation.
parsValue	The parameters of the called operation.
transmissionFailure	The failure message which might occur at transmission.

Description

This operation is called by CH to log a multicast call operation. This event occurs after call execution. This event is used for logging the inter-component communication.

Return Values

None.

tliPrGetCallDetected_m

Logs `getcall` enqueue operation.

```
void tliPrGetCallDetected_m(String am, long int ts, String
src, long int line, TriComponentId c, TriPortId atPort,
TriPortId fromPort, TriSignatureId signature,
TriParameterList pars, TriAddress address);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.
atPort	The port via which the call is received.
fromPort	The port via which the call is sent.
signature	The signature of the detected call.
pars	The encoded parameters of detected call.
address	The address of the destination within the SUT.

Description

This operation is called by SA to log the `getcall` enqueue operation. This event occurs after call is enqueued. This event is used for logging the communication with the SUT.

Return Values

None.

tliPrGetCallDetected_c

Logs `getcall` enqueue operation.

```
void tliPrGetCallDetected_c(String am, long int ts, String
src, long int line, TriComponentId c, TriPortId atPort,
TriPortId fromPort, TriSignatureId signature,
TciParameterListType parsValue);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.
atPort	The port via which the call is received.
fromPort	The port via which the call is sent.
signature	The signature of the called operation.
parsValue	The parameters of detected call.

Description

This operation is called by CH to log the `getcall` enqueue operation. This event occurs after call is enqueued. This event is used for logging the inter-component communication.

Return Values

None.

tliPrGetCallMismatch_m

Logs mismatch of a `getcall`.

```
void tliPrGetCallMismatch_m(String am, long int ts, String
src, long int line, TriComponentId c, TriPortId port,
TriSignatureId signature, TciParameterListType parsValue,
TciValueTemplate parsTmpl, TciValueDifferenceList diff,
TriAddress address, TciValueTemplate addressTmpl);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.

line	The line number where the request is performed.
c	The component which produces this event.
port	The port via which the call is received.
signature	The signature of the detected call.
parsValue	The parameters of detected call.
parsTmpl	The template used to check the parameter match.
diffs	The difference/the mismatch between call and template
address	The address of the source within the SUT.
addressTmpl	The expected address of the source within the SUT.

Description

This operation is called by TE to log the mismatch of a `getcall`. This event occurs after `getcall` is checked against a template. This event is used for logging the communication with the SUT.

Return Values

None.

tliPrGetCallMismatch_c

Logs mismatch of a `getcall`.

```
void tliPrGetCallMismatch_c(String am, long int ts, String
src, long int line, TriComponentId c, TriPortId port,
TriSignatureId signature, TciParameterListType parsValue,
TciValueTemplate parsTmpl, TciValueDifferenceList diffs,
TriComponentId from, TciNonValueTemplate fromTmpl);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.

line	The line number where the request is performed.
c	The component which produces this event.
port	The port via which the call is received.
signature	The signature of the detected call.
parsValue	The parameters of detected call.
parsTmpl	The template used to check the parameter match.
diffs	The difference/the mismatch between message and template
from	The component which called the operation.
fromTmpl	The expected calling component.

Description

This operation is called by TE to log the mismatch of a `getcall`. This event occurs after `getcall` is checked against a template. This event is used for logging the inter-component communication.

Return Values

None.

tliPrGetCall_m

Logs getting a call.

```
void tliPrGetCall_m(String am, long int ts, String src,
long int line, TriComponentId c, TriPortId port,
TriSignatureId signature, TciParameterListType parsValue,
TciValueTemplate parsTmpl, TriAddress address,
TciValueTemplate addressTmpl);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.

line	The line number where the request is performed.
c	The component which produces this event.
port	The port via which the call is received.
signature	The signature of the detected call.
parsValue	The parameters of detected call.
parsTmpl	The template used to check the parameter match.
address	The address of the source within the SUT.
addressTmpl	The expected address of the source within the SUT.

Description

This operation is called by TE to log getting a call. This event occurs after `getcall` has matched against a template. This event is used for logging the communication with the SUT.

Return Values

None.

tliPrGetCall_c

Logs getting a call.

```
void tliPrGetCall_c(String am, long int ts, String src,
long int line, TriComponentId c, TriPortId port,
TriSignatureId signature, TciParameterListType parsValue,
TciValueTemplate parsTmpl, TriComponentId from,
TciNonValueTemplate fromTmpl);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.

port	The port via which the call is received.
signature	The signature of the detected call.
parsValue	The parameters of detected call.
parsTmpl	The template used to check the parameter match.
from	The component which called the operation.
fromTmpl	The expected calling component.

Description

This operation is called by TE to log getting a call. This event occurs after `getcall` has matched against a template. This event is used for logging the inter-component communication.

Return Values

None.

tliPrReply_m

Logs unicast reply operation.

```
void tliPrReply_m(String am, long int ts, String src, long
int line, TriComponentId c, TriPortId atPort, TriPortId
toPort, TriSignatureId signature, TciValue parsValue,
TciValue replValue, TriAddress address, TriStatus
encoderFailure, TriParameterList pars, TriParameter repl,
TriStatus transmissionFailure);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.
atPort	The port via which the reply is sent.

toPort	The port for which the reply is sent.
signature	The signature relating to the reply.
parsValue	The signature parameters relating to the reply.
replValue	The reply to be sent.
address	The address of the destination within the SUT.
encoderFailure	The failure message which might occur at encoding.
pars	The encoded signature parameters relating to the reply.
repl	The encoded reply.
transmissionFailure	The failure message which might occur at transmission.

Description

This operation is called by SA to log a unicast reply operation. This event occurs after reply execution. This event is used for logging the communication with the SUT

Return Values

None.

tliPrReply_m_BC

Logs broadcast reply operation.

```
void tliPrReply_m_BC(String am, long int ts, String src,
long int line, TriComponentId c, TriPortId atPort,
TriPortId toPort, TriSignatureId signature, TciValue
parsValue, TciValue replValue, TriStatus encoderFailure,
TriParameterList pars, TriParameter repl, TriStatus
transmissionFailure);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.
atPort	The port via which the reply is sent.
toPort	The port for which the reply is sent.
signature	The signature relating to the reply.
parsValue	The signature parameters relating to the reply.
replValue	The reply to be sent.
encoderFailure	The failure message which might occur at encoding.
pars	The encoded signature parameters relating to the reply.
repl	The encoded reply.
transmissionFailure	The failure message which might occur at transmission.

Description

This operation is called by SA to log a broadcast reply operation. This event occurs after reply execution. This event is used for logging the communication with the SUT

Return Values

None.

tliPrReply_m_MC

Logs multicast reply operation.

```
void tliPrReply_m_MC(String am, long int ts, String src,
long int line, TriComponentId c, TriPortId atPort,
```

```
TriPortId toPort, TriSignatureId signature, TciValue
parsValue, TciValue replValue, TriAddressList addresses,
TriStatus encoderFailure, TriParameterList pars,
TriParameter repl, TriStatus transmissionFailure);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.
`atPort`	The port via which the reply is sent.
`toPort`	The port for which the reply is sent.
`signature`	The signature relating to the reply.
`parsValue`	The signature parameters relating to the reply.
`replValue`	The reply to be sent.
`addresses`	The addresses of the destinations within the SUT.
`encoderFailure`	The failure message which might occur at encoding.
`pars`	The encoded signature parameters relating to the reply.
`repl`	The encoded reply.
`transmissionFailure`	The failure message which might occur at transmission.

Description

This operation is called by SA to log a multicast reply operation. This event occurs after reply execution. This event is used for logging the communication with the SUT

Return Values

None.

tliPrReply_c

Logs unicast reply operation.

```
void tliPrReply_c(String am, long int ts, String src, long
int line, TriComponentId c, TriPortId atPort, TriPortId
toPort, TriSignatureId signature, TciValue parsValue,
TciValue replValue, TriStatus transmissionFailure);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.
`atPort`	The port via which the reply is sent.
`toPort`	The port for which the reply is sent.
`signature`	The signature relating to the reply.
`parsValue`	The signature parameters relating to the reply.
`replValue`	The reply to be sent.
`transmissionFailure`	The failure message which might occur at transmission.

Description

This operation is called by CH to log a unicast reply operation. This event occurs after reply execution. This event is used for logging the inter-component communication.

Return Values

None.

tliPrReply_c_BC

Logs broadcast reply operation.

```
void tliPrReply_c_BC(String am, long int ts, String src,
long int line, TriComponentId c, TriPortId atPort,
TriPortIdList toPortList, TriSignatureId signature,
TciValue parsValue, TciValue replValue, TriStatus
transmissionFailure);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.
`atPort`	The port via which the reply is sent.
`toPortList`	List of ports to which the reply is sent.
`signature`	The signature relating to the reply.
`parsValue`	The signature parameters relating to the reply.
`replValue`	The reply to be sent.
`transmissionFailure`	The failure message which might occur at transmission.

Description

This operation is called by CH to log a broadcast reply operation. This event occurs after reply execution. This event is used for logging the inter-component communication.

Return Values

None.

tliPrReply_c_MC

Logs multicast reply operation.

```
void tliPrReply_c_MC(String am, long int ts, String src,
long int line, TriComponentId c, TriPortId atPort,
TriPortIdList toPortList, TriSignatureId signature,
TciValue parsValue, TciValue replValue, TriStatus
transmissionFailure);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.
`atPort`	The port via which the reply is sent.
`toPortList`	List of ports to which the reply is sent.
`signature`	The signature relating to the reply.
`parsValue`	The signature parameters relating to the reply.
`replValue`	The reply to be sent.
`transmissionFailure`	The failure message which might occur at transmission.

Description

This operation is called by CH to log a multicast reply operation. This event occurs after reply execution. This event is used for logging the inter-component communication.

Return Values

None.

tliPrGetReplyDetected_m

Logs `getreply` enqueue operation.

```
void tliPrGetReplyDetected_m(String am, long int ts, String
src, long int line, TriComponentId c, TriPortId port,
TriSignatureId signature, TriParameterList pars,
TriParameter repl, TriAddress address);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.
`atPort`	The port to which the reply is enqueued.
`fromPort`	The port from which the reply is sent.
`signature`	The signature relating to the reply.
`pars`	The encoded signature parameters relating to the reply.
`repl`	The received encoded reply.
`address`	The address of the source within the SUT.

Description

This operation is called by SA to log the `getreply` enqueue operation. This event occurs after `getreply` is enqueued. This event is used for logging the communication with the SUT.

Note

`getreply` *is a TTCN-3 port operation. When a reply to previously made procedure call is received from a communication channel it is added to the port queue. Later the runtime system will extract it from the port queue (in a first-in-first-out order), then generate other events like "reply matched template". The function tliPrGetReplyDetected_m() is intended to log the receive of a reply and its addition to the port queue.*

Return Values

None.

tliPrGetReplyDetected_c

Logs `getreply` enqueue operation.

```
void tliPrGetReplyDetected_c(String am, long int ts, String
src, long int line, TriComponentId c, TriPortId atPort,
TriPortId fromPort, TriSignatureId signature,
TciParameterListType parsValue, TciValue replValue);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.
`atPort`	The port to which the reply is enqueued.
`fromPort`	The port from which the reply is sent.
`signature`	The signature relating to the reply.
`parsValue`	The signature parameters relating to the reply.
`replValue`	The received reply.

Description

This operation is called by CH to log the `getreply` enqueue operation. This event occurs after `getreply` is enqueued. This event is used for logging the inter-component communication.

Return Values

None.

tliPrGetReplyMismatch_m

Logs mismatch of a `getreply` operation.

```
void tliPrGetReplyMismatch_m(String am, long int ts, String
src, long int line, TriComponentId c, TriPortId port,
TriSignatureId signature, TciParameterListType parsValue,
TciValueTemplate parsTmpl, TciValue replValue,
TciValueTemplate replyTmpl, TciValueDifferenceList diffs,
TriAddress address, TciValueTemplate addressTmpl);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.
`port`	The port via which the reply is received.
`signature`	The signature relating to the reply.
`parsValue`	The signature parameters relating to the reply.
`parsTmpl`	The signature template relating to the reply.
`replValue`	The received reply.
`replyTmpl`	The template used to check the reply match.
`diffs`	The difference/the mismatch between reply and template
`address`	The address of the source within the SUT.
`addressTmpl`	The expected address of the source within the SUT.

Description

This operation is called by TE to log the mismatch of a `getreply` operation. This event occurs after `getreply` is checked against a template. This event is used for logging the communication with SUT.

Return Values

None.

tliPrGetReplyMismatch_c

Logs mismatch of a `getreply` operation.

```
void tliPrGetReplyMismatch_c(String am, long int ts, String
src, long int line, TriComponentId c, TriPortId port,
TriSignatureId signature, TciParameterListType parsValue,
TciValueTemplate parsTmpl, TciValue replValue,
TciValueTemplate replyTmpl, TciValueDifferenceList diffs,
TriComponentId from, TciNonValueTemplate fromTmpl);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.
`port`	The port via which the reply is received.
`signature`	The signature relating to the reply.
`parsValue`	The signature parameters relating to the reply.
`parsTmpl`	The signature template relating to the reply.
`repl`	The received reply.
`replyTmpl`	The template used to check the reply match.
`diffs`	The difference/the mismatch between reply and template
`from`	The component which sent the reply.
`fromTmpl`	The expected replying component.

Description

This operation is called by TE to log the mismatch of a `getreply` operation. This event occurs after `getreply` is checked against a template. This event is used for logging the inter-component communication.

Return Values

None.

tliPrGetReply_m

Logs getting a reply.

```
void tliPrGetReply_m(String am, long int ts, String src,
long int line, TriComponentId c, TriPortId port,
TriSignatureId signature, TciParameterListType parsValue,
TciValueTemplate parsTmpl, TciValue replValue,
TciValueTemplate replyTmpl, TriAddress address,
TciValueTemplate addressTmpl);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.
`port`	The port via which the reply is received.
`signature`	The signature relating to the reply.
`parsValue`	The signature parameters relating to the reply.
`parsTmpl`	The signature template relating to the reply.
`replValue`	The received reply.
`replyTmpl`	The template used to check the reply match.
`address`	The address of the source within the SUT.
`addressTmpl`	The expected address of the source within the SUT.

Description

This operation is called by TE to log getting a reply. This event occurs after `getreply` is checked against a template. This event is used for logging the communication with SUT.

Return Values

None.

tliPrGetReply_c

Logs getting a reply.

```
void tliPrGetReply_c(String am, long int ts, String src,
long int line, TriComponentId c, TriPortId port,
TriSignatureId signature, TciParameterListType parsValue,
TciValueTemplate parsTmpl, TciValue replValue,
TciValueTemplate replyTmpl, TriComponentId from,
TciNonValueTemplate fromTmpl);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.
`port`	The port via which the reply is received.
`signature`	The signature relating to the reply.
`parsValue`	The signature parameters relating to the reply.
`parsTmpl`	The signature template relating to the reply.
`replValue`	The received reply.
`replyTmpl`	The template used to check the reply match.
`from`	The component which sent the reply.
`fromTmpl`	The expected replying component.

Description

This operation is called by TE to log getting a reply. This event occurs after `getreply` is checked against a template. This event is used for logging the inter-component communication.

Return Values

None.

tliPrRaise_m

Logs unicast raise operation.

```
void tliPrRaise_m(String am, long int ts, String src, long
int line, TriComponentId c, TriPortId atPort, TriPortId
toPort, TriSignatureId signature, TciParameterListType
parsValue, TciValue excValue, TriAddress address, TriStatus
encoderFailure, TriException exc, TriStatus
transmissionFailure);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.
`atPort`	The port via which the exception is sent.
`toPort`	The port to which the exception is sent.
`signature`	The signature relating to the exception.
`parsValue`	The signature parameters relating to the exception.
`excValue`	The exception to be sent.
`address`	The address of the destination within the SUT.

encoderFailure	The failure message which might occur at encoding.
exc	The encoded exception.
transmissionFailure	The failure message which might occur at transmission.

Description

This operation is called by SA to log a unicast raise operation. This event occurs after reply execution. This event is used for logging the communication with the SUT.

Return Values

None.

tliPrRaise_m_BC

Logs broadcast raise operation.

```
void tliPrRaise_m_BC(String am, long int ts, String src,
long int line, TriComponentId c, TriPortId atPort,
TriPortId toPort, TriSignatureId signature,
TciParameterListType parsValue, TciValue excValue,
TriStatus encoderFailure, TriException exc, TriStatus
transmissionFailure);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.
atPort	The port via which the exception is sent.
toPort	The port to which the exception is sent.
signature	The signature relating to the exception.

parsValue	The signature parameters relating to the exception.
excValue	The exception to be sent.
encoderFailure	The failure message which might occur at encoding.
exc	The encoded exception.
transmissionFailure	The failure message which might occur at transmission.

Description

This operation is called by SA to log a broadcast raise operation. This event occurs after reply execution. This event is used for logging the communication with the SUT.

Return Values

None.

tliPrRaise_m_MC

Logs multicast raise operation.

```
void tliPrRaise_m_MC(String am, long int ts, String src,
long int line, TriComponentId c, TriPortId atPort,
TriPortId toPort, TriSignatureId signature,
TciParameterListType parsValue, TciValue excValue,
TriAddressList addresses, TriStatus encoderFailure,
TriException exc, TriStatus transmissionFailure);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.
atPort	The port via which the exception is sent.

toPort	The port to which the exception is sent.
signature	The signature relating to the exception.
parsValue	The signature parameters relating to the exception.
excValue	The exception to be sent.
addresses	The addresses of the destinations within the SUT.
encoderFailure	The failure message which might occur at encoding.
exc	The encoded exception.
transmissionFailure	The failure message which might occur at transmission.

Description

This operation is called by SA to log a multicast raise operation. This event occurs after reply execution. This event is used for logging the communication with the SUT.

Return Values

None.

tliPrRaise_c

Logs unicast raise operation.

```
void tliPrRaise_c(String am, long int ts, String src, long
int line, TriComponentId c, TriPortId atPort, TriPortId
toPort, TriSignatureId signature, TciParameterListType
parsValue, TciValue excValue, TriStatus
transmissionFailure);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.

line	The line number where the request is performed.
c	The component which produces this event.
atPort	The port via which the exception is sent.
toPort	The port to which the exception is sent.
signature	The signature relating to the exception.
parsValue	The signature parameters relating to the exception.
excValue	The exception to be sent.
transmissionFailure	The failure message which might occur at transmission.

Description

This operation is called by CH to log a unicast raise operation. This event occurs after reply execution. This event is used for logging the inter-component communication.

Return Values

None.

tliPrRaise_c_BC

Logs broadcast raise operation.

```
void tliPrRaise_c_BC(String am, long int ts, String src,
long int line, TriComponentId c, TriPortId atPort,
TriPortIdList toPortList, TriSignatureId signature,
TciParameterListType parsValue, TciValue excValue,
TriStatus transmissionFailure);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.

c	The component which produces this event.
atPort	The port via which the exception is sent.
toPortList	List of ports to which the exception is sent.
signature	The signature relating to the exception.
parsValue	The signature parameters relating to the exception.
excValue	The exception to be sent.
transmissionFailure	The failure message which might occur at transmission.

Description

This operation is called by CH to log a broadcast raise operation. This event occurs after reply execution. This event is used for logging the inter-component communication.

Return Values

None.

tliPrRaise_c_MC

Logs multicast raise operation.

```
void tliPrRaise_c_MC(String am, long int ts, String src,
long int line, TriComponentId c, TriPortId atPort,
TriPortIdList toPortList, TriSignatureId signature,
TciParameterListType parsValue, TciValue excValue,
TriStatus transmissionFailure);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.

atPort	The port via which the exception is sent.
toPortList	List of ports to which the exception is sent.
signature	The signature relating to the exception.
parsValue	The signature parameters relating to the exception.
excValue	The exception to be sent.
transmissionFailure	The failure message which might occur at transmission.

Description

This operation is called by CH to log a multicast raise operation. This event occurs after reply execution. This event is used for logging the inter-component communication.

Return Values

None.

tliPrCatchDetected_m

Logs catch enqueue operation.

```
void tliPrCatchDetected_m(String am, long int ts, String
src, long int line, TriComponentId c, TriPortId atPort,
TriPortId fromPort, TriSignatureId signature, TriException
exc, TriAddress address);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.
atPort	The port to which the exception is enqueued.

fromPort	The port from which the exception is sent.
signature	The signature relating to the exception.
exc	The caught exception.
address	The address of the source within the SUT.

Description

This operation is called by SA to log the catch enqueue operation. This event occurs after catch is enqueued. This event is used for logging the communication with the SUT.

Return Values

None.

tliPrCatchDetected_c

Logs catch enqueue operation.

```
void tliPrCatchDetected_c(String am, long int ts, String
src, long int line, TriComponentId c, TriPortId atPort,
TriPortId fromPort, TriSignatureId signature, TciValue
excValue, TriAddress address);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.
atPort	The port to which the exception is enqueued.
fromPort	The port from which the exception is sent.

signature	The signature relating to the exception.
excValue	The caught exception.
address	The address of the source within the SUT.

Description

This operation is called by CH to log the catch enqueue operation. This event occurs after catch is enqueued. This event is used for logging the inter-component communication.

Return Values

None.

tliPrCatchMismatch_m

Logs mismatch of a catch operation.

```
void tliPrCatchMismatch_m(String am, long int ts, String
src, long int line, TriComponentId c, TriPortId port,
TriSignatureId signature, TciValue excValue,
TciValueTemplate excTmpl, TciValueDifferenceList diffs,
TriAddress address, TciValueTemplate addressTmpl);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.
port	The port via which the exception is received.
signature	The signature relating to the exception.
excValue	The received exception.
excTmpl	The template used to check the exception match.

diffs	The difference/the mismatch between exception and template
address	The address of the source within the SUT.
addressTmpl	The expected address of the source within the SUT.

Description

This operation is called by TE to log the mismatch of a catch operation. This event occurs after catch is checked against a template. This event is used for logging the communication with SUT.

Return Values

None.

tliPrCatchMismatch_c

Logs mismatch of a catch operation.

```
void tliPrCatchMismatch_c(String am, long int ts, String
src, long int line, TriComponentId c, TriPortId port,
TriSignatureId signature, TciValue excValue,
TciValueTemplate excTmpl, TciValueDifferenceList diffs,
TriComponentId from, TciNonValueTemplate fromTmpl);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.
port	The port via which the exception is received.
signature	The signature relating to the exception.
excValue	The received exception.
excTmpl	The template used to check the exception match.

diffs	The difference/the mismatch between exception and template
from	The component which sent the reply.
fromTmpl	The expected replying component.

Description

This operation is called by TE to log the mismatch of a catch operation. This event occurs after catch is checked against a template. This event is used for logging the inter-component communication.

Return Values

None.

tliPrCatch_m

Logs catching an exception.

```
void tliPrCatch_m(String am, long int ts, String src, long
int line, TriComponentId c, TriPortId port, TriSignatureId
signature, TciValue excValue, TciValueTemplate excTmpl,
TriAddress address, TciValueTemplate addressTmpl);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.
port	The port via which the exception is received.
signature	The signature relating to the exception.
excValue	The received exception.

excTmpl	The template used to check the exception match.
address	The address of the source within the SUT.
addressTmpl	The expected address of the source within the SUT.

Description

This operation is called by SA to log catching an exception. This event occurs after catch is checked against a template. This event is used for logging the communication with SUT.

Return Values

None.

tliPrCatch_c

Logs catching an exception.

```
void tliPrCatch_c(String am, long int ts, String src, long
int line, TriComponentId c, TriPortId port, TriSignatureId
signature, TciValue excValue, TciValueTemplate excTmpl,
TriComponentId from, TciNonValueTemplate fromTmpl);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.
port	The port via which the exception is received.
signature	The signature relating to the exception.
excValue	The received exception.

excTmpl	The template used to check the exception match.
from	The component which sent the reply.
fromTmpl	The expected replying component.

Description

This operation is called by CH to log catching an exception. This event occurs after catch is checked against a template. This event is used for logging the inter-component communication.

Return Values

None.

tliPrCatchTimeoutDetected

Logs detection of a catch timeout.

```
void tliPrCatchTimeoutDetected(String am, long int ts,
String src, long int line, TriComponentId c, TriPortId
port, TriSignatureId signature);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	the line number where the request is performed.
c	The component which produces this event.
port	The port via which the exception is received.
signature	The signature relating to the exception.

Description

This operation is called by PA to log the detection of a catch timeout. This event occurs after the timeout is enqueued.

Return Values

None.

tliPrCatchTimeout

Logs catching a timeout.

```
void tliPrCatchTimeout (String am, long int ts, String src,
long int line, TriComponentId c, TriPortId port,
TriSignatureId signature);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.
`port`	The port via which the exception is received.
`signature`	The signature relating to the exception.

Description

This operation is called by TE to log catching a timeout. This event occurs after the catch timeout has been performed.

Return Values

None.

tliCCreate

Logs create component operation.

```
void tliCCreate(String am, long int ts, String src, long
int line, TriComponentId c, TriComponentId comp, String
name, unsigned char alive);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.
comp	The component which is created.
name	Name of the created component.
alive	Signals whether component is alive.

Description

This operation is called by TE to log the create component operation. This event occurs after component creation.

Return Values

None.

tliCStart

Logs start component operation.

```
void tliCStart(String am, long int ts, String src, long int
line, TriComponentId c, TriComponentId comp,
TciBehaviourIdType beh, TciParameterListType pars);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.

comp	The component which is started.
beh	The behavior being started on the component.
pars	The parameters of the started behavior.

Description

This operation is called by TE to log the start component operation. This event occurs after component start.

Return Values

None.

tliCRunning

Logs running component operation.

```
void tliCRunning(String am, long int ts, String src, long
int line, TriComponentId c, TriComponentId comp,
ComponentStatus status);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.
comp	The component which is checked to be running.
status	The status of this component.

Description

This operation is called by TE to log the running component operation. This event occurs after component running.

Return Values

None.

tliCAlive

Logs alive component operation.

```
void tliCAlive(String am, long int ts, String src, long int
line, TriComponentId c, TriComponentId comp,
ComponentStatus status);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.
`comp`	The component which is checked to be running.
`status`	The status of this component.

Description

This operation is called by TE to log the alive component operation. This event occurs after component alive.

Return Values

None.

tliCStop

Logs stop component operation.

```
void tliCStop(String am, long int ts, String src, long int
line, TriComponentId c, TriComponentId comp);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.
`comp`	The component which is stopped.

Description

This operation is called by TE to log the stop component operation. This event occurs after component stop.

Return Values

None.

tliCKill

Logs kill component operation.

```
void tliCKill(String am, long int ts, String src, long int
line, TriComponentId c, TriComponentId comp);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.
`comp`	The component which is stopped.

Description

This operation is called by TE to log the kill component operation. This event occurs after component kill.

Return Values

None.

tliCDoneMismatch

Logs mismatch of a done component operation.

```
void tliCDoneMismatch(String am, long int ts, String src,
long int line, TriComponentId c, TriComponentId comp,
TciNonValueTemplate compTmpl);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.
`comp`	The first component which is not yet done.
`compTmpl`	The template used to check the done match.

Description

This operation is called by TE to log the mismatch of a done component operation. This event occurs after done is checked against a template.

Return Values

None.

tliCDone

Logs done component operation.

```
void tliCDone (String am, long int ts, String src, long int
line, TriComponentId c, TriComponentId comp,
TciNonValueTemplate compTmpl);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.
comp	The component which is done.
compTmpl	The template used to check the done match.

Description

This operation is called by TE to log the done component operation. This event occurs after the done operation.

Return Values

None.

tliCKilledMismatch

Logs mismatch of a killed component operation.

```
void tliCKilledMismatch(String am, long int ts, String src,
long int line, TriComponentId c, TriComponentId comp,
TciNonValueTemplate compTmpl);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.

line	The line number where the request is performed.
c	The component which produces this event.
comp	The first component which is not yet killed.
compTmpl	The template used to check the done match.

Description

This operation is called by TE to log the mismatch of a killed component operation. This event occurs after killed is checked against a template.

Return Values

None.

tliCKilled

Logs killed component operation.

```
void tliCKilled (String am, long int ts, String src, long
int line, TriComponentId c, TriComponentId comp,
TciNonValueTemplate compTmpl);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.
comp	The component which is killed.
compTmpl	The template used to check the done match.

Description

This operation is called by TE to log the killed component operation. This event occurs after the killed operation.

Return Values

None.

tliCTerminated

Logs termination of a component.

```
void tliCTerminated(String am, long int ts, String src,
long int line, TriComponentId c, TriComponentId comp,
TciValue verdict);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.
verdict	The verdict of the component.

Description

This operation is called by TE to log the termination of a component. This event occurs after the termination of the component.

Return Values

None.

tliPConnect

Logs connect operation.

```
void tliPConnect(String am, long int ts, String src, long
int line, TriComponentId c, TriPortId port1, TriPortId
port2);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.
`port1`	The first port to be connected.
`port2`	The second port to be connected.

Description

This operation is called by CH to log the connect operation. This event occurs after the connect operation.

Return Values

None.

tliPDisconnect

Logs disconnect operation.

```
void tliPDisconnect(String am, long int ts, String src,
long int line, TriComponentId c, TriPortId port1, TriPortId
port2);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.

c	The component which produces this event.
port1	The first port to be disconnected.
port2	The second port to be disconnected.

Description

This operation is called by CH to log the disconnect operation. This event occurs after the disconnect operation.

Return Values

None.

tliPMap

Logs map operation.

```
void tliPMap(String am, long int ts, String src, long int
line, TriComponentId c, TriPortId port1, TriPortId port2);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.
port1	The first port to be mapped.
port2	The second port to be mapped.

Description

This operation is called by SA to log the map operation. This event occurs after the map operation.

Return Values

None.

tliPUnmap

Logs an un-map operation.

```
void tliPUnmap(String am, long int ts, String src, long int
line, TriComponentId c, TriPortId port1, TriPortId port2);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.
`port1`	The first port to be unmapped.
`port2`	The second port to be unmapped.

Description

This operation is called by SA to log an un-map operation. This event occurs after the un-map operation.

Return Values

None.

tliPClear

Logs port clear operation.

```
void tliPClear(String am, long int ts, String src, long int
line, TriComponentId c, TriPortId port);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.
port	The port to be cleared.

Description

This operation is called by TE to log the port clear operation. This event occurs after the port clear operation.

Return Values

None.

tliPStart

Logs port start operation.

```
void tliPStart(String am, long int ts, String src, long int
line, TriComponentId c, TriPortId port);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.
port	The port to be started.

Description

This operation is called by TE to log the port start operation. This event occurs after the port start operation.

Return Values

None.

tliPStop

Logs port stop operation.

```
void tliPStop(String am, long int ts, String src, long int
line, TriComponentId c, TriPortId port);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.
`port`	The port to be stopped.

Description

This operation is called by TE to log the port stop operation. This event occurs after the port stop operation.

Return Values

None.

tliPHalt

Logs port halt operation.

```
void tliPHalt(String am, long int ts, String src, long int
line, TriComponentId c, TriPortId port);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.
`port`	The port to be stopped.

Description

This operation is called by TE to log the port halt operation. This event occurs after the port halt operation.

Return Values

None.

tliEncode

Logs encode operation.

```
void tliEncode(String am, long int ts, String src, long int
line, TriComponentId c, TciValue val, TriStatus
encoderFailure, TriMessage msg, String codec);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.
`value`	The value to be encoded.

encoderFailure	The failure message which might occur at encoding.
msg	The encoded value.
codec	The used encoder.

Description

This operation is called by CD to log the encode operation.

Return Values

None.

tliDecode

Logs decode operation.

```
void tliDecode(String am, long int ts, String src, long int
line, TriComponentId c, TciValue val, TriStatus
decoderFailure, TriMessage msg, String codec);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.
msg	The value to be decoded.
decoderFailure	The failure message which might occur at decoding.
value	The decoded value.
codec	The used decoder.

Description

This operation is called by CD to log the decode operation.

Return Values

None.

tliTTimeoutDetected

Logs detection of a timeout.

```
void tliTTimeoutDetected(String am, long int ts, String
src, long int line, TriComponentId c, TriTimerId timer);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.
`timer`	The timer that timed out.

Description

This operation is called by PA to log the detection of a timeout. This event occurs after timeout is enqueued.

Return Values

None.

tliTTimeoutMismatch

Logs timeout mismatch.

```
void tliTTimeoutMismatch(String am, long int ts, String
src, long int line, TriComponentId c, TriTimerId timer,
TciNonValueTemplate timerTmpl);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.
`timer`	The first timer which is not yet stopped.
`timerTmpl`	The timer template that did not match.

Description

This operation is called by TE to log a timeout mismatch. This event occurs after a timeout match failed.

Return Values

None.

tliTTimeout

Logs timeout match.

```
void tliTTimeoutMismatch(String am, long int ts, String
src, long int line, TriComponentId c, TciNonValueTemplate
timerTmpl);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.

c	The component which produces this event.
timer	The timer which timed out.
timerTmpl	The timer template that matched.

Description

This operation is called by TE to log a timeout match. This event occurs after a timeout matched.

Return Values

None.

tliTStart

Logs start of a timer.

```
void tliTStart(String am, long int ts, String src, long int
line, TriComponentId c, TriTimerId timer, TriTimerDuration
dur);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.
timer	The timer that is started.
dur	The timer duration.

Description

This operation is called by PA to log the start of a timer. This event occurs after the start timer operation.

Return Values

None.

tliTStop

Logs stop of a timer.

```
void tliTStop(String am, long int ts, String src, long int
line, TriComponentId c, TriTimerId timer, TriTimerDuration
dur);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.
`timer`	The timer that is stopped.
`dur`	Timer duration at the moment of stopping it.

Description

This operation is called by PA to log the stop of a timer. This event occurs after the stop timer operation.

Return Values

None.

tliTRead

Logs reading of a timer.

```
void tliTRead(String am, long int ts, String src, long int
line, TriComponentId c, TriTimerId timer, TriTimerDuration
elapsed);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.
timer	The timer that is started.
elapsed	The elapsed time of the timer.

Description

This operation is called by PA to log the reading of a timer. This event occurs after the read timer operation.

Return Values

None.

tliTRunning

Logs running timer operation.

```
void tliTRunning(String am, long int ts, String src, long
int line, TriComponentId c, TriTimerId timer, TimerStatus
status);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.

c	The component which produces this event.
timer	The timer which is checked to be running.
status	The status of this component.

Description

This operation is called by PA to log the running timer operation. This event occurs after the running timer operation.

Return Values

None.

tliSEnter

Logs entering of a scope.

```
void tliSEnter(String am, long int ts, String src, long int
line, TriComponentId c, QualifiedName name,
TciParameterListType parsValue, String kind);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.
name	The name of the scope.
parsValue	The parameters of the scope.
kind	The kind of the scope.

Description

This operation is called by TE to log the entering of a scope. This event occurs after the scope has been entered.

Return Values

None.

tliSLeave

Logs leaving of a scope.

```
void tliSLeave(String am, long int ts, String src, long int
line, TriComponentId c, QualifiedName name,
TciParameterListType parsValue, TciValue val, String kind);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.
`name`	The name of the scope.
`val`	The return value of the scope.
`parsValue`	Values of formal parameters when leaving scope.
`kind`	The kind of the scope.

Description

This operation is called by TE to log the leaving of a scope. This event occurs after the scope has been left.

Return Values

None.

tliVar

Logs modification of the value of a variable.

```
void tliVar(String am, long int ts, String src, long int
```

```
line, TriComponentId c, QualifiedName name, TciValue
varValue);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.
name	The name of the variable.
varValue	The new value of the variable.

Description

This operation is called by TE to log the modification of the value of a variable. This event occurs after the values have been changed.

Return Values

None.

tliModulePar

Logs value of a module parameter.

```
void tliModulePar(String am, long int ts, String src, long
int line, TriComponentId c, QualifiedName name, TciValue
parValue);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.

c	The component which produces this event.
name	The name of the module parameter.
parValue	The value of the module parameter.

Description

This operation is called by TE to log the value of a module parameter. This event occurs after the access to the value of a module parameter.

Return Values

None.

tliGetVerdict

Logs a getverdict operation.

```
void tliGetVerdict(String am, long int ts, String src, long
int line, TriComponentId c, TciValue verdict);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.
verdict	The current value of the local verdict.

Description

This operation is called by TE to log the getverdict operation. This event occurs after the getverdict operation.

Return Values

None.

tliSetVerdict

Logs `setverdict` operation.

```
void tliSetVerdict(String am, long int ts, String src, long
int line, TriComponentId c, TciValue verdict);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.
`verdict`	The value to be set to the local verdict.

Description

This operation is called by TE to log the `setverdict` operation. This event occurs after the `setverdict` operation.

Return Values

None.

tliLog

Logs TTCN-3 statement log.

```
void tliLog (String am, long int ts, String src, long int
line, TriComponentId c, String log);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.

line	The line number where the request is performed.
c	The component which produces this event.
log	Value to be logged.

Description

This operation is called by TM to log the TTCN-3 statement log. This event occurs after the TTCN-3 log operation.

Return Values

None.

tliAEnter

Logs entering an alt.

```
void tliAEnter(String am, long int ts, String src, long int
line, TriComponentId c);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.

Description

This operation is called by TE to log entering an alt. This event occurs after an alt has been entered.

Return Values

None.

tliALeave

Logs leaving an alt.

```
void tliALeave(String am, long int ts, String src, long int
line, TriComponentId c);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.

Description

This operation is called by TE to log leaving an alt. This event occurs after the alt has been leaved.

Return Values

None.

tliANomatch

Logs a no-match of an alt.

```
void tliANomatch (String am, long int ts, String src, long
int line, TriComponentId c);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).

src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.

Description

This operation is called by TE to log the no-match of an alt. This event occurs after the alt has not matched.

Return Values

None.

tliARepeat

Logs repeating an alt.

```
void tliARepeat(String am, long int ts, String src, long
int line, TriComponentId c);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.

Description

This operation is called by TE to log repeating an alt. This event occurs when the alt is been repeated.

Return Values

None.

tliADefaults

Logs entering the default section.

```
void tliADefaults(String am, long int ts, String src, long
int line, TriComponentId c);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.

Description

This operation is called by TE to log entering the default section. This event occurs after the default section has been entered.

Return Values

None.

tliAActivate

Logs activation of a default.

```
void tliAActivate(String am, long int ts, String src, long
int line, TriComponentId c, QualifiedName name,
TciParameterListType pars, TciValue ref);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.

line	The line number where the request is performed.
c	The component which produces this event.
name	The name of the default.
pars	The parameter of the default.
ref	The resulting default reference.

Description

This operation is called by TE to log the activation of a default. This event occurs after the default activation.

Return Values

None.

tliADeactivate

Logs deactivation of a default.

```
void tliADeactivate(String am, long int ts, String src,
long int line, TriComponentId c, TciValue ref);
```

Parameters

am	An additional message.
ts	The time when the event is produced (in milliseconds from process start).
src	The source file of the test specification.
line	The line number where the request is performed.
c	The component which produces this event.
ref	The resulting default reference.

Description

This operation is called by TE to log the deactivation of a default. This event occurs after the default deactivation.

Return Values

None.

tliAWait

Logs that the component awaits events for a new snapshot.

```
void tliAWait(String am, long ts, String src, long line,
TriComponentId c);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.

Description

This operation is called by TE to log that the component awaits events for a new snapshot.

Return Values

None.

tliAction

Logs the SUT action statement.

```
void tliAction(String am, long ts, String src, long line,
TriComponentId c, String action);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.
`action`	SUT action string.

Description

This operation is called by TE to log that the SUT action statement.

Return Values

None.

tliMatch

Logs the successfully executed match operation.

```
void tliMatch(String am, long ts, String src, long line,
TriComponentId c, TciValue expr, TciValueTemplate tmpl);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.
`expr`	The value which is matched.
`tmpl`	The template which is used in matching operation.

Description

This operation is called by TE to log successfully executed match operation.

Return Values

None.

tliMatchMismatch

Logs the unsuccessfully executed match operation - mismatch occurred .

```
void tliMatchMismatch(String am, long ts, String src, long
line, TriComponentId c, TciValue expr, TciValueTemplate
tmpl, TciValueDifferenceList diffs);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.
`expr`	The value which is matched.
`tmpl`	The template which is used in matching operation.
`diffs`	List of differences between value and template.

Description

This operation is called by TE to log unsuccessfully executed match operation - mismatch occurred.

Return Values

None.

tliInfo

Logs additional test suite execution information.

```
void tliInfo(String am, long ts, String src, long line,
TriComponentId c, long int level, String info);
```

Parameters

`am`	An additional message.
`ts`	The time when the event is produced (in milliseconds from process start).
`src`	The source file of the test specification.
`line`	The line number where the request is performed.
`c`	The component which produces this event.
`level`	Severity of the informal message
`info`	Text information

Description

This operation is used to log additional information during test execution. The generation of this event is tool dependent as well as the usage of the parameters level and info.

Return Values

None.

TCI Template Interface

tciIsOmitValueTemplate

Checks whether specified template is 'omit'

```
unsigned char tciIsOmitValueTemplate(TciValueTemplate
templateId);
```

Parameters

`templateId`	identifier of the value template instance

Description

This operation may be called to check whether template represent 'omit' value template or not.

Return Values

Returns 'true' if specified template represents 'omit' value template, false otherwise.

tciIsAnyValueTemplate

Checks whether specified template is '?'

```
unsigned char tciIsAnyValueTemplate(TciValueTemplate
templateId);
```

Parameters

`templateId`	identifier of the value template instance

Description

This operation may be called to check whether template represent any ('?') value template or not.

Return Values

Returns 'true' if specified template represents any ('?') value template, false otherwise.

tciIsAnyOrOmitValueTemplate

Checks whether specified template is '*'

```
unsigned char tciIsAnyOrOmitValueTemplate(TciValueTemplate
templateId);
```

Parameters

`templateId`	identifier of the value template instance

Description

This operation may be called to check whether template represent any or omit ('*') value template or not.

Return Values

Returns 'true' if specified template represents any or omit ('*') value template, false otherwise.

tciGetValueTemplateDef

Returns string representation of the template definition

```
String tciGetValueTemplateDef(TciValueTemplate templateId);
```

Parameters

`templateId`	identifier of the value template instance

Description

This operation may be called to obtain string representation of a value template definition.

Return Values

Returns string representation of the template definition for specified value template

tciIsAnyNonValueTemplate

Checks whether specified template is 'any <instance>'

```
unsigned char tciIsAnyNonValueTemplate(TciNonValueTemplate
inst);
```

Parameters

`templateId`	identifier of the value template instance

Description

This operation may be called to check whether template represent any ('any <instance>') non-value template or not.

Return Values

Returns 'true' if specified template represents any ('any <instance>') non-value template, false otherwise.

tciIsAllNonValueTemplate

Checks whether specified template is 'all <instance>'

```
unsigned char tciIsAllNonValueTemplate(TciNonValueTemplate
inst);
```

Parameters

`templateId`	identifier of the non-value template instance

Description

This operation may be called to check whether template represent any ('all <instance>') non-value template or not.

Return Values

Returns 'true' if specified template represents any ('all <instance>') non-value template, false otherwise.

tciGetNonValueTemplateDef

Returns string representation of the template definition

```
String tciGetNonValueTemplateDef(TciNonValueTemplate
templateId);
```

Parameters

`templateId`	identifier of the non-value template instance

Description

This operation may be called to obtain string representation of a non-value template definition.

Return Values

Returns string representation of the template definition for specified non-value template

11

Regular Expressions in Search

Regular Expressions in Search

You may use the following regular expressions when finding and replacing in IBM Rational Systems Tester.

Regular expression	Explanation
.	(Period.) Any single character.
[]	Any one of the characters contained in the brackets, or any of an ASCII range of characters separated by a hyphen (-). For example, • `b[aeiou]d` matches bad, bed, bid, bod, and bud • `r[eo]+d` matches red, rod, reed, and rood, but not reod or roed. • `x[0-9]` matches x0, x1, x2, and so on. If the first character in the brackets is a caret (^), then the regular expression matches any characters except those in the brackets.
^	The beginning of a line.
$	The end of a line.
\(\)	Indicates a tagged expression to retain for replacement purposes. If the expression in the Find What box is `\(lpsz\)BigPointer`, and the expression in the Replace With box is \1NewPointer, all selected occurrences of `lpszBigPointer` are replaced with `lpszNewPointer`. Each occurrence of a tagged expression is numbered according to its order in the Find What box, and its replacement expression is \n, where 1 corresponds to the first tagged expression, 2 to the second, and so on. You can have up to nine tagged expressions.

Regular expression	Explanation
\~	No match if the following character or characters occur. For example, `b\~a+d` matches `bbd`, `bcd`, `bdd`, and so on, but not `bad`. You can use this expression to prefix a group of characters you want to exclude, which is useful for excluding matches of particular words. For example, `foo\~\(lish\)` matches `foo` in `food` and `afoot` but not in `foolish`.
`\{c\!c\}`	Any one of the characters separated by the alternation symbol (\!). For example, `\{j\!u\}+fruit` finds `jfruit`, `jjfruit`, `ufruit`, `ujfruit`, `uufruit`, and so on.
*	None or more of the preceding characters or expressions. For example, `ba*c` matches `bc`, `bac`, `baac`, `baaac`, and so on.
+	At least one or more of the preceding characters or expressions. For example, `ba+c` matches `bac`, `baac`, `baaac`, but not `bc`.
\{\}	Any sequence of characters between the escaped braces. For example, `\{ju\}+fruit` finds `jufruit`, `jujufruit`, `jujujufruit`, and so on. It will not find `jfruit`, `ufruit`, or `ujfruit`, because the sequence `ju` is not in any of those strings.
[^]	Any character except those following the caret (^) character in the brackets, or any of an ASCII range of characters separated by a hyphen (-). For example, `x[^0-9]` matches `xa`, `xb`, `xc`, and so on, but not `x0`, `x1`, `x2`, and so on.
\:a	Any single alphanumeric character [`a – z,A – Z,0 – 9`].
\:b	Any white-space character. The `\:b` finds tabs and spaces. There is no alternate syntax to express `:b`.
\:c	Any single alphabetic character [`a – z,A – Z`].
\:d	Any decimal digit [0 – 9].

Regular expression	Explanation
\:n	Any unsigned number \{[0-9]+\.[0-9]*\![0-9]*\.[0-9]+\![0-9]+\}. For example, \:n should match `123`, `.45`, and `123.45`.
\:z	Any unsigned decimal integer [0 – 9]+.
\:h	Any hexadecimal number [`0-9 a-f A-F`]+.
\:i	Any C/C++ identifier [a – z, A – Z_$] [a – z, A – Z,0 – 9_$]+.
\:w	Any alphabetic string [a – z, A – Z]+. The string need not be bounded by white space or appear at the beginning or the end of a line.
\:q	Any quoted string `\{"[^"]*"\!'[^']*'\}`.
\	Removes the pattern match characteristic in the Find What text box from the special characters listed above. For example, `100$` matches `100` at the end of a line, but `100\$` matches the character string `100$` anywhere on a line.

Common Reference

The reference chapters listed in this section describe functionality that is valid for all types of IBM Rational Systems Tester projects.

12

Useful Shortcut Keys

This section lists useful shortcut keys that you can use. Access keys can be used in the same way as other standard applications.

(UNIX) This only applies to Exceed users: If you use a non-American keyboard, you must map the ALT button correctly in order to use the access keys.

To map the ALT button:

1. Click the **Start** button, point to **Programs**, point to **Exceed** and click **xconfig**.
2. In the dialog that opens, double-click **Input**. The Input dialog opens.
3. In the **Alt key** field, select **To X** or **Right To Window, Left To X**.
4. Close the dialog.

Note

UNIX: Some short-cut sequences (such as ALT-X, CTRL-H) may be intercepted by the X11 window manager and can cause other actions to be taken than those described in this manual. In most cases it is possible to configure the X11 window manager not to intercept specific short-cut sequences. Please refer to the documentation on your window manager for further information regarding short-cuts.

Workspace Operations

Keyboard shortcut	Description
CTRL + N Then CTRL + TAB to Workspaces tab	Create a new workspace
CTRL + O	Open an existing workspace.
MINUS SIGN (-) on the numeric keypad	Contracts the tree of a selected entity.
MULTIPLICATION SIGN (*) on the numeric keypad	Expands the model tree one level below the selection. Can be used repeatedly to expand deeper.
PLUS SIGN (+) on the numeric keypad	Expands the selection.

Project Operations

Keyboard shortcut	Description
CTRL + N Then CTRL + TAB to Project tab	Create a new project
CTRL + O	Open project.

File Operations

Keyboard shortcut	Description
CTRL + N	Create a new file
CTRL + O	Open a file
CTRL + P	Print the active document
CTRL + S	Save active document

Navigate in Files

Keyboard shortcut	Description
CTRL + DOWN ARROW	Scroll down a few rows at a time, without moving the insertion point
CTRL + END	Move insertion point to end of file
CTRL + SHIFT + G	Opens the **Go to line number** dialog
CTRL + HOME	Move insertion point to beginning of file
CTRL + LEFT ARROW	Step left one word at a time
CTRL + M	Open Navigator tab in Output window
CTRL + RIGHT ARROW	Step right one word at a time
CTRL + UP ARROW	Scroll up a few rows at a time, without moving the insertion point
END	Move insertion point to end of line
HOME	Move insertion point to beginning of line

Navigate in TTCN-3 Files

Keyboard shortcut	Description
ALT + F2	Edit named bookmarks
CTRL + E	Toggle insertion point between beginning and end of current scope
CTRL + F2	Toggle an unnamed bookmark
CTRL + G	Go to a line number or named bookmark
CTRL + SPACEBAR	Display entity list
F2	Go to the next bookmark
SHIFT + F2	Go to the previous unnamed bookmark

Highlight Text

Keyboard shortcut	Description
CTRL + A	Select all (**TTCN-3 only**)
CTRL + SHIFT + END	Highlight text to the end of the file
CTRL + SHIFT + HOME	Highlight text to the beginning of the file
CTRL + SHIFT + LEFT ARROW	Highlight one word at a time to the left
CTRL + SHIFT + RIGHT ARROW	Highlight one word at a time to the right
SHIFT + DOWN ARROW	Highlight one row downwards
SHIFT + END	Highlight to the end of the line
SHIFT + HOME	Highlight to the beginning of the line
SHIFT + LEFT ARROW	Highlight one character at a time to the left
SHIFT + RIGHT ARROW	Highlight one character at a time to the right
SHIFT + UP ARROW	Highlight one row upwards

Edit Text

Keyboard shortcut	Description
CTRL + A	Select all
CTRL + ALT + F	Find in files (**TTCN-3**)
CTRL + ALT + I	Find previous incremental search match (**TTCN-3**)
CTRL + BACK-SPACE	Step left in a file and delete a whole word at a time (**TTCN-3**)

Keyboard shortcut	Description
CTRL + C	Copy
CTRL + DELETE	Delete the word to the right of the cursor (**TTCN-3**)
CTRL + F	Find in active file
CTRL + H	Replace
CTRL + I	Go to the next incremental search match (**TTCN-3**)
CTRL + SPACEBAR SHIFT + SPACEBAR	Name completion, if a definition is found that matches the current name up to the cursor position. If there are multiple matches a Name completion scroll menu will open.
CTRL + V	Paste
CTRL + X	Cut
CTRL + Y	Redo
CTRL + Z	Undo
F3	Go to the next find match (**TTCN-3**)
F1	Help with textual syntax on current selection.
SHIFT + arrow keys	Extends the current text selection. Requires that text is selected
SHIFT + END	Selects text from cursor position to end of text row.
SHIFT + F3	Go to the previous find match (**TTCN-3 only**)
SHIFT + HOME	Selects text from start of text row to cursor position.

Application Builder Shortcuts

Keyboard shortcut	Description
CTRL + SCROLL LOCK	Stops the build process
F5	Launch the current configuration
F6	Generate current configuration

Keyboard shortcut	Description
F7	Build the current configuration
SHIFT + F5	Stops the execution
SHIFT + F6	Update configuration

Window Navigation

Keyboard shortcut	Description
ALT + 1	Toggle full screen mode
CTRL + F4	Close the active window
CTRL + SHIFT + TAB CTRL + SHIFT + F6	Navigate to the previous window
CTRL + TAB CTRL + F6	Navigate to the next window

Properties editor

Keyboard shortcut	Description
ALT + ENTER	Display Properties editor
CTRL + BACKSPACE	Go to owner, change scope in model tree to the owner of the current selection
CTRL + ALT + C	Switch to Control view
CTRL + ALT + T	Switch to Text view

Show/Hide Windows and Dialogs

Keyboard shortcut	Description
ALT + 0	Show/ hide workspace window
ALT + 2	Show/ hide Output window
ALT + ENTER	Display Properties editor
CTRL + Q	Open Query dialog on selection
F1	Display Help

Zoom/Pan

Keyboard shortcut	Description
<Rotate the wheel button>	Scroll the diagram vertically (requires an IntelliMouse pointing device)
<Double-click middle mouse button>	Zoom to 100%
SHIFT + <double-click middle mouse button>	Zoom to fit editor window
SHIFT + <rotate wheel button>	Zoom in or zoom out depending on the rotate direction.The zoom in point will be where the mouse pointer is located
CTRL + SHIFT + <Rotate the wheel button>	When a single line is selected the diagram will be scrolled along the line until one of the endpoints are centered in view (requires an IntelliMouse pointing device)
MINUS SIGN (-) on the numeric keypad	Zoom out 25% (This works when a diagram is active and not in text edit mode for any element)

Keyboard shortcut	Description
PLUS SIGN (+) on the numeric keypad	Zoom in 25% (This works when a diagram is active and not in text edit mode for any element)
LESS-THAN SIGN (<)	When a single line is selected the diagram will be scrolled to the source endpoint of the line.
GREATER-THAN SIGN (>)	When a single line is selected the diagram will be scrolled to the destination endpoint of the line.

13

Setting Up the Tool Environment

This section mainly provides information how to integrate IBM Rational Systems Tester with different tools.

Working with Links

The link concept in IBM Rational Systems Tester allows any element to be linked either to another element or to an external element. The external element can be a file or a web page. It is also possible to support links specific to an integration with other software tools.

Link options

IBM Rational Systems Tester offers you the possibility to customize link creation behavior. You can change link options via the Tools menu by selecting Options or clicking on the Display Link Options toolbar button.

Active link end is an active target, not an active source

If this option is disabled, then when you use automatic creation of links, you will create links from your active link end to the other models.

If this option is enabled, then you will create links to your active link end from the other models.

Automatically create links between modified objects and active link end

If this option is enabled, then when you select an active link end, all your modifications will be linked to this link end.

Show link indicators

If this option is enabled, IBM Rational Systems Tester will show the link markers.

Hyperlink options

IBM Rational Systems Tester offers you the possibility to customize link creation behavior via the Tools menu by selecting Options or clicking on the Display Link Options toolbar button.

By default, make hyperlink to a workspace element

When the Insert hyperlink dialog is started this option controls the start setting of the **Link to** text field.

When checked, the application allows you to select the target of your hyperlink within your workspace. Otherwise, you are prompted to specify another type of target, such as an existing file or a web page.

Insert hyperlink dialog

This dialog allows you to search for a target for a link. The following is present in the dialog.

- **Link to:** text field allowing a selection if the link is external (web page or file) or within the elements of the current workspace.
- **Text to display** will be visible through the link output tab or the Links dialog, in the **name** column.
- **Unique Resource Location of the hyperlink target:** text field allowing you to specify a web page location directly.
- List for selection of recently used files, web pages and links, with radio buttons for **Recent files**, **Browsed pages** and **Inserted links** to be displayed in the field.

URL

With respect to Hyperlinks, objects that have a link going to them are identified by a unique string. IBM Rational Systems Tester has its own protocol for representing link targets.

A Uniform Resource Location (URL) describes where to find a given object of IBM Rational Systems Tester, it appears as:

```
tlog://<project_path>:plugin_ident:string_ident
```

It consists of the following parts:

- `<project_path>` is the windows path of the project containing the object.
- `plugin_ident` is the IBM Rational Systems Tester internal identification of the IBM Rational Systems Tester add-in which owns the object.
- `String ident` is a string representing the object for a given project of a given plug in.

URLs in `html` pages or in Microsoft Word documents following this format will allow a web browser to launch IBM Rational Systems Tester and select the wished object.

Copy URL

You can get the URL for a presentation element from the shortcut menu command **Copy URL**.

Missing targets

In IBM Rational Systems Tester, depending on the currently active project in your workspace, the link target can be missing.

When a missing target refers to a IBM Rational Systems Tester object, the message that the project is not loaded will be issued.

Note

The link source can not be missing, as the source is the carrier of the link information.

Link operations

Link commands can be operated from:

- The link menu
- The link toolbar
- The shortcut menu will have a context sensitive part with a links section containing a subset of the link commands

The following commands are accessible for links:

Start Link (CTRL+K)

Start Many Links

Make Link from Start (CTRL+L)

Display Outgoing Links

This command will produce a list in the link tab of the Output window of the links going out from the selected entity.

Display Incoming Links

This command will produce a list in the link tab of the Output window of the links coming in to the selected entity.

Edit Links

Link Options

Dependency Add hyperlinks

You can add hyperlinks to elements in your workspace. The hyperlinks can also go to external files or web pages (default). This allows you to show relations to external elements in a simple way. It is also possible to create a hyperlink between elements.

An out link will be indicated with a blue underline.

An in link will be indicated with a dashed blue underline in diagrams and in the Workspace window with a blue arrow to the right of the element linked to.

Add Dependency links

You can add Dependency links to elements in your workspace. This allows you to show relations to for example elements in other packages.

An out link will be modeled as a Dependency.

Add links

Creating a link

1. To create a link, select the object you want as source for the link.
2. Select the command Start Link (CTRL+K) from the Link toolbar, the Link menu or right-click the element and from the shortcut menu and choose Links, then point to **Start Link**.

If you created a Hyperlink:

- From the Insert Hyperlink dialog then select the object that you want as target for the link. It is also possible to change the setting of the **Link to** field and close the dialog to add a link to an element in the current workspace.

If you created a Dependency link:

- Right-click the object that you want as target for the link in the current workspace and select Make Link from Start (CTRL+L).

Automatic creation

There is a special mode for link creation, which is called **Automatic Creation**. By combining Start Many Links and the option Automatically create links between modified objects and active link end, you can trigger an efficient mode of creating links. Launch this mode, and continue work within IBM Rational Systems Tester. IBM Rational Systems Tester will automatically make links.

1. Set the type of links you will make to your active object, incoming or outgoing by specifying the option Active link end is an active target, not an active source.
2. Select your active link end.

Link messages

Whatever means you used to create a link, a message in the Messages output tab indicates that a link has been created.

Configuration Management

IBM Rational Systems Tester supports integration schemes with configuration management tools.

- An integration scheme based on the Microsoft Source Control Integration Interface. This means that as long as the source control system that you use support the Microsoft Source Control Integration Interface it should work with IBM Rational Systems Tester. There is currently support for Integration with IBM Rational ClearCase which is regularly verified to work using Microsoft Source Control Integration Interface.
- A utility to start Tau Compare and Merge operations on u2 file versions selected directly in the configuration management tool.

Source control provider

The General tab of **Tools**->**Options** contains the **Source control provider** drop down that enables the user to select source control scheme. IBM Rational Systems Tester must be restarted for this to take effect.

See also

"Source control information" on page 769

"Multiple configuration management tools" on page 772

"Source control commands" on page 773

Source control information

The status of the files in the configuration management tool is displayed by different icons in the File View. The following icons apply:

- **blue check mark**

  This icon indicates that the file or folder is checked out.
- **orange check mark**

  This icon is used for folders that are partially checked out. It indicates that in the folder there are files that are checked in and files that are checked out or that the folder contains files that are not added to source control.

- **red x**

  This icon indicates that the file or folder is checked in.

- **no icon**

  If no icon is available the file or folder is not added to the configuration management tool data base.

The status can also be viewed in the property page of a file.

Note

It is necessary to install the Synergy integration via a separate installer before it is possible to activate the integration.

Generic Source Code Control Integration

Integration with IBM Rational ClearCase

This section describes how to integrate Rational ClearCase with IBM Rational Systems Tester. For further details see the Rational ClearCase user documentation.

The integration uses the Microsoft Source Control Interface and the commands that are supported for ClearCase are the same as described in section "Source control commands" on page 773.

Install IBM Rational ClearCase integration

Windows

The source control system that will be used is specified in a system registry When ClearCase is installed, this source control system registry key should be set automatically. However, if you have more than one configuration management tool installed locally, you may have to edit this value manually.

1. Start IBM Rational Systems Tester.
2. On the **Tools** menu, click **Options**.
3. In the general tab, select **Generic Source Control (SCC)** in the Source control provider choice.

4. Click **OK**.

The next time you start IBM Rational Systems Tester, a new menu, Source Control, will be available from the Project menu. A new toolbar is also added as well.

UNIX

You need to set the register keys and the environment variables. You should be logged on with your normal user identity. If you run on a network with multiple UNIX versions available you have to make a set-up for each UNIX version.

1. Make sure that the ClearCase PATH environment variable is set correctly.
2. To set the register keys, run the script:

   ```
   <installationsdir>/bin/setreg_ClearCase
   ```

   You only have to run the script the first time you will access ClearCase.
3. To set the environment variables, run the script:

   ```
   source <installationsdir>bin/setenv_ClearCase
   ```

   If you do not update your login file with this path, you need to re-run it each time you login.
4. Start IBM Rational Systems Tester.
5. On the **Tools** menu, click **Options**.
6. In the general tab, select **Generic Source Control (SCC)** in the Source control provider choice.
7. Click **OK**.

The next time you start IBM Rational Systems Tester, a new menu, Source Control, will be available from the Project menu. A new toolbar is also added as well.

Note

Before you can use the source control commands, the files in your project must be located in a ClearCase view. See the IBM Rational ClearCase documentation for further instructions.

See also

"Configuration Management" on page 769

"Multiple configuration management tools" on page 772

IBM Rational ClearCase user documentation

Multiple configuration management tools

Windows

If you have more than one Configuration Management (CM) tool installed, you must select which CM tool that you will use. This is determined by the value of the registry key:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\SourceCodeControlProvider]
```

If you want to change provider, you must change the value of this registry key.

The providers that you have installed are listed as values for the registry key:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\SourceCodeControlProvider\Inst
alledSCCProviders]
```

Example 26: Registry key settings ----------

Microsoft SourceSafe:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\SourceCodeControlProvider]
   "ProviderRegKey"="Software\Microsoft\SourceSafe\ccm"
```

IBM Rational ClearCase:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\SourceCodeControlProvider]
   "ProviderRegKey"="Software\Atria\ClearCase"
```

UNIX

If you have more than one Configuration Management (CM) tool installed, you must select which CM tool that you will use.

You need to set the register keys and the environment variables. You should be logged on with your normal user identity.

1. To change the register keys, run the script for your configuration management tool:

   ClearCase: `<installationsdir>/bin/setreg_ClearCase`

   You only need to run the script the first time you will access your CM tool.

2. To change the environment variables, run the script for your configuration management tool:

   ClearCase:

   `source <installationsdir>bin/setenv_ClearCase`

   If you previously have updated your login file with one of the paths above, just edit that file.

Source control commands

The basic commands and functions of your CM tool is available in a separate Source Control menu which is available in the Project menu. The commands are also available via the source control toolbar.

For a number of commands, a file dialog opens where you can select which files that the command should apply to. Depending on the command and the file or folder you selected before the command is issued, the listed files in the dialog may vary. For instance if you select the check out command on a folder, only files that are still checked in are listed in the dialog.

Note

The following commands are defined in the Microsoft SCC Interface documentation. All commands may not be available for your configuration management tool and the functionality of the commands may vary. Your configuration tool may also add commands and toolbar buttons that are unique for that configuration management tool. They are not listed here.

Get latest version

This command retrieves the latest copy of the file from the source control server and copies it to your computer. It is still read-only. If you are working in on a project with more than one team member, update your local copies frequently to incorporate changes made by others.

1. Click the file or folder in the File View.
2. On the **Project** menu, point to **Source Control** and click **Get Latest Version**. The file selection dialog opens.
3. Select the files you want to update and press **OK**.

Check out

This command retrieves the latest version of a file or folder from the source control server and reserves the file or folder for you. The file is copied to your computer and the status changes from read-only to read/write. Unless you allow multiple check outs, the file is locked on your source control server.

Use CTRL + ENTER for line breaks in the comment field.

1. Click the file or folder in the File View.
2. On the **Project** menu, point to **Source Control** and click **Check Out**. The file selection dialog opens.
3. Select the files you want to check out and press **OK**.

Check in

This command copies your local version to the source control server as the latest version of the item. The status of the item changes from read/write to read-only. It is now possible for other users to check out the file. If you have not made any changes to the item you should undo the check out rather than check in the item.

Use CTRL + ENTER for line breaks in the comment field.

1. Make sure that you have saved your files.
2. Click the file or folder in the File View.
3. On the **Project** menu, point to **Source Control** and click **Check In**. The file selection dialog opens.
4. Select the files you want to check in and press **OK**.

Elements that are checked in and therefore not editable are marked with a Gray bar between the element symbol and the element name. The bar does not relate to if the file that the element belongs to is writable, it is an internal flag only.

Undo check out

This command returns the item to the source control server. No changes are saved. This is the command you should use if you have not made any changes to a file that you have checked out.

1. Click the file or folder in the File View.
2. On the **Project** menu, point to **Source Control** and click **Undo Check Out**. The file selection dialog opens.
3. Select the files you want to undo the check out and press **OK**.

Add to source control

This command adds the item to the source control server.

1. Click the file or folder in the File View.
2. On the **Project** menu, point to **Source Control** and click **Add to source Control**. The file selection dialog opens.
3. Select the files you want to add and press **OK**.

Note

If you selected to add the project file, all files in the project will be listed in the file selection dialog.

Note

*The underlying source control provider may impose restrictions on the files added to source control that can cause the "**Add to source control**" operation to fail. Common restrictions relate for example to the file location, file names and user privileges. Refer to the documentation of your source control provider for detailed information.*

Remove from source control

This command allows you to remove files and folders from the source control server. You cannot remove entire projects or solutions with this command. See your source control documentation for further instructions.

1. Click the file or folder in the File View.
2. On the **Project** menu, point to **Source Control** and click **Remove from source Control**. The file selection dialog opens.
3. Select the files you want to remove and press **OK**.

Show history

A configuration management tool keeps a record for all versions of items that you have added to the source control server. This command allows you to list the history record for one file at a time.

1. Click the file or folder in the File View.
2. On the **Project** menu, point to **Source Control** and click **Show History**.

Show differences

This command allows you to show the differences between your local copy of an item and the latest version on the source control server. This command can only be applied to one file at a time.

1. Click the file or folder in the File View.
2. On the **Project** menu, point to **Source Control** and click **Show Differences**.

Source control properties

This command displays the properties of the selected item. The dialog that opens is tool dependent.

1. Click the file or folder in the File View.
2. On the **Project** menu, point to **Source Control** and click **Source Control Properties**. Your configuration management tool displays the properties of the item.

Refresh status

This command updates the status of the selected items from your configuration management tool. This is a useful command if you have done operations on the files directly in your configuration management tool. If you refresh the project file, (*.ttp), that status of all files in the project will be refreshed.

1. Click the file or folder in the File View.
2. On the **Project** menu, point to **Source Control** and click **Refresh Status**.

Execute CM tool

This command invokes the Configuration Management (CM) tool that is connected to IBM Rational Systems Tester. The CM tool that is connected is dependent on the settings of the registry key:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\SourceCodeControlProvider]
```

1. Click the file or folder in the File View.
2. On the **Project** menu, point to **Source Control** and click **Source Control**.

Import module

This command allows you to find files that are stored on your source control server but are not part of the project you are working with. The files that you find are added to your local computer and inserted in your project.

1. Click the file or folder in the File View.
2. On the **Project** menu, point to **Source Control** and click **Import from Source Control**.

Note

Import from Source Control *is not used with ClearCase.*

Requirements Management (Windows)

IBM Rational DOORS is a state-of-the-art requirements management tool ensuring traceability through the overall development process.

The DOORS integration reduces the amount of work and provides a consistent user interface and behavior.

The key features of the DOORS integration are:

- make information in IBM Rational Systems Tester visible in DOORS
- view and manage DOORS formal modules in IBM Rational Systems Tester
- view and manage DOORS links in IBM Rational Systems Tester

One goal of this integration is to let each user work with either IBM Rational Systems Tester or DOORS, without the need to learn the other tool, or to have both tools running at the same time. The following chapters focus on the use of the integration from the IBM Rational Systems Tester side.

It is assumed that you have a basic knowledge of DOORS. Please study the DOORS User Manual if you need more information about using DOORS.

Note

This integration need to be installed via a separate installer.

Launching DOORS

To ensure that the integration works properly, it is recommended to first start IBM Rational Systems Tester, then launch DOORS from IBM Rational Systems Tester. This is especially important in a multiuser environment such as Citrix, where several instances of e.g. DOORS can be running at the same time on one machine. Always launching DOORS from IBM Rational Systems Tester guarantees that the tools will be able to communicate correctly with each other, in both directions.

It is also possible to first start DOORS, and then launch IBM Rational Systems Tester from DOORS. This will be done implicitly by DOORS for a number of operations, e.g. “Commit to IBM Rational Systems Tester”.

Update/Commit a DOORS module after Archive/Restore

If a DOORS module has been archived and then restored, and possibly also moved from one DOORS database to another, the references in IBM Rational Systems Tester to this module have to be updated (and vice versa). This is accomplished by using either the **Commit** or the **Update** command. The DOORS database will be searched, and if a module which matches the information in IBM Rational Systems Tester is found, all references in IBM Rational Systems Tester will be updated to reflect the new location of the DOORS module.

See also

"Recommendation for Windows users" on page 33

"DOORS Integration Menus (Windows)" on page 797

Managing IBM Rational Systems Tester Information in DOORS (Windows)

Export a DOORS module

Surrogate modules

When a IBM Rational Systems Tester model is exported, a DOORS formal module is created. However, as the original objects of this module are not DOORS objects, the module is called "surrogate".

Export

When exporting to DOORS, two steps are necessary to export data from IBM Rational Systems Tester:

1. Prepare the data to export
2. Perform the export.

Note

*The following describes exporting TTCN-3 models. It is also possible to export a file/project hierarchy from the **File View**.*

Prepare export

A TTCN-3 object hierarchy can be exported from the **Structured View** to a DOORS formal module.

To prepare the export of a TTCN-3 model sub-hierarchy to DOORS, perform the following steps:

1. Select the item you want to be the root of your DOORS formal module in the **Structured View** (or the **File View** if exporting a project hierarchy).
2. Right-click on this item, select **DOORS** and then **Prepare Export** from the shortcut menu.
3. The **DOORS** tab will be shown, containing a view of the exported hierarchy.

To revert, choose **Unexport** from the same menu.

Note

You may export the whole test suite structure by choosing "Prepare Export" for the child of a root (project) element in the Structured View. In this case all TTCN-3 data in later added modules and physical files will be automatically included into set of exported objects.

Export a module to DOORS as a surrogate

To create a surrogate module in DOORS, follow these steps:

1. Select the module in the DOORS tab.
2. Right-click, and from the shortcut menu select **Commit to DOORS...**. If DOORS is not already running, it will be started now. After providing your login information, you can proceed with the export.
3. The **Export Module** dialog allows setting the parameters for the surrogate DOORS module.
4. Type the **name** of the surrogate module to create in DOORS.
5. Choose the **location** in the DOORS database.
6. Fill in **Object identifiers** data: start number and prefix; or use the default values.
7. Press OK.

IBM Rational Systems Tester now creates a formal module in DOORS, containing objects corresponding to the exported objects in IBM Rational Systems Tester.

Unexport of a DOORS module

This command will remove the relationship between a model in IBM Rational Systems Tester and the corresponding formal module in DOORS. After this, it will no longer be possible to **Commit to DOORS** for the module. The exported module in the DOORS tab in the workspace will also disappear. However, the corresponding formal module will still be present in DOORS.

The command is activated with the following steps:

1. Select the root object of the exported model in the workspace window.
2. Right-click on the item, from the shortcut menu select **DOORS** and then **Unexport**.

Committing changes from IBM Rational Systems Tester to DOORS

Described below is how to commit changes made to an exported TTCN-3 model:

1. Select the exported TTCN-3 model in the **DOORS** tab.
2. Right-click and select **Commit to DOORS...** from the shortcut menu.

The surrogate module in DOORS should not be edited. Changes should always be made in IBM Rational Systems Tester and propagated to DOORS.

Update/Commit on Open/Save

When this option is enabled, each time the IBM Rational Systems Tester workspace is saved, all changes to an exported model will be committed to the corresponding DOORS surrogate module. If DOORS is not already running, it will be started.

Propagate test case verdicts to DOORS

When tests are executed under the control of IBM Rational Systems Tester GUI test management information about test case verdicts is persistently stored in the project. For each defined test case this information includes:

- Total number of test case executions performed
- Total number of passed executions
- Verdict from last execution

If test case is included into exported structure then above-mentioned data is propagated to DOORS.

Note

When test case is executed under real-time debugger verdicts are not considered

Attributes created for exported definitions

Each TTCN-3 definition exported to DOORS is augmented with a set of attributes

- ***TTCN3 Kind*** - object kind (constant, function, testcase, etc)
- ***TTCN3 Location*** - slash-separated path to object starting from the defining module (e.g. "MyModule/MyTestCase")
- ***TTCN3 File Path*** - path to TTCN-3 file name (e.g. c:\MyProjects\MyTest.ttcn)
- ***TTCN3 Definition*** - TTCN-3 definition as defined in the test suite. Definition is given without "body" i.e., function definition doesn't show function body, template definition doesn't show constraint, constant definition doesn't show assigned value, etc)
- ***TTCN3 Display Name*** - a TTCN-3 object name as displayed in the structured view
- ***TTCN3 TotalExecutions*** - total number of test case executions (applicable only to test cases)
- ***TTCN3 TotalPassed*** - total number of test cases executions ended with "pass" verdict (applicable only to test cases)
- ***TTCN3 LastVerdict*** - verdict of the last execution (applicable only to test cases)

Managing DOORS Modules in IBM Rational Systems Tester (Windows)

It is possible to import a DOORS formal module into IBM Rational Systems Tester, containing:

- all the objects of the formal module
- all the outgoing links from the objects of this formal module to all the other formal modules of the DOORS database

- a set of predefined attributes for the objects

The formal module will be included as a file in a IBM Rational Systems Tester project, representing a snapshot of the DOORS formal module. OLE objects and pictures in the DOORS formal module are currently not supported by IBM Rational Systems Tester, only plain text.

Even if you, as a DOORS user, have full access control to the objects in this module, you will not be able to modify them in IBM Rational Systems Tester. The requirements they contain are shown to you:

- for information purposes
- to serve as link ends of the DOORS links that you may create in IBM Rational Systems Tester, for example from (or to) TTCN-3 elements

Import a DOORS formal module

To import a DOORS module into your current project, do the following:

1. Select the Project menu, Add To Project and then Components…
2. Choose the component “DOORS Imported module file” and click Insert.

   If DOORS is not already running, it will be started now. After providing your login information, you should be connected to DOORS and ready to import a formal module. The next page of the DOORS wizard, the **Module selection** dialog, is displayed.
3. Select the formal module you wish to import and click Next.

   The **Choose view, baseline and destination location** page of the DOORS wizard is displayed. Select the DOORS view (default: Standard view) and baseline (default: [None]) which should be used for the import.

   You can also choose the name and the path of the file which will contain the information from the imported DOORS module. By default, the name of the DOORS module and the path of the current IBM Rational Systems Tester project file are provided.
4. Check the information in the dialog, change if needed.
5. Click Finish.

   The information in the DOORS module will now be shown in IBM Rational Systems Tester, in a new workspace tab called DOORS, and a new DOORS Imported Module (.dim) file should appear in the **File View** of your project.

At this point, you have successfully imported a DOORS formal module, with its objects and its outgoing links.

You can also import a DOORS formal module to your active project by selecting the **Project** menu, **Import** and then **Import from DOORS**… This is equivalent to the operation described above.

Note that if you choose to import a DOORS view, the DOORS formal module will be displayed during the import.

View a DOORS module

There is a workspace tab labeled **DOORS** which shows a hierarchical view of the objects in the DOORS modules present in a project.

IBM Rational Systems Tester offers you the possibility to get an overview of the requirements of a DOORS formal module. To open the DOORS view:

1. Go to the **DOORS** workspace tab.
2. Select a formal module, double-click the formal module or object name.

In this view it is possible to see the full text of the requirement.

Locate a DOORS object from IBM Rational Systems Tester

The IBM Rational Systems Tester **Locate** command can be used to view the corresponding DOORS object in the current or baseline version of the DOORS module, by following these steps:

1. Select the DOORS object (for example in the DOORS workspace tab).
2. Right-click, from the shortcut menu select **DOORS** and then **Locate**.

IBM Rational Systems Tester then launches DOORS. After logging in, the surrogate formal module is opened and the corresponding object is selected.

Update a DOORS module

When a DOORS module has been imported to IBM Rational Systems Tester, any changes made in the DOORS module will not be visible in IBM Rational Systems Tester until you do an update from DOORS:

1. To update an imported DOORS module, first select it in the **DOORS** workspace tab.
2. Right-click and select **Update from DOORS...** from the shortcut menu.

The DOORS tab and the DOORS view in IBM Rational Systems Tester will now reflect the changes done in DOORS; e.g. deleted, added, or modified objects.

Note

This option is not available for baselined DOORS modules, as these are always read-only.

Committing changes from IBM Rational Systems Tester to DOORS

Committing an imported module to DOORS will not update the incoming links, as the DOORS view in IBM Rational Systems Tester is read-only.

Update/Commit on Open/Save

When this option is enabled, each time a IBM Rational Systems Tester workspace is opened, modules imported from DOORS will be updated with all changes made in the corresponding formal modules in DOORS. If DOORS is not already running, it will be started.

Note

This option is not available for baselined DOORS modules, as these are always read-only.

Changing view or baseline for an imported module

By selecting the shortcut menu command **Change View/Baseline...** for an imported DOORS module, the currently used view and baseline will be displayed in a dialog. If these settings are changed, the imported module will be updated accordingly.

Removing a DOORS module from a project

The shortcut menu command **Remove** can be used to remove an imported DOORS module from the DOORS view in IBM Rational Systems Tester. Doing so will delete the DOORS Imported Module (.dim) file from the file system and from the project file, and it will remove all links to/from the imported DOORS module.

View synchronization information

You can select a DOORS module and open the Properties dialog using the shortcut menu (or ALT+ENTER). This dialog contains the following information regarding the module:

Integration

This tab displays information about the synchronized formal module.

- **Name** is the name of the DOORS formal module.
- **Location** is the identification number of the DOORS formal module.
- **Type** indicates if the module is imported or exported.
- **Last update** indicates the date when the last synchronization occurred.

DOORS toolbar

It is possible to perform the previously described operations by using the buttons of the DOORS toolbar:

- **Start DOORS**: this starts DOORS if it is not already started. After providing the login information, IBM Rational Systems Tester will be connected to DOORS.
- **Import a DOORS formal module**: this will launch the DOORS Import Formal Module wizard.
- **Locate an element in DOORS**: this starts DOORS and selects a given object into DOORS.
- **Export a hierarchy to DOORS**: this is a shortcut that combines the **Prepare Export** and **Commit to DOORS** commands.

Traceability Links (Windows)

It is possible to manage DOORS links with IBM Rational Systems Tester. This section presents the basics you have to know to create and delete links in IBM Rational Systems Tester, and navigate the links.

Incoming and outgoing links

A link has a source and a target. For the source, the link is called an outgoing link and for the target it is called an incoming link. So, if you create a link, you will see that there is a link going to the target, but also coming from the source.

Link sets and link modules

In DOORS, you must have link sets and link modules to be able to create links. This is not the case in IBM Rational Systems Tester. The link is created and stored with its source, so you must have the correct access rights. In Example 28 on page 793 it will be shown how DOORS Link modules are created to export IBM Rational Systems Tester links to DOORS. Link sets are implicit in IBM Rational Systems Tester, but if you export your links to DOORS they will appear in the correct link set.

Link kind

IBM Rational Systems Tester is able to manage other links than traceability (DOORS) links, therefore you must always specify a link kind. The available link kinds depend on what is installed on your system. If the DOORS integrations is properly installed, Traceability links, Dependency links and Hyperlinks are available by default.

Missing Targets

When browsing for a link (in or out) in DOORS, the link always reaches its target if it is in the same database as the source. However, in IBM Rational Systems Tester, depending on the currently active project in your workspace, the link target can be missing.

If the missing target is a DOORS object, IBM Rational Systems Tester automatically opens DOORS to show you the actual target. If the missing target refers to a IBM Rational Systems Tester object, only the message that the project is not loaded will be issued.

Note

The link source can not be missing, as the source is the carrier of the link information.

See also

Working with Links

Link operations

Link commands can be operated from:

- The link menu
- The link toolbar
- The shortcut menu which has a links section containing a subset of the link commands

Link commands

The following commands are available for links:

Start Link (CTRL+K)

Start Many Links

Make Link from Start (CTRL+L)

Display Outgoing

This command will produce a list in the link tab of the Output window of the links going out from the selected entity.

Display Incoming

This command will produce a list in the link tab of the Output window of the links coming in to the selected entity.

Edit...

Options...

Current Link Kind list

Before issuing any link commands for DOORS links, make sure that you have selected **Traceability link** in the **Current Link Kind** list. This list is found in the link toolbar.

Link options

IBM Rational Systems Tester offers you the possibility to customize link creation behavior. You can change the link options via the Tools menu by selecting Options, or clicking on the Display Link Options toolbar button.

Active link end is an active target, not an active source

This option is only effective when you create multiple links. If this option is disabled (which is the default), links will be created from your active link end to the other models. If this option is enabled, the links will be created in the reverse direction, i.e. from the other models to your active link end.

Show link indicators

If this option is enabled, IBM Rational Systems Tester will show the link markers.

Link creation

Creating a link

To create a link, simply select the object which should be the source of the link, and choose the command Start Link (CTRL+K). Then, select the object that you want to be the target for the link, and choose the Make Link from Start (CTRL+L) command.

Creating multiple links from one source

If you have more than one link to create from the same source (or to the same target), you can use the Start Many Links command.

1. Set the type of links you will make to your active object, incoming or outgoing, by specifying the option: Active link end is an active target, not an active source.
2. Select your active link end.

3. Choose the Start Many Links command.
4. Select your other link ends and create each link by selecting the **Make Link from Start** command.
5. Stop the automatic creation by choosing the **Clear Start Many Links** command, or by pressing the corresponding quick button.

Link messages

Whenever a link is created using any of the above methods, a message in the Messages output tab indicates that a link has been created.

Example 27: Message output from link creation ––––––––––––––––––––

Traceability link created from 1 <start> to 2 <next>

––

This message is especially useful when making links in the automatic mode.

Delete a link

To delete a link, use the Links dialog.

Note

Links in baselined DOORS modules can not be changed, as these are always read-only.

Viewing links

IBM Rational Systems Tester shows the links coming from or going to a given object in three different ways:

- Link markers
- Context Link menu
- Links dialog

Link markers

Objects with links going to another object is displayed with a marker, also when the object has incoming links to itself.

For traceability links, the markers used are the red and orange arrows familiar to DOORS users.

Context Link menu

The link markers only give an indication of existing links. To find out which objects are linked, right-click the link marker to see a menu displaying the link kind, and in a sub menu, the linked objects. The same information is also available in the **Links** shortcut menu for an object, in the sub menus **Incoming/Outgoing**.

Links dialog

A list of all objects linked to a given object can be viewed in the **Links** dialog. Select an object and choose **Edit...** (from the Links menu or the Links toolbar).

The Links dialog shows all the objects connected with a selected object. An object which is not loaded into IBM Rational Systems Tester is referred to by its URL instead of its name. It is possible to change the direction to see either objects that are targets or sources of the links. It is also possible to delete a link by selecting the corresponding object, and then press the Delete key (or use the right-click menu choice “Delete link”).

Navigation

To navigate to a link endpoint, right-click the link marker to see a menu displaying the link kind; then, in the sub menu, select a link endpoint. Or, use the Context Link menu. The effect of navigating a link is to mark the link end as selected. Navigation can be done both to the link sources and the link targets of a given object.

There are three situations that can be identified when you want to navigate to a link target (a link source is always available):

- The target of the link is loaded and directly reachable
- The target of the link is not loaded, and it is in an external (un-launched) tool
- The target of the link is not loaded, and it is not reachable

If the target is a DOORS object, IBM Rational Systems Tester will automatically launch DOORS (prompting you to provide the login information) and select the target object.

Synchronizing links

The power of the integration between DOORS and IBM Rational Systems Tester comes with the possibility to work with the links with both tools. It is possible to export links created in IBM Rational Systems Tester to DOORS, and import links created in DOORS.

Links where the source is a DOORS object should be made in DOORS, and then synchronized to IBM Rational Systems Tester. Trying to make a link *from* an object in the DOORS view in IBM Rational Systems Tester could lead to problems if you do not have the correct access rights to the corresponding DOORS requirements module. Of course, making links *to* an object in the DOORS view in IBM Rational Systems Tester will not present this problem, as the DOORS requirements module will not be affected when synchronizing these links.

If there is a link between a DOORS object within IBM Rational Systems Tester and a IBM Rational Systems Tester object, then to be able to export and view this link in DOORS, the IBM Rational Systems Tester model must be exported to DOORS.

Important!

All links are fully synchronized between IBM Rational Systems Tester and DOORS. It doesn't matter which tool a link was originally created in. It is therefore important that all links are present in one tool before committing to the other. If not, links will be deleted.

Links to requirements not represented in IBM Rational Systems Tester are imported and set up symbolically.

Note

The synchronization of links handles only outgoing links, meaning that if you commit a module from IBM Rational Systems Tester to DOORS, all links which have their source in that module (indicated by a red arrow pointing to the right) will be exported to DOORS, but not any links which have their target in that module. These links can be exported by finding the source module, and then do "Commit to DOORS". Similarly, if you do "Update from IBM Rational Systems Tester"in DOORS, this is equivalent to "Commit to DOORS" in IBM Rational Systems Tester - it is the same command which is executed in both cases.

Example 28: Committing links ––––––––––––––––––––––––––––––––––

Synchronizing the DOORS formal module that contains the source objects will also synchronize the links. IBM Rational Systems Tester will automatically detect that the links have link ends in a DOORS module and will synchronize the links.

Links in IBM Rational Systems Tester can also be exported to a DOORS database.

Consider this scenario: Export a formal module containing a TTCN-3 model to DOORS. Import a DOORS formal module and make some links to the objects, see Figure 4 on page 793.

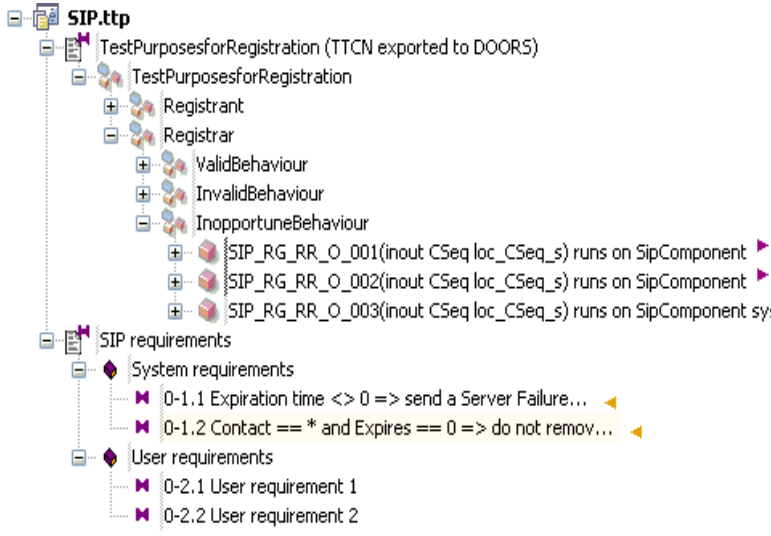

Figure 4: IBM Rational Systems Tester with a model exported to DOORS, and an imported DOORS formal module

Commit the changes in the previously exported model

After exporting the model “SIP.ttp”, you can commit the changes to DOORS, and the corresponding links will be created in DOORS.

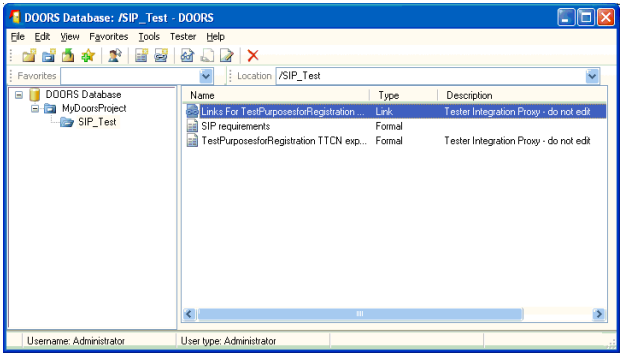

Figure 5: DOORS database view

The database view (in DOORS) will now show the formal module and the link module containing the link created in IBM Rational Systems Tester, see Figure 5 on page 794.

It is then possible to open the formal modules (in DOORS) to check their contents and that the links have been created correctly.

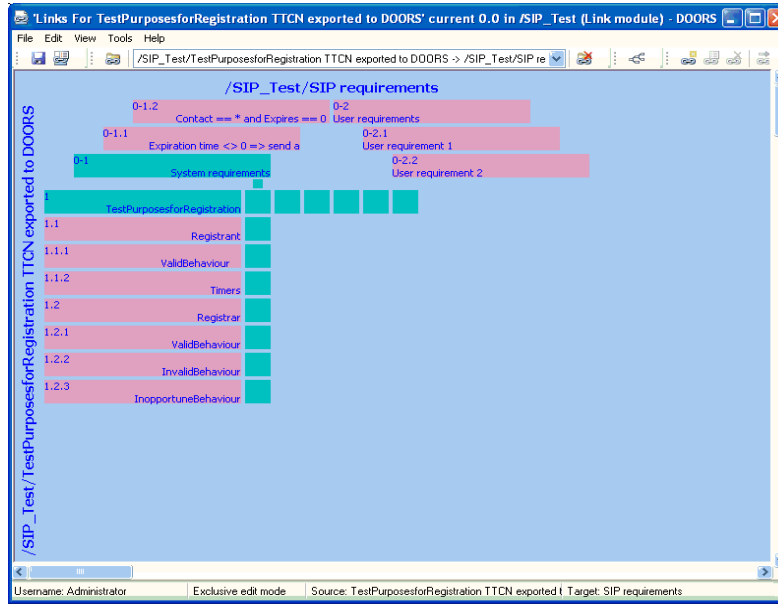

Figure 6: DOORS link module

Open the Link module (in DOORS) which will show the link set implicitly created by IBM Rational Systems Tester, see Figure 6 on page 795.

Example 29: Updating links ----------

IBM Rational Systems Tester can also import links created within DOORS. Consider the previously exported model. Make a link from it to the formal module using DOORS.

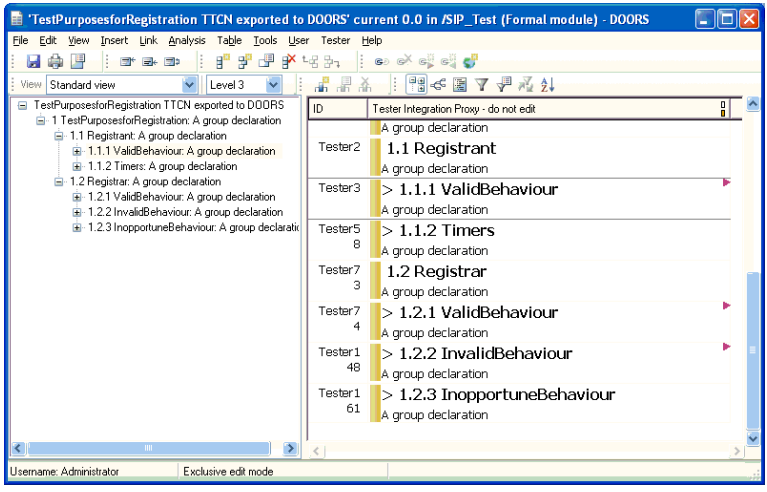

Figure 7: New link created in DOORS

Go back to IBM Rational Systems Tester and update the changes made in DOORS.

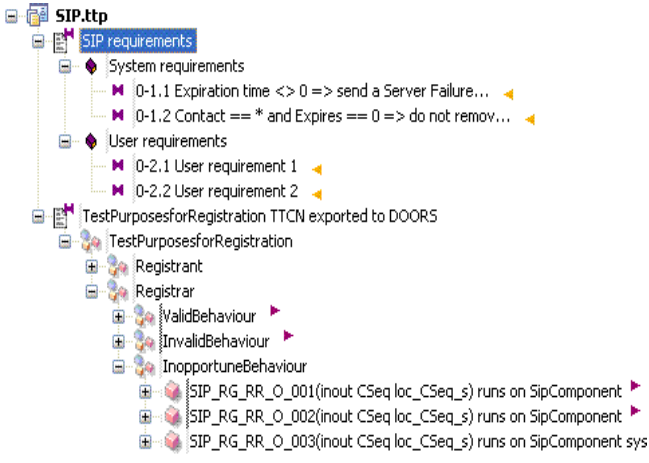

Figure 8: IBM Rational Systems Tester with DOORS module and updated link

The link is now correctly imported into IBM Rational Systems Tester.

DOORS Integration Menus (Windows)

In the Rational Systems Tester sub-menu of DOORS, you can find the following commands:

Available commands in database explorer menus

Create TTCN-3 Test Suite...

This command will start up a wizard with the objective of creating a new project in IBM Rational Systems Tester.

You will be prompted for the project name and location. Once this has been gathered, IBM Rational Systems Tester will be started and a new project will be created.

IBM Rational Systems Tester will then export this information to a DOORS surrogate module.

Open in Rational Systems Tester

This option is only enabled if there is a currently selected module in the Database Explorer.

For modules containing surrogate information, a IBM Rational Systems Tester client will be started with that particular workspace loaded.

It is not possible to open modules containing requirement information (i.e. not surrogate modules) in IBM Rational Systems Tester.

Start Rational Systems Tester

This option will start a IBM Rational Systems Tester client with no particular workspace loaded.

Help

This option opens the help file containing information about the integration.

About Integration...

This option will display a dialog with information about the integration, e.g. copyright and version number.

Shortcut menu for database explorer

In the database explorer, the following commands are available in the shortcut menu (under the Rational Systems Tester submenu):

Create TTCN-3 Test Suite...

Open in Rational Systems Tester

Start Rational Systems Tester

Surrogate module menus

The commands described below are found in the **Rational Systems Tester** menu.

Update from Rational Systems Tester

This option is only enabled for modules containing surrogate information. It will make a call to IBM Rational Systems Tester in order to begin an export operation. This will in turn cause the IBM Rational Systems Tester client to be started if it is not already running.

Commit Links to Rational Systems Tester

This command will propagate outgoing links from the current module to the corresponding IBM Rational Systems Tester model. It will make a call to IBM Rational Systems Tester to begin an import operation. This will cause the IBM Rational Systems Tester client to be started if it is not already running.

Locate

This option is only enabled for modules containing surrogate information.

This will cause the corresponding IBM Rational Systems Tester item for the selected surrogate entry to be selected. The IBM Rational Systems Tester client will be started if it is not already running.

Filter (Sub-Menu) (TTCN-3 only)

The Filter function provides an important aspect of the integration allowing traceability between requirements and design.

Identify Design Elements Not Justified by Requirements

This option is only enabled for modules containing surrogate information.

It will query all out-links in the object hierarchy and filter on out-links that do not have a corresponding entry in another (assumed to be a requirements) module.

Note that if the description for the IBM Rational Systems Tester surrogate module is changed, this function will not work as expected.

Identify Design Elements by TTCN-3 Kind (TTCN-3 only)

This option will present the user with a list of all TTCN-3 kinds present in the surrogate module. A filter will then be applied so that only objects with these values will be displayed.

Open in Rational Systems Tester

Help

About Integration...

Shortcut menu for surrogate modules

In a formal module, when a requirement is selected, the following commands will be available in the shortcut menu (under the Rational Systems Tester submenu):

Locate

Requirements module menus

The commands described below are found in the **Rational Systems Tester** menu.

Commit to Rational Systems Tester

If the requirements module already has been exported to IBM Rational Systems Tester, the corresponding IBM Rational Systems Tester project will be used. Otherwise, you will be prompted for the name and path to a IBM Rational Systems Tester project which this module should be imported to.

A call to IBM Rational Systems Tester will be made for it to begin an import operation. This will in turn cause the IBM Rational Systems Tester client to be started if it is not already running.

Update Links from Rational Systems Tester

This command will propagate the outgoing links from the IBM Rational Systems Tester model that corresponds to the current DOORS module into DOORS. It will make a call to IBM Rational Systems Tester to begin an export operation.This will cause the IBM Rational Systems Tester client to be started if it is not already running.

Open Linked Surrogate Item in Rational Systems Tester

This option allows you to open linked surrogate entries in IBM Rational Systems Tester.

If in-links exist from multiple surrogate entries then a dialog will be displayed allowing a specific selection to be made.

Link Requirement to Selected Item in Rational Systems Tester

This will create an out-link from the current object to the corresponding entry in the surrogate module for the currently selected object in the IBM Rational Systems Tester workspace.

Filter (Sub-Menu) (TTCN-3 only)

Identify Requirements Not Addressed by Design Elements

This will query all in-links in the object hierarchy and filter on in-links that do not have a corresponding entry in any surrogate module.

When filtering all ancestors of accepted objects will be shown. This will preserve the heading hierarchy and provide help to place each requirement.

Note that if the description for the IBM Rational Systems Tester surrogate module is changed, this function will not work as expected.

Start Rational Systems Tester

Help

About Integration...

Shortcut menu for requirements modules

In a requirements module, when a requirement is selected, the following commands will be available in the shortcut menu (under the Rational Systems Tester submenu):

Open Linked Surrogate Item in Rational Systems Tester

Link Requirement to Selected Item in Rational Systems Tester

14

Printing

This chapter describes different ways of printing a diagram and how to change print settings.

Adding and Removing Printers (UNIX)

The MainWin Control Panel allows you to make your printers available in the IBM Rational Systems Tester print dialog.

To open the MainWin Control Panel:

- From the terminal window, type:
  `<installation directory>/bin/mwcontrol`

The control panel opens.

To add a printer:

1. When the control panel is open, click **Printers**. The Printer window opens.
2. Click **Add New Printer** and follow the instructions in the add printer wizard that opens.
3. When you have completed the wizard, close the printer dialog and the Control Panel.
4. Restart IBM Rational Systems Tester.

To remove a printer:

1. When the control panel is open, click **Printers**. The Printer window opens.
2. Right-click the printer you want to remove and click **Delete**.
3. Close the printer dialog and the Control Panel.
4. Restart IBM Rational Systems Tester.

Printing Diagrams

There are several ways of printing diagrams. You can print single diagrams from:

- The diagram itself.
- The Model View.
- The Print Manager.
- The diagram preview window.

You can print multiple diagrams from:

- The Model View.
- The Print Manager.

Note

Using a white/transparent background for an Icon image may result in a black background when printing. This is related to a Windows postscript driver PS level 2. Changing to PS level 1 may remove the situation. Using a colored background or frame will also prevent this.

Adding and setting up printers (UNIX only)

The procedure how to add and set up printers for use from IBM Rational Systems Tester on UNIX hosts is done with the **MainWin Control Panel**, described in detail in the Installation Guide.

Print settings

To change print settings:

1. On the **File** menu, select **Print Setup**.
2. In the Print Setup dialog, select printer, paper size and other properties allowed for the selected printer. The paper size and orientation will be used to determine the default diagram size in the editors.

3. Click **OK**.

1. Print files

To print a file:

1. Open the file that you want to print, and place the cursor somewhere in the text.
2. On the **File** menu, click **Print** or click the print icon in the toolbar.
3. In the Print dialog, change settings according to your preferences.
4. Click **OK**.

Select diagrams to be printed

All diagrams in your model are available in the Model View. The Print Manager allows you to select which diagrams to print. To open the Print Manager, click **Print Manager**, on the **File** menu.

The diagrams that are included in the container that is active in the Model View, are listed in the Print Manager. Use the **Track Selection** button if you want to change container in the Model View. If the button is not pressed in, the contents in the Print Manager is locked to the first selection you made.

Figure 9: Track selection button, when not selected

You can also decide which type of diagrams you want to print by checking or clearing the diagram type check boxes in the **Filter** area.

You can calculate the number of pages to print by clicking **Pages** in the **Print** window.

Preview of diagrams

To get a preview of a diagram:

1. Select the diagram in the Model View.
2. Select **Print Preview** on the **File** menu. A preview of the diagram is displayed.
   – You can scroll to other diagrams by using the **Next Page** and **Previous Page** buttons.

Print a single diagram

To print a single diagram from the diagram itself:

1. Open the diagram.
2. Select **Print** on the **File** menu. The standard print dialog is displayed.

To print a single diagram from the Model View:

1. Select the diagram in the Model View.
2. Right-click the diagram and select **Print**. The standard print dialog is displayed.

To print a single diagram from the Print window in the Print Manager:

1. Select the diagram in the Model View. The diagram icon is displayed in the **Selection** area.
2. Click the **Print** button. The standard print dialog is displayed.

To print a single diagram from the preview window:

- Select **Print**. The standard print dialog is displayed

Print multiple diagrams

To print multiple diagrams from the Model View:

1. Select the diagrams in the Model View.
2. Select **Print Manager** on the **File** menu. The **Print** window is displayed.
3. Click the **Print** button or select **Print Preview** on the **File** menu and then **Print**. The standard print dialog is displayed.

You can print diagrams of the same type(s) at the same time if you use the Print window and the Filter functionality.

To print multiple diagrams from the Print window:

1. Select **Print Manager** on the **File** menu. The **Print** window is displayed.
2. In the Model View, select the diagram(s) you want to print. The diagrams and page numbers for the diagram type(s) you selected are displayed in the **Selection** area.
3. Click the **Print** button or select **Print Preview** on the **File** menu and then **Print**. The standard print dialog is displayed.

15

Internationalization Support

This section describes the internationalization support in IBM Rational Systems Tester. The main focus of this document is Chinese, Japanese and Korean (CJK) language handling.

Supported environments

This section describes specific information for Internationalization support of system environments. The information not described in this section is common through all languages. Please refer to the installation guide for general information.

Supported platforms

The internationalization support in IBM Rational Systems Tester is available for Windows 2000 and XP. It is assumed that you use a local version of Windows and set the locale to use your local language.

Configuration Management

IBM Rational Systems Tester does not support CJK environments beyond limitations of each configuration management tool for CJK support.

IME (Input Method Editor)

Default IMEs bundled in Windows are supported. Using supported IME, you can enter your local characters inline.

Font settings

By selecting the correct font for your language, your language is displayed correctly.

1. Select **Tools** and then **Options** from the IBM Rational Systems Tester menu bar**.**
2. Select **Format** tab.
3. Choose **Category** and specify font type
   - **Dialog fixed** : the font type setting for dialogs using a fixed width font.
   - **Developer diagram symbol font**: the font type setting for other symbols and diagrams.
   - **Report Windows**: the font type setting for tabs in the Output window.
   - **Output Windows**: the font type setting for Message, Model Verifier and Script tabs in the Output window.
   - **Tcl Files**: the font type setting for Tcl and text files opened in IBM Rational Systems Tester.
   - **C/C++ Header/Source**: the font type setting for C/C++ header and source files opened in IBM Rational Systems Tester.

Note

The instructions presented below should be performed before you start to create elements in your diagrams.

There is also fonts settings for diagrams elements.

1. Select **Tools** and then **Options** from the Tau menu bar.
2. Select **Font settings** tab.
3. Specify font types. See "Font settings" on page 825.

Note

You can also change the font style and size for each element from the Diagram element properties toolbar.

Modeling with CJK characters

IBM Rational Systems Tester supports modeling with CJK characters. You can use CJK characters for

- names of all elements
- comments
- Charstring literals.

You can type CJK characters in the same way as English characters. No special operation is needed to draw models with CJK characters.

Code generation with CJK characters

Element names in the model are used as names of identifiers in generated C/C++ files. However, in C/C++ grammar, CJK characters are illegal for identifier names. So, IBM Rational Systems Tester have a mechanism to provide legal names for C/C++ code and it can be done in two ways.

Automatic UTF-16 naming

If CJK characters are used in an element name, IBM Rational Systems Tester provides the legal name in generated C/C++ code. The provided legal name consists of a UTF-16 big endian encoded hexadecimal string that is prefixed by `tau_` and suffixed by `_tau`

Example 30: UTF-16 encoded name

MALMÖ is encoded as

```
tau_004D0041004C004D00D6_tau
```

Using ansiName stereotype

If you want to use CJK characters for element names and you want to specify legal name for generated C/++ code, you can use the stereotype ansiName.

1. From the **Tools** menu select **Customize**. Go to the Add-Ins tab.
2. Check the **Internationalization** add-in to load ansiName stereotype in the current project. (You need to activate it per project.)
3. Enter the element name in the diagram with CJK characters.

4. In the Model View, right-click the element for which you want to specify a legal name.
5. On the shortcut menu, click **Properties**.
6. From the **Filter** box, select **ansiName**.
7. Type the legal name for the code generation in the **name** field.

When you specify a legal name by ansiName, the specified name is used for C/C++ code.

Names of files and folders used by build tool chain

Names of class symbols and package symbols

The names of class symbols and package symbols are used as generated file and folder names. If CJK characters are used in symbol names, the encoding mechanism described in "Code generation with CJK characters" on page 811 is applied to name the generated folders and files. If you want to specify the file and folder names, then use ansiName stereotypes as in "Using ansiName stereotype" on page 811.

Comments

Comments will not be presented in generated C/C++ files. This applies to both CJK characters, English characters and any other language.

Encode type of files used by build tools

The encode type of the files that are processed by the build tool chain (such as the files holding the intermediate format and the generated C/C++ files) are encoded using the **system locale** encoding. For example, on Japanese Windows, the generated files will be encoded using SHIFT-JIS, so that Microsoft Visual C++ can handle and compile the files.

Therefore, you need to set correct locale for your language to generate files with the correct encoding.

1. Open the Windows Control panel
2. Open the **Regional Settings** dialog and select the **General** tab.
3. Make sure that the selected locale is the correct one.

Handling textual files

Textual files can be opened inside IBM Rational Systems Tester. IBM Rational Systems Tester supports local ANSI encoding and UTF-8 for the textual file. When existing textual files are opened in IBM Rational Systems Tester, IBM Rational Systems Tester saves the files in the original encoding. When the textual file is created in IBM Rational Systems Tester, the file will be saved in UTF-8 by default. You can select encode type from the **Save as** dialog.

Restrictions

- Single byte Japanese Katakana and Japanese characters defined between 0x80 and 0xFF in Shift-JIS are not supported.
- CJK characters are not supported for Project names.
- Using CJK names for messages and message text parameters may result in displaying UTF-16 big endian encoded names instead of original CJK names, when tracing a Model Verifier in Sequence diagrams. In order to find out the original names, decoding must take place in an external application that supports displaying UTF-16 encoded characters.

See also

"Automatic UTF-16 naming" on page 811.

16

Dialog Help

This section lists the help texts that are displayed when you click the help button in dialogs.

The New Wizard

Files tab

This dialog provides the possibility to add new files to your design.

- When adding a file you must specify a file name and a location.
- The file can be added to an existing project. The project must be opened in the File View in order for you to add the file to it.
- The new file is opened in the Desktop.

Projects tab

This dialog provides the possibility to add a new project.

When you add a project, you specify how the project will be used. Depending on your choice, different add-ins will be loaded at start-up, for example:

TTCN Project
No add-ins are loaded. This is a Rational Systems Tester project. The structured view will be available in the workspace window.

Should you later want to change the default add-ins in your project, you can do so using the Add-ins tab (**Tools** menu, select **Customize**).

- When adding a project you must specify a project name and a location.
- The project can be included in the current workspace, or a new workspace can be created for the project.

See also

"Working with Projects" on page 18 in Chapter 1, *Introduction to IBM Rational Systems Tester 3.3*

"Add-ins tab" on page 821

TTCN Projects - page 2

This dialog allows you to set properties for your TTCN-3 project.

- Specify the **file directory** where new files will be created.

- Select if you want to **add a TTCN-3 file** to the project. You must specify the file name.
- If you add a new file to the project, you may also **add an empty module** to the file. In that case, you should also type a module identifier.
- Select **Copy makefile configuration file**, if you want this file added to your project.

See also

"Makefile Generation" on page 79 in Chapter 4, *Creating an ETS*

TTCN Projects - page 3

This dialog confirms your selections. Click **Back** if you want to make any changes.

Workspaces

This dialog provides the possibility to add a new workspace.

- When adding a workspace you must specify a workspace name and a location.
- The new workspace is loaded in the Workspace window.

See also

"Working with Workspaces" on page 16 in Chapter 1, *Introduction to IBM Rational Systems Tester 3.3*

Customize

Commands tab

This tab lists the default menus with toolbar buttons, commands and menus that you can add to a toolbar or menu. It allows you to move, delete or add buttons to your toolbars.

1. In the **Categories** box, click the toolbar name that you want to customize.
2. In the **Buttons** area, drag the item from the dialog on to the toolbar. Click the item first to receive information about the specific item.

3. To remove an item from a toolbar, drag the item from the toolbar on to the dialog.

To add a button to a toolbar:

1. Make sure that the toolbar you want to change is displayed.
2. In the **Categories** box, the available toolbar buttons or items are grouped. Select the category where the toolbar button or item you want to add is located.
3. Click a button or item to receive information about its functionality.
4. Drag the button or item from the **Buttons** area to the toolbar in the user interface.

To delete a button from a toolbar:

1. Make sure that the toolbar you want to change is displayed.
2. Drag the button or item off the toolbar.

When you delete a default button from a toolbar, the button is still available in the Customize dialog box. However, when you delete a toolbar button with a custom appearance, its appearance is permanently lost, although the command is still available (Customize dialog box, Commands tab).

Hint

To save a toolbar button with a custom appearance for later use, create a toolbar for storing unused buttons, move the button to this storage toolbar, and then hide the storage toolbar.

Toolbars tab

This tab lists standard and custom toolbars.

Select or clear the check boxes to display or hide the toolbars. Each toolbar appears either in the default location or in the last location that it is moved to. The menu bar cannot be hidden.

Show Tooltips
Click the check box to enable tooltips to be displayed when the cursor moves over a button or field in the toolbars.

Large Buttons
Click the check box to display larger sized buttons in the toolbars.

Create a new toolbar:

1. Click **New**.
2. In the dialog that opens, type the name of the toolbar. The new toolbar appears in the toolbar area of the interface.
3. From the **Commands** tab, select the items that you want to add to the toolbar.

Restore the default toolbar settings:

1. Click the toolbar in the list.
2. Click **Reset**.

A user-created toolbar cannot be restored.

Delete a user-created toolbar:

1. Click the toolbar in the list.
2. Click **Delete**.

A default toolbar cannot be deleted.

Rename a user-created toolbar:

1. Click the toolbar in the list.
2. In the **Toolbar Name** field, type a new name for the toolbar.
3. Click the toolbar again to save the change.

Create New Toolbar

Type the name of the new custom toolbar. You can use upper or lower case letters, but each name must be unique regardless of case. The name must be unique from other toolbars. If you want to change this name later, you can edit the name in the Toolbar Name box on the Toolbars tab.

Windows layouts

This tab allows you to customize the appearance of the Windows layout. You can save toolbar positions, visibility and location of docked windows.

Save a new layout:

1. Click the New button.
2. Type a name for your layout.
3. Close the window.

To restore a new layout:

1. Click the layout you want to restore.
2. Click Restore.

To delete a layout:

1. Click the layout you want to delete.
2. Click the Delete button.

Tools tab

This tab allows you to add commands in the Tools menu. These commands can be associated with any program that runs on your operating system. The information is saved in a file named Tools.dat in the directory:

```
C:\Documents and Settings\<user>\Application Data\Telelogic\Shared
```

Add a command to the Tools menu:

1. Click the **New (Insert)** button. A blank line, indicated by an empty rectangle, appears in the **Menu Contents** box.
2. Type the name of the command as it will appear in the Tools menu. Press **ENTER** to save the name.
3. In the **Command** field, type the path to the program. You can also use the browse button to locate the program.
4. In the **Arguments** text box, browse or type any arguments to be passed to the program. Use the drop-down arrow next to the Arguments text box to display a menu of arguments.
5. In the **Initial directory** box, browse or type the file directory where the command will be located.
6. If the program is a console program, for instance the Windows command prompt, you can select to have it run in the Output window. Just select the **Use Output Window** check box.

7. Select the **Prompt for Arguments** check box, if you want to be able to change argument each time you want to use the command.
8. Select the **Use OEM format** check box, if you want to the application's output to be in OEM format.
9. Click **OK**. The command appears in the Tools menu.

Additional tasks

- To insert the command in a submenu, separate the menu name and the name with a backslash '\'. For instance, the command Notepad in an editor menu should be typed `editor\Notepad`.
- To insert an access key, type an ampersand '&' before the selected letter in the name.
- Move commands up and down in the menu by using the **Move Up** and **Move Down** buttons.
- To change the name of the command, double-click it and type a new name.

Delete a command in the Tools menu:

1. Click the command in the list.
2. Click the **Delete** button.

Add-ins tab

Add-ins are used to extend the tool functionality. From the Add-ins tab you can load a selection of predefined add-ins.

The different add-ins are optimized for different build situations. This means that you often do not have to change any build settings. When you click an add-in, you will see its usage in the description field.

- To load or unload add-ins, select or clear the check boxes. Close the dialog.
- You can modify the available add-ins. Click the add-in and click **Modify**. The dialog that opens allows you to customize the add-in according to your needs.
- You can also create your own add-ins by clicking **Create**. The dialog that opens allows you to configure the add-in according to your needs.

See also

"Contents and structure of an add-in" on page 1608 in Chapter 45, *Customizing Telelogic Tau*

Options

General

This tab allows you to set general options:

The **Display status bar** allows you to show or hide the status bar that is available at the bottom of the IBM Rational Systems Tester user interface.

When the Output window is closed, information that is normally listed in the different tabs is not displayed. However, when selecting the **Show output window when receiving content** check box, the output window will open automatically when new information is listed, for instance after a manual check.

Connect to the Source Control System at start-up
If you have a source control system installed, you can select this check box. This enables a source control menu and toolbar for interaction with your source control system.

Show advanced option page
Select to display an additional tab with all options listed in a tree structure.

Tabbed documents
Select to open documents in a single window as tabs.

Disable external program launch for these type of file
In this field you can specify the extension of files, that any associated external application is not to be launched when they are opened from within IBM Rational Systems Tester. For instance, if you add the `.htm` extension, `htm` files will not be opened in your web browser if opened from within IBM Rational Systems Tester.

Select the default help context
If there are many Telelogic tools installed, you can choose which help file to use as default by selecting the file in the **Select the default help context** box.

URN Map
Use the **URN Map** (Universal Resource Name) to define shorthand names for file storage locations. For example:

```
home:C:\MyHomeDir;work:C:\MyWorkDir
```

Here "`home`" is shorthand for `C:\MyHomeDir` and "`work`" is shorthand for `C:\MyWorkDir`. Each user may define URNs for his/her environment. These are used by some components for referring to files, bitmaps and other resources.

Save

This tab allows you to set save options in IBM Rational Systems Tester.

Save before running tools
Select to automatically save any unsaved work before an external tool is launched.

Prompt before saving files and projects
Select to be prompted when files and projects are being saved when an editor closes.

Automatic reload of externally modified files
You will by default receive an information message and be prompted to reload an externally modified file. Select this option to automatically reload a file that has been modified in another tool than IBM Rational Systems Tester.

Save project's add-in state in all the loaded projects
This option will let any loaded add-ins become activated for all projects currently loaded.

Auto-backup
Select the **Activate** check box to allow automatic saves of your model in predetermined intervals. Enter the desired number of minutes between the saves, either by typing the number or by clicking the up and down buttons.

Workspace

This tab allows you to set general options for the workspace that you have opened.

Reload last workspace at startup
Select to open the workspace that you were working in the last time IBM Rational Systems Tester is running.

Warn on project file status change
Select to receive a warning if the status of the project file you are working in has been changed to read-only. This will protect you from potentially loosing unsaved work.

Projects default location
When you create new projects, you will receive a suggestion where the project file will be stored. In this text field, type a path, or browse to a folder, where the new projects will be stored.

Format

This tab allows you to format the appearance of text and colors in windows and files.

When you have selected a category, you can select:

- **Font** and **Size** of the text in the file or window.
- Background color and text color for the selected Category. By default, system colors defined in the control panel are used. Clear the **Automatic** check boxes to select text and background colors.

Links

This tab allows you to customize link creation behavior.

Active link end is an active target, not an active source
If this option is off, then when you use automatic creation of links, you will create links from your active link end to the other models. If this option is on, then you will create links to your active link end from the other models.

Automatically create links between modified objects and active link end
If this option is on, then when you select an active link end, all your modifications will be linked to this link end.

Show link indicators
If this option is on, IBM Rational Systems Tester will show the link markers.

Compare/Merge

This tab allows you to change options for compare/merge.

External text compare/merge

Default values for these options are compatible with the textual compare/merge tool of Synergy CM if installed.

External text compare/merge tool path

Path to the external text compare/merge tool that can be used from the review differences dialog.

Command line switches

Depending on what compare/merge usecase it is, the external compare/merge tool will be called with the corresponding command line switches option.

Advanced

The Advanced tab allows you to change the values for any option. Navigate the window to find the option which value you want to change. To change the option value, select the value and click F2.

TTCN Dialogs

Find In Files

Use this dialog when you want to look for an item in any of your files.

- Type the file types you want to look for in the in Files/file types field. Separate multiple file types with a semicolon. For example, if you want to look for your item in HTML files and ASN-1 files you should type: `*.html;*.asn`

- By default, the search result will be presented in a tab in the Output window called Find in Files 1. If you want to select the Output to pane 2 check box, the search result will instead be presented in the tab called Find in Files 2. This feature is useful if you want to search for several items.

See also

"Find and Replace" on page 55

Go To

This dialog allows you to quickly jump to anywhere in your files.

- Select if you want to go to a line or a bookmark. If you have not defined any bookmarks, the list of bookmarks will be empty.
- If you press the pin button before the jump is performed, the Go To dialog does not close after the jump has been performed.

See also

"Bookmarks" on page 61

"Go To" on page 63

Go To Line

This dialog allows you to quickly jump to anywhere in a text file.

- Enter a line number and press OK.

17

Additional Resources

This section list documents that are not part of the help file, but that may help you to extend your knowledge about IBM Rational Systems Tester. Links to useful web resources are also provided.

Links

Telelogic links

IBM Rational Systems Tester

TTCN-3 documents

TTCN-3 Tutorial (PDF)

A basic tutorial that allows you to become familiar with Rational Systems Tester and the TRI integration. The tutorial is available in your installation in:

```
.../locale/en/tutorial.pdf
```

Other links

Borland C/C++

C/C++ dialect supported by the Borland builder.
http://www.borland.com/cbuilder

Cygwin

For information about the contents of various Cygwin versions, see:
http://www.cygwin.com

GNU C/C++

C/C++ dialect supported by the GNU Compiler Collection.
http://www.gnu.org/software/gcc

ITU-T

Formerly CCITT
http://www.itu.int/

Macrovision

For more information about FLEXnet or Macrovision, please see:
http://www.macrovision.com

Microsoft Visual C/C++

C/C++ dialect supported by Microsoft Visual C++:
http://msdn.microsoft.com/visualc

MISRA

The code generated by the AgileC Code Generator is to a large extent compliant with the MISRA coding rules described in the document “MISRA-C:2004 Guidelines for the use of the C language in critical systems” from October 2004. Please see:

http://www.misra.org.uk

OCL

For more information about OCL (Object Constraint Language), see:
http://www.omg.org

OMG

For more information about Object Management Group (OMG), see:
http://www.omg.org

PDF

PDF files are opened and read with Adobe Acrobat Reader:
www.adobe.com

Tcl

For detailed information refer to the Tcl Developer Site
http://tcl.activestate.com/

TTCN-3

The TTCN-3 standard can be downloaded from
http://www.etsi.org

XML

For information about Extensible Markup Language (XML), see:
http://www.w3.org/XML

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z