

Tester

TTCN-3 Tutorial

This edition applies to Telelogic Tester version 3.2 and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright IBM Corporation 1997, 2008.

US Government Users Restricted Rights—Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

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:

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

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, MERCHANTABILITY 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.

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

Licenseses 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.

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 announce-

ments 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.

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.

Copyright license

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. © Copyright IBM Corp. 2000, 2008.

IBM trademarks

IBM, the IBM logo, ibm.com, Telelogic, and Tau are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at www.ibm.com/legal/copytrade.shtml.

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.

HP and HP-UX are registered trademarks of Hewlett-Packard Corporation.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Macrovision and FLEXnet are registered trademarks or trademarks of Macrovision Corporation.

Microsoft, Windows, Windows 2003, Windows XP, Windows Vista and/or other Microsoft products referenced herein are either trademarks or registered trademarks of Microsoft Corporation.

Netscape and Netscape Enterprise Server are registered trademarks of Netscape Communications Corporation in the United States and other countries.

Sun, Sun Microsystems, Solaris, and Java are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Pentium is a trademark of Intel Corporation.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product or service names may be trademarks or service marks of others.

Contacting IBM Rational Software Support

Support and information for Telelogic products is currently being transitioned from the Telelogic Support site to the IBM Rational Software Support site. During this transition phase, your product support location depends on your customer history.

Product support

- If you are a heritage customer, meaning you were a Telelogic customer prior to November 1, 2008, please visit the Tau Support Web site. Telelogic customers will be redirected automatically to the IBM Rational Software Support site after the product information has been migrated.
- If you are a new Rational customer, meaning you did not have Telelogic-licensed products prior to November 1, 2008, please visit the [IBM Rational Software Support site](#).

Before you contact Support, gather the background information that you will need to describe your problem. When describing a problem to an IBM software support specialist, be as specific as possible and include all relevant background information so that the specialist can help you solve the problem efficiently. To save time, know the answers to these questions:

- What software versions were you running when the problem occurred?
- Do you have logs, traces, or messages that are related to the problem?
- Can you reproduce the problem? If so, what steps do you take to reproduce it?
- Is there a workaround for the problem? If so, be prepared to describe the workaround.

Other information

For Rational software product news, events, and other information, visit the [IBM Rational Software Web site](#).

Table of Contents

How to contact Customer Support	iii
Introduction	2
Purpose of this tutorial	2
Terminology	2
Files	3
System overview	4
Goal	4
Abstract Test Suite and System Under Test	4
The Abstract Test Suite	6
Goal	6
TTCN-3 Project creation	6
Add files to the project	6
Analyze the test suite files	7
Test behavior	8
Goal	8
Look at test case TC1a	8
Look at test case TC2s	9
Look at test case TCTimer	10
Communication and Ports	11
Goal	11
Port mappings of TC1a	11
Port mappings of TC2s	12
Integration implementation	13
Goal	13
Code modules	13
Add the integration files to the project	13
In-depth look at TRI TE-SA functions	14
In-depth look at TRI PA-SA functions	16
Build the ETS	18
Goal	18

Table of Contents

Prepare makefile generation	18
Settings in the dialog	18
Compile the test suite	18
Build the ETS	19
Run the ETS	20
Goal	20
Start the SUT	20
Execution switch	20
Run ETS from Tester	20
Run the ETS from command-line	22
Logging	22
Add the log module	23
Rebuild and run the ETS again	23
Module parameters	23
ETS with different module parameter values	24
Food for thought	25
Separate communication channels	25
More PTCs	25
Timer limitations	25

1

Tester TRI Integration

Introduction

Purpose of this tutorial

This tutorial aims at giving the reader a hands-on experience on what is required from TRI integration and how to properly implement one. You are expected to know TTCN-3 to be able to follow the tutorial as well as having read Tester “Technical Integration Documentation” (PDF-file found on distribution CD) and the TRI Specification.

The tutorial will try to answer the following questions:

- What is a TRI integration? Why is it necessary and how do I implement it?
- How do I create a log and plug it into the executable test suite?
- How do I develop and add encoding and decoding functions? Why is it necessary?
- How do I build a complete executable test suite and how do I run it?

Note

The tutorial implementation and this tutorial are made for the Windows platform. It should be easy however, for a skilled programmer to adopt the instructions and modify the implementation for a Unix or other non-Windows platform.

Terminology

Often used terms in this tutorial that you should be familiar with:

ATS	Abstract Test Suite
ETS	Executable Test Suite
Compiler	Tester TTCN-3 Compiler
MTC	Master Test Component
PTC	Parallel Test Component
RTS	Run-Time System
SUT	System Under Test
TRI	TTCN-3 Run-time Interface

Files

In the text there will sometimes be references to the `<tutorial_dir>` directory. This is the example directory that will reside in your installation, typically for Windows this would be:

```
[Tester_Installation_Directory]\examples\ttcn3tutorial
```

System overview

Goal

After this part you should understand the layout of the test system, the system under test (SUT), the code modules involved and how they relate to each other.

Abstract Test Suite and System Under Test

An Abstract Test Suite (ATS) is generally developed with the System Under Test in mind. The SUT has a number of potential communication channels (ports) through which data is transferred (a protocol). In this case the SUT is a separately running process which communicates through a socket. Before the tests are performed the communication is set up between the Executable Test Suite (ETS) and the SUT.

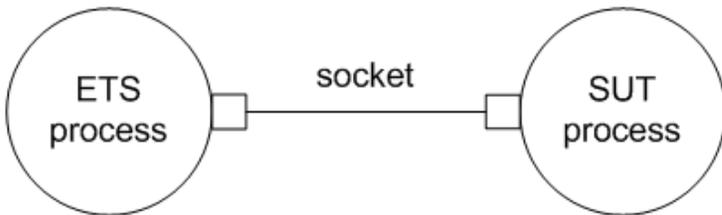


Figure 1: ETS and SUT processes

The ATS is written in TTCN-3 and as always, to be able to compile and link the TTCN-3 code into an executable, an integration (or adaptation) needs to be implemented. In this tutorial, the integration is written according to the TRI specification. The integration is required to handle the ports of the ATS, their mappings to TSI (Test System Interface) ports and the communication of data to the SUT over the socket.

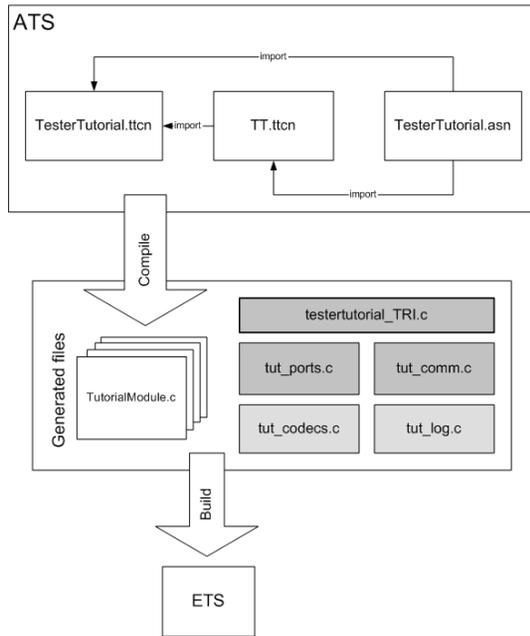


Figure 2: ATS to ETS

The gray modules in [Figure 2 on page 5](#) are all part of the TRI integration implementation. The light gray ones are not strictly defined by TRI but nevertheless needed to complete the integration with encoding and decoding functions (`tut_codec.c`). The log mechanism (`tut_log.c`) is completely optional but added to show its use. The modules will be explained in more detail in the subsequent parts of this tutorial.

The Abstract Test Suite

Goal

In this part you will create and insert files into a TTCN-3 Project in Tester. The analysis tool will also be used on the newly created project.

Note

In Tester there is an option of creating template projects from a set of pre-defined examples included with the tool installation, including the TTCN-3 tutorial example files. For the Tester tutorial, it is not necessary to copy the supplied source files anywhere since they are not supposed to be edited as part of the tutorial. If you choose to open a template project dialog and select the `ttn3tutorial`, the supplied files will be copied to a project folder of your choice. The project and workspace files will be created from the actions described in this document.

TTCN-3 Project creation

Note

When running this Tutorial on Linux platform ensure that `OSTYPE` environment variable is set to "linux" before you start Telelogic Tester.

Start Telelogic Tester. In the **File** menu choose **New** and create a new **TTCN Project** and give it a proper name and location. Keep the **Create new workspace** choice selected.

Note

Ensure that path to project does not contain spaces. ASN.1 compiler may behave incorrectly when it's invoked for files containing spaces in the path.

In the next dialog uncheck the **Add TTCN-3 file to the project** and **Add adapter templates to the project**. The reason for this is that all the necessary ATS files will be added later on. Click **Next** and then **Finish** to finalize the project creation.

Now that you have yourself a new empty project it is time to bring the ATS files into the project.

Add files to the project

In the **Workspace** view (the area on the left side), choose the **File View** by clicking that tab at the bottom. Now you should see the **Tutorial.ttp** project appear.

Right-click on the project and choose **New Folder**. Create a folder named **ATS Files**, in the **File extensions** field enter: *.ttcn; *.asn

Right-click the new folder **ATS Files** folder and choose **Insert files**. Add the `TesterTutorial.ttcn`, `TT.ttcn` and `TesterTutorial.asn` files from the `<tutorial_dir>` directory. The Workspace should now look like [Figure 3 on page 7](#).

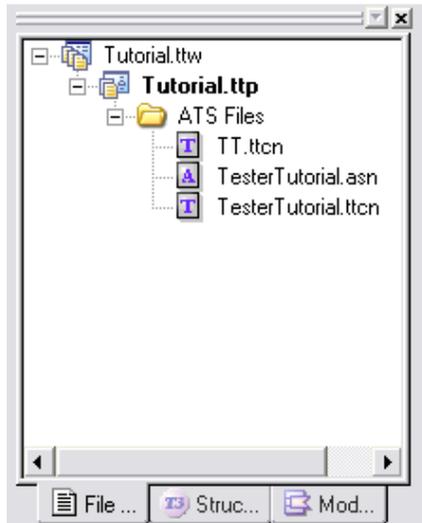


Figure 3: Workspace File View

Analyze the test suite files

Now all the ATS files have been added to the project. **Save All** by clicking the icon on the tool bar or selecting the entry in the **File** menu. Finally click the **Analyze** button (check box symbol) or the entry in the **Project** menu. The

analysis should go through without problems, warnings or errors. The default settings for analysis are sufficient. Go to the **Project** menu, open the Settings dialog. The settings are found in TTCN-3 and ASN.1 tabs.

Test behavior

Goal

This part shows the purpose of the test cases and what is communicated between the ETS and the SUT during test execution.

In the test suite there are three test cases, one using asynchronous communication only (TC1a), one that uses synchronous communication (TC2s) and the last one (TCtimer) uses timers without any communication whatsoever.

Look at test case TC1a

Open the **TesterTutorial.ttcn** file and locate the test case definition named TC1a.

This test case requests the SUT to perform two operations on supplied operands. One operation is requested by the PTC and the second by the MTC. Below is a picture that shows how the message sequence goes:

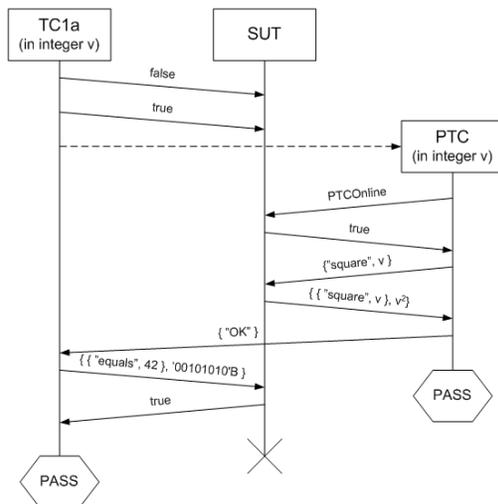


Figure 4: Sequence diagram of TC1a

The picture shows what happens in this test case. A PTC is created to perform a request to the SUT to calculate the “square” operation on a value, it verifies the returned result and signals to the MTC that it went fine and then it terminates. After the MTC detects that the PTC operation is finished it requests the SUT to perform an equality operation on an integer value and a bit string value and then the test is finished.

Look at test case TC2s

Open the **TesterTutorial.ttcn** file and locate the test case definition named **TC2s**.

This test case is somewhat simpler but instead involves synchronous communication. Below is a picture that shows how the message sequence goes:

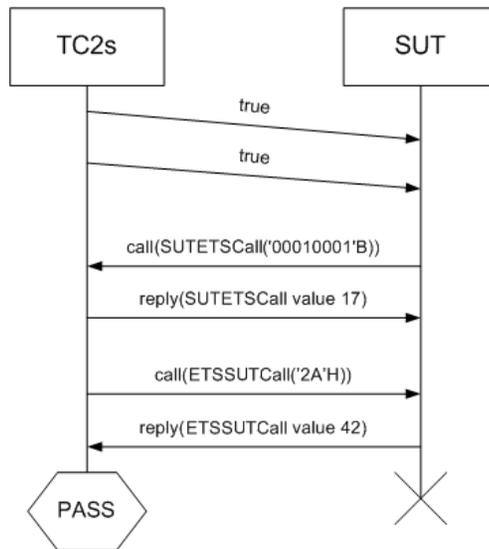


Figure 5: Sequence diagram of TC2s

The picture shows what happens in this test case. The SUT initiates a call to the MTC which returns an integer value converted from the bit string argument. The same operation is then done in the other direction with a hex string value after which the test is finished.

Look at test case TCTimer

Open the **TesterTutorial.ttcn** file and locate the test case definition named `TCTimer`. The test case is simple to explain without a picture. The MTC starts both its timers (`T1` and `T2`) and waits for them to time out in the specific order `T1` followed by `T2`. This test sets the requirement for the timer implementation that it can handle multiple timers. More about the timer implementation is explained in [“Integration implementation” on page 14](#).

Communication and Ports

Goal

This part shows the communication between the test and the SUT and the role of the ports.

The SUT presents four ports through which communication can be performed. They are called P0-P3. The SUT ports have different roles and purposes. The port roles are described for each of the test cases.

Port mappings of TC1a

Open the **TesterTutorial.ttcn** file and look at the port definitions of the component type `MyMTCa` on which test case `TC1a` runs. It has two ports; `MTC2TSIport` and `MTC2PTCport`.

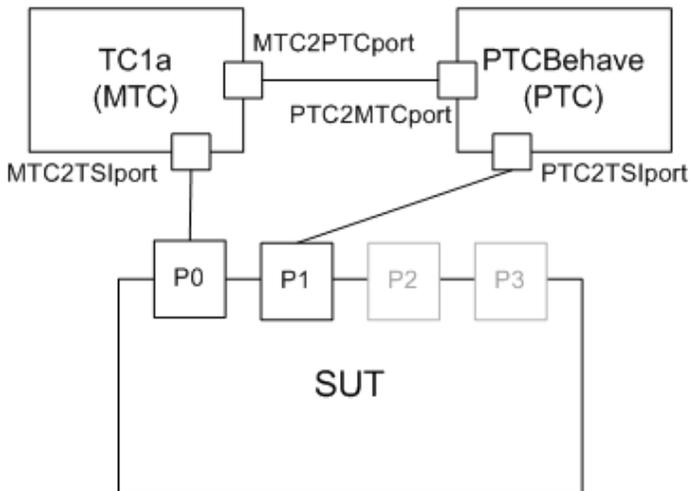


Figure 6: Port mappings of TC1a

As can be seen in the component and port type declarations, the TSI port P0 is used by the MTC and the P1 port is used by the PTC.

Port mappings of TC2s

Open the **TesterTutorial.ttcn** file and look at the port definitions of the component type `MyMTCs` on which test case `TC2s` runs. It has three ports: `MPortIn`, `MPortOut` and `MPortInit`.

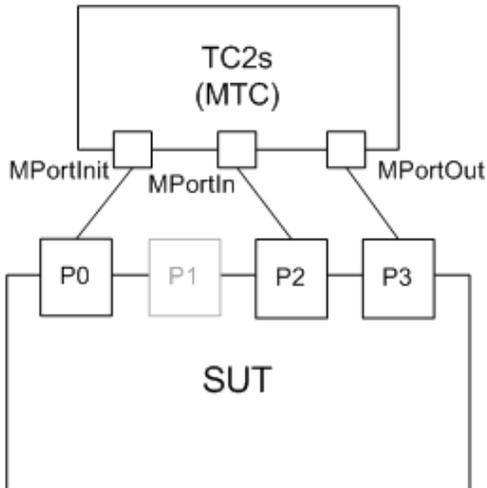


Figure 7: Port mappings of TC2s

The `MPortInit-P0` port mapping is used for asynchronous initialization signaling and the `MPortIn-P2` and `MPortOut-P3` mappings are used for synchronous communication.

Integration implementation

Goal

This part shows what is required from a TRI integration implementation, which functions need to be implemented and how. After this part you will have built a TTCN-3 Project with both ATS files and the necessary TRI based integration.

Code modules

The implementation code is divided into five different code modules (with separate C header files):

- `testertutorial_tri.c`
This is the main integration file which implements the necessary TRI functions. The next step covers the functions in this file.
- `tut_comm.c`
This contains the helper functions for composing and decomposing the information sent between the ETS and the SUT over the socket.
- `tut_ports.c`
This contains the representation and helper functions for port mappings.
- `tut_codecs.c`
This contains the required encoding and decoding functions for the used types. There is only codecs support for the built-in TTCN-3 types used in the test suite; `boolean`, `integer`, `bitstring` and `hexstring`. Note that the ASN.1 types already have the codecs set for them, automatically generated by the Compiler.
- `tut_log.c`
This is contains the log mechanism developed for the tutorial. It is covered in [“Run the ETS” on page 21](#).

Add the integration files to the project

Make sure you have the **File View** tab active in the **Workspace** view. Right-click on the project and choose **New Folder**. Create a folder named `Integration Files`, in the **File extensions** field enter: `*.c; *.h`

Right-click the new `Integration Files` folder and choose **Insert files**. Add the C files listed above from the `<tutorial_dir>` except `tut_log.c`, together with their respective header files. `tut_log.c` file, containing the new log mechanism, will be added in the next part. Note that the `testertutorial_tri.c` file has no header file. The project should now look something like below, the order of the files are not important:

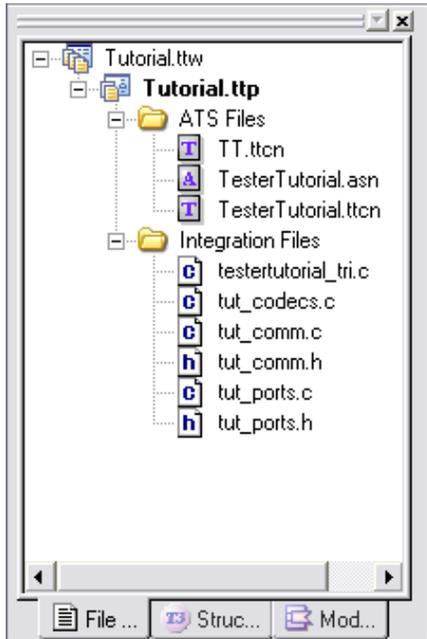


Figure 8: Workspace TRI file view.

In-depth look at TRI TE-SA functions

Now it's time to look at the TRI integration of this tutorial. There are a number of functions that needs to be implemented and they are covered below, going through each one of them. Some functions are not implemented in this tutorial, namely the `triRaise`, `triSUTActionInformal` and `triSUTActionTemplate` functions as well as all functions with MC and BC suffixes that implement multicast and broadcast communication operations.

Open the `testertutorial_tri.c` file to look at the TRI functions. First the TE->SA functions:

- `triSAReset`

This function is invoked in the initialization phase of the ETS execution, before any TTCN-3 test code has been executed. In this tutorial it performs three tasks; it initializes the socket communication, it creates the protection for the port representation and finally it creates the socket reader thread.

- `triExecuteTestcase`

This function is invoked before each test case is started. In this implementation it ensures that port representation structures are cleared.

- `triEndTestcase`

This function is invoked at the end of each test case and usually performs cleanup actions. In this implementation the only thing that is necessary is clearing the port representation.

- `triMap`

Each time a map statement is encountered, this function is called. It can also be called implicitly if you have not specified a system component for your test case. This is not exemplified in the tutorial, see the TTCN-3 standard specification for information on this.

In this implementation a new mapping is created and stored in a list of ports.

- `triUnmap`

The unmap statement is not used in the ATS for this tutorial but this function is still implemented for symmetry reasons.

In this implementation the mapping is removed from the list of ports.

- `triSend`, `triCall`, `triReply`

These are the communication functions of TRI. All these functions works in the same way since in TRI, even the synchronous operations of call/reply are made asynchronous. The synchronous behavior is handled completely by the RTS.

- `triSendMC`, `triSendBC`, `triCallMC`, `triCallBC`, `triReplyMC`, `triReplyBC`, `triRaise`, `triRaiseMC`, `triRaiseBC` (not implemented)

These functions are not implemented for this tutorial. It is analogous to the previous functions and can be implemented by the interested user.

Functions with MC and BC suffixes correspond to multicast and broadcast versions of communication operations.

- `triSUTActionInformal`, `triSUTActionTemplate` (not implemented)

These simple functions are not implemented for this tutorial. Their purpose and actions are completely test suite and SUT dependent. They have no meaning in this tutorial.

In-depth look at TRI PA-SA functions

There are two different ways of implementing timers. One is a *passive* scheme, where timer status is polled by the RTS and the other is *active*, where timeouts are signaled through the `triTimeout` function. The implementation for this tutorial is the *active* one since that is the approach suggested by TRI. To instruct the ETS to use this scheme, the boolean configuration key `t3rt.timers.assuming_all_active` must be set to true.

Then we have the timer related functions

- `triPAReset`

This function is called once, before the control part of the root module is executed. In our case we simply initialize the timer representation.

- `triStartTimer`

This is called whenever a timer is explicitly started in the ETS. Note that implicit timers, like when you specify a timeout value on `execute` or `call` statements, will not trigger this function, it is dealt with in a different way. See the reference documentation for more details. The design here is to create a new thread for each new timer and after it has slept for the appropriate amount of time it generates its own timeout.

- `triStopTimer`

This is called when a timer is explicitly stopped by the TTCN-3 code using the `stop` statement.

- `triReadTimer`

This is called when a timer is explicitly read by the TTCN-3 code using the `read` statement.

- `triTimerRunning`

This is called when a timer is explicitly checked by the TTCN-3 code using the `running` statement.

- `triExternalFunction` (not implemented)

This is called when an externally declared function is called from the ETS. No such functions are used in this tutorial so it is left unimplemented.

Build the ETS

Goal

In this part you will generate a makefile for the tutorial and build an executable test suite.

Prepare makefile generation

The configuration for the makefile generation usually is defined using “Makefile Options” dialog on “Build” tab in project settings.

Open the **Project->Settings** dialog and bring up the **Build** tab. Press “Makefile Options...” button. **Makefile Options** window opens. As you may see window has several tabs. Typically you need to use **General** and **Adapter** tabs only.

Open **Adapter** tab. “Libraries” entry control contain only one `libt3tri.lib` library. Add `ws2_32.lib` after it, use comma to separate libraries in the list.

Note

C files that are put into project are assumed to be adapter files (unless they are excluded from build) thus there is not need to specify them in “Source files” entry control.

Settings in the dialog

Open the **Project->Settings** dialog and bring up the **Build** tab. In the **Root Module** field, write `TutorialModule` which is the name of the top module for the tutorial test suite. In the **Test management** choose “Static using RTS functions in ETS.”

Compile the test suite

Now push the **Compiler** button or choose **Project->Compiler** from the menu. The code is generated together with a proper makefile (`makefile_TRI.mak`). Tester automatically sets the makefile name in the project settings to `makefile_TRI.mak`.

Build the ETS

Open the **Project->Settings**, in the **Build** tab and ensure that `makefile_TRI.mak` is set for the **Makefile** field.

Press the **Build** toolbar button or the **Project->Build** menu entry and the Compiler invokes the make command on the makefile and the ETS is built.

In your directory you should now have a file named `TutorialModule.exe` which is your finished ETS.

Run the ETS

Goal

Run the test executable including how to set module parameters and to add your own log mechanism.

Related documentation: “Execution Logs” (PDF on CD)

The SUT must be started before the ETS is executed. This is due to the fact that it acts as a server from a socket point-of-view and the tutorial ETS needs to connect to it.

Start the SUT

From command-line, run the `<tutorial_dir>/tutorial_sut.exe`.

```
> tutorial_sut.exe [-p <portnr>]
```

The default port number is 30000 and doesn't have to be changed unless it collides with another process on your machine. If the port number is changed on the SUT side, it also has to be modified on the ETS side.

The SUT is running properly when it says:

```
SUT: Tutorial SUT is running.  
Start the tutorial ETS.
```

Execution switch

The test is using generated codecs for the ASN.1 types. The SUT uses the standardized BER encoding and the ETS will of course need to use the same-codec. The selection of `codec` can be controlled through applying a switch.

If the `codec` is not specified, the test will end in error due to missing encoder/decoder function.

Run ETS from Tester

From the **Project** menu, select **Settings**. Go to the **Execution** tab. Switches shall be given in the **Additional execution switches** field under the **Execution** tab in the **Settings** dialog.

Apply the following switch:

```
-confbool t3asn.ExtASN.BER.DEFINITE-LENGTH true
```

All keys passed to RTS should be declared inside `-t3rt ""` command line parameter key:

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

Check **Assume timers in TRI are implemented as “active”**. This sets `t3rt.timers.assuming_all_active` key.

Note

Switches can also be supplied in a configuration file specified in the 'Optional configuration file' field. For your convenience, there is a file named `TesterTutorial.conf` in the tutorial directory that contains the appropriate codec switch.

In the **Execution** tab ensure that **TutorialModule.exe** executable is set for the Executable test binary field. Select the `TesterTutorial.conf` configuration file for the **Optional configuration file** field. The dialog should look something like this (here, assuming your directory is `C:\tutorial`):

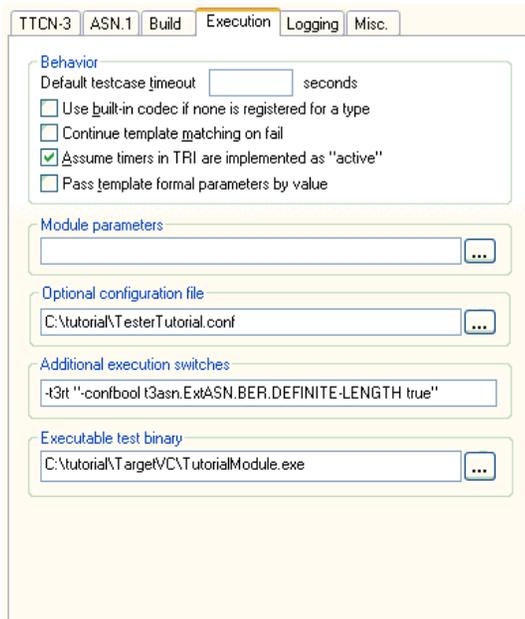


Figure 9: Settings for ETS execution

Now push the **Execute** button and the test should run. The execution log is written in the output window of Tester. See the documentation on “Execution Logs” (PDF on CD).

See also

“ASN.1 Encoding” on page 1346 in Chapter 36, Creating an ETS
“Configuration Files” on page 1373 in Chapter 38, Execute Tests

Run the ETS from command-line

From the command-line, go to your project directory.

All keys passed to RTS should be declared inside `-t3rt ""` command line parameter key. Apply the following switches:

```
-t3rt "-confbool t3asn.ExtASN.BER.DEFINITE-LENGTH true  
-confbool t3rt.timers.assuming_all_active true"
```

Command line switches can also be supplied in a configuration file. For your convenience, there is a file named **TesterTutorial.conf** in the tutorial directory that contains the appropriate `codec` switch, you can use it like this:

```
> TutorialModule.exe -t3rt "-file TesterTutorial.conf"
```

The ETS can be invoked with no other special switches unless you have changed the SUT socket port number. To change port number on ETS side use “`t3rt.tutorial.sut_port <integer key>`”. For example to assign port #12345 add `-t3rt "-confint t3rt.tutorial.sut_port 12345"` to ETS command line parameters.

See also

Chapter 38, Execute Tests in Tester help.

Logging

The logging of the test execution is based on log events generated by the TTCN-3 RTS. New log mechanisms can be plugged in to the RTS to fill the need for special logging formats and destinations (for example a database). For information on how to develop a log mechanism, see the “Technical integration information” documentation (PDF on CD).

The log mechanism module in this tutorial is logging the events of the test execution into separate HTML files, one per component. It also generates an index page over the generated files. The files are all named `tut-<hhmms>-n.html` and the index file is named `tut-<hhmms>-index.html (<hhmms>= time stamp)`.

Add the log module

Add the `tut_log.c` file from the `<tutorial_dir>` directory into your project in the Integration Files folder. This will automatically add it to the list of adapter files in the generated makefile.

Rebuild and run the ETS again

Redo the following to run the tutorial again:

- [“Compile the test suite” on page 19](#)
- [“Build the ETS” on page 20](#)
- [“Start the SUT” on page 21](#)
- [“Run ETS from Tester” on page 21](#) or [“Run the ETS from command-line” on page 23](#)

Make sure that any previous SUT or ETS process is terminated.

Module parameters

The main TTCN-3 module (`TutorialModule`) has a module parameter called `MPar1` declared in the test suite as:

```
modulepar { integer MPar1 := 5 };
```

This value is used as argument to the Square operation involved in the TC1a test case. To override the default value for this parameter, a new value can be given on the command-line with a switch or through a module parameter input file. The notation for using a direct switch is:

```
>TutorialModule.exe -t3rt "-par Mpar1 10"
```

and notation for module parameters file is:

```
>TutorialModule.exe -t3rt "-parfile tut_parfile"
```

Where the `tut_parfile` should contain the entry:

```
MPar1 := 8;
```

ETS with different module parameter values

Try these two different ways of overriding the module parameter while going through [“Run the ETS” on page 21](#) again. Remember that the `TesterTutorial.conf` file must still be given as argument to the ETS.

Food for thought

In this section some points are made about the implementation of this tutorial and how it might differ from a more realistic situation.

Separate communication channels

All communication with the tutorial SUT is made using one channel (socket) only. This is how the SUT has been defined, the design with only one socket is chosen for readability reasons.

In another situation it might be that the SUT uses separate communication channels for each port. This would mean that the port representation must keep the channel stored in each port mapping, which is a simple thing to implement. It would also mean that each communication channel needs to be checked for incoming data, either by having a separate thread for each channel or having the (equivalent of the) `tut_socket_reader` function to trigger communication on all known channels.

More PTCs

There is no limitation in the implementation if a test involves multiple PTCs. The `TC1a` test case creates one PTC. This PTC communicates with the SUT and only when it is finished, the MTC communicates with the SUT. Two threads are running but they communicate over the same socket at different times.

The implementation uses local variables only and uses a single call to the socket `send` function so it is safe.

Timer limitations

The timer implementation is not multi-thread safe and has a limitation on the number of timers. It works currently since the only component that is executing any behavior is the MTC. If multiple components should use timers, the list would have to be protected with a mutex and the list would most probably have to be dynamic in size. This is quite straightforward and is left for the interested reader to implement.