Rational. Systems Tester

Technical Integration 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.

1

Introduction

About This Document

This Technical Integration Documentation describes what an ETS is, what parts it consists of and the required work necessary to make it perform its sole intended task, executing tests.

The aim of this document is to enable a developer (or at least someone with programming experience) to develop a working integration. The first chapter also aims at giving managers and testers an overview of the subject.

Chapter 2, ETS Architecture, describes the composition of an ETS and a little on how they interact to perform the execution of an ETS.

Chapter 3, Integrations, describes the necessity of an integration as well as information on how to choose an already existing integration or implement an integration from scratch.

Chapter 4, Log Mechanisms, describes how to create and apply user-defined log mechanisms to attain the desired log output.

Chapter 5, Codecs Systems, describes how to implement encoders and de-coders and how to make the ETS use them where intended.

Chapter 6, Miscellaneous, describes wide string and binary string support.

Chapter 7, Runtime System APIs, is an extensive function reference chapter that describes each function in the available interfaces individually on what they do or how they should be implemented.

See Also

”Creating an ETS” in the online help

”Executing Tests” in the online help

2

ETS Architecture

ETS Architecture Overview

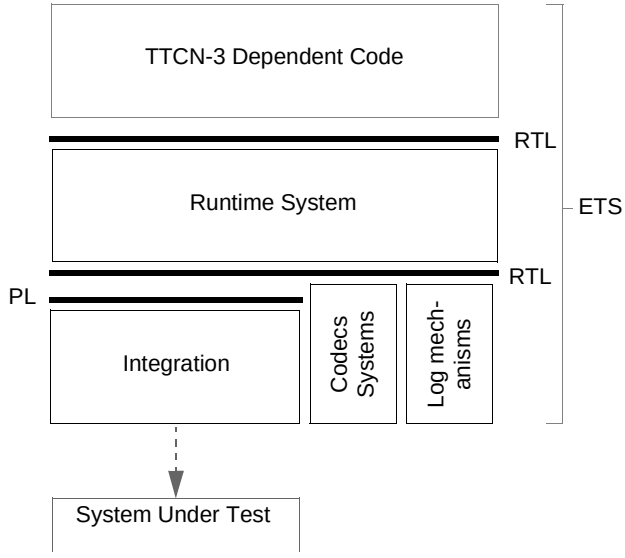


Figure 1: Architecture of an Executable Test Suite (ETS)

This illustration depicts the main parts of an ETS.

- **TTCN-3 Dependent Code**

  This is the necessary information generated from the TTCN-3 test suite by the Rational Systems Tester Compiler.

- **Runtime System (RTS)**

  This is the provided library implementing the semantics of TTCN-3. The RTS provides the services defined by the Runtime Layer (RTL) interface.

- **Integration**

  This part must be provided by the user, in order to connect the test system to the System Under Test (SUT). The integration implements the services defined by the Platform Layer (PL) interface.

From the RTS's point of view, an integration is an implementation of the PL interface. Besides an implementation of the PL interface, at least one **codecs systems** is required to perform encoding and decoding of values.

Log mechanisms are code modules that handles all the log events generated from the RTS. Any number of log mechanisms can be plugged into the RTS. A text based log mechanism and an MSC log mechanism are also provided with the RTS.

See Also

"TTCN-3 ATS Generated Code" on page 5

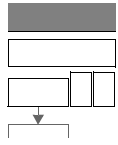
"Runtime System" on page 5

"Integrations" on page 8

"Codecs Systems" on page 9

"Logging" on page 10

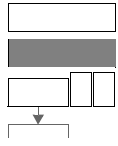
TTCN-3 ATS Generated Code



The generated code from the Compiler maps the constructs of the TTCN-3 language to corresponding representations in the C language. Details on this mapping is outside the scope of this section. The generated code is supposed to be "invisible" since it just serves as a (vital) intermediate step in the transition from the ATS to the ETS.

Some generated entities (for example type descriptors) are described in "Runtime System Details" on page 23 in Chapter 3, *Integrations*.

Runtime System



The runtime system is the "engine" of the TTCN-3 test suite execution. It handles values, controls components, and so on.

In the figure "Architecture of an Executable Test Suite (ETS)" on page 4, the two interfaces defined by the RTS can be seen as bold horizontal lines. One is the Runtime Layer (RTL), and the other is the Platform Layer (PL).

The RTL is the services provided by the RTS. This is used by the generated code, by non-TRI integrations, encoders/decoders, and so on. The PL defines the services that the RTS needs from the integration to be able to function properly. Both these interfaces are in turn divided into smaller parts.

Note

For TRI based integrations there is also the TRI specification that is a pre-requisite before attempting to implement a TRI based integration. The TRI has its place inside the integration module. See "TRI Based Integrations" on page 12 in Chapter 3, Integrations.

Execution Environment

The execution model of the RTS is based on the concept of component instances (of declared component types), running in parallel, executing TTCN-3 functions (test case, test step, function, and so on).

The implementation of the threads of execution is provided by the integration (through the PL interface) to make the RTS as independent from the platform as possible.

The intended execution model is to have components running in separate threads of execution (multiple threads within one process, or as separate processes). By nature, such an execution model is prone to "race" conditions. This means that the test results may depend on the scheduling of tasks in the underlying platform. This integration "freedom" might become a real problem only when running poorly written test cases, where the potential scheduling variations have not been taken into consideration.

Memory Handling

Throughout the execution of the test suite's control part, a lot of memory allocation has to be made. These allocations can either be done using permanent or temporary memory.

The temporary allocation strategy is managed by the RTS, and is a dynamically growing memory area expanded on demand. A separate temporary memory area exists for each TTCN-3 component. The temporary allocation is a position-oriented, stack based implementation with de-allocations made by manipulating (resetting) positions. When a memory position is reset to a previous mark, with memory being allocated again, the previously de-allocated blocks will be reused.

The main idea behind this functionality is to minimize problems with permanent memory allocations. Permanent allocations in heap memory are known to be time consuming (that is, slow) and if allocated memory is not explicitly de-allocated, memory leaks will occur.

Permanent allocations are allocations made in heap memory, which therefore have to be de-allocated explicitly. The services needed for permanent memory allocations are provided by the PL implementation and are necessary for the implementation of the temporary memory handling in the RTL.

Pre-initialization, Initialization and Finalization

The integration interface is divided into several modules, each having a specific responsibility. Each one of these module has to be initialized before it can carry out its services and clean-up after it has finished.

The initialization step has been divided in two phases, the “pre-initialization” and the “initialization” phase. In most cases it is sufficient to place all initialization code into the initialization phase. The only module that is required to be (at least partially) functional after the pre-initialization phase is the PL Memory module (described in “Memory” on page 22 in Chapter 3, *Integrations*).

Configurability

The RTS can be configured in a number of ways to control its behavior. For this, a general storage facility is available, populated with key/value pairs where the keys are predefined and known to the RTS. The values are represented as TTCN-3 RTS values, so basically any kind of value can be used for configuration information.

The storage facility is also intended to be used by integrations, codecs, and so on, where configurability is relevant.

The storage facility can be populated either by providing special switches on the command-line to the ETS, or programmatically using a single API function to set the values. The command-line approach is limited to a subset of the basic, non-structured types in the RTS. Reading from the storage is done by a single access function.

Source Tracking

The RTS has a mechanism available to keep information on source code locations during execution. This is primarily intended to track the test execution through the TTCN-3 test suite source. This mechanism is referred to as “source location” information.

The secondary “flavor” of this mechanism is to track execution in other integration modules like, for example, encoder and decoder functions and log mechanisms implementations. This approach is referred to as “target code location” information. It is intended to be used as an optional (but handy) mechanism to be used by integration developers.

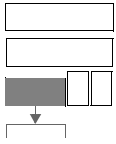
Source tracking information is visible in all events logged by the built-in log mechanism. The functions can be found in “RTL Source Tracking Functions” on page 287 in Chapter 7, *Runtime System APIs*.

See Also

“Runtime Layer API” on page 48 in Chapter 7, *Runtime System APIs*

“Platform Layer API” on page 306 in Chapter 7, *Runtime System APIs*

Integrations



One of the parts of the ETS that you, as a user, have to provide is the module that makes the RTS interact with the SUT. This is referred to as “an integration”.

An integration is an implementation of the PL interface defined by the RTS. The PL interface is what the RTS requires in terms of integration services to be fully functional. The integration provides memory primitives, representation of timers, handling of time, SUT communication, task concurrency primitives, and so on.

Different SUTs require different integrations in terms of timer implementations, communication mechanisms, and so on. The RTS also requires other services such as concurrency primitives (for executing parallel test components), semaphores for thread synchronization, and so on.

Equipping the ETS with an integration implementation is done by implementing a set of required functions that the RTS needs. This can be done in three different ways:

- **Using the provided TRI integration**

  This is the most simple choice which minimizes the implementation effort to only cover the System Adaptor (SA) and Platform Adaptor (PA) functions that the TRI document specifies.

- **Extending and modifying the non-TRI example integration**

  Starting from a working example implementation of a non-TRI integration you can adjust the integration for your own needs without having to implement all of the PL functions from scratch.

- **Implementing a non-TRI integration from scratch**

  This is by far the most flexible way of making an integration but it requires some more work than either of the previous alternatives. Using this approach means that you must implement all the functions in the PL yourself.

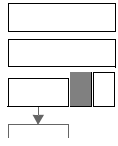
See Also

Chapter 3, Integrations, for details on how to make an integration, and for descriptions of already provided integrations.

“TRI API” on page 348 in Chapter 7, *Runtime System APIs*

“Platform Layer API” on page 306 in Chapter 7, *Runtime System APIs*

Codecs Systems



To be able to pass TTCN-3 values between components through the communication primitives, for example `send` and `call`, functions to encode and decode values must be provided. The encoders take values and encode them to a transferable binary representation, and the decoders make the reverse operation.

After the initialization phase, all declared types that will have to be encoded must have one associated encoder function and one associated decoder function. It is the responsibility of the codecs system to provide both these function as well as making the association.

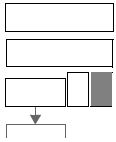
A codecs system is basically a set of encoder and decoder functions. Preferably they all implement the same encoding scheme (BER or PER for ASN.1, for example).

The RTS supports multiple codecs systems, which are registered at runtime in the initialization phase. In this phase, all registered codecs systems will be asked in turn to associate encoder and decoder functions for the existing types in the system.

See Also

“Codecs Systems” on page 41 in Chapter 5, *Codecs Systems*.

Logging



Logging the execution of a test is an essential part of the RTS. A default text-based log mechanism is provided, as well as a mechanism that logs to file with MSC-96 syntax.

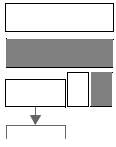
The RTS supports a very flexible and easy way to plug in any number of user-defined log mechanisms to cover any need from any customer.

Each component has its own log instances, and only events and information messages that is related to that specific component will be logged to its corresponding log instances.

See Also

“Log Mechanisms” on page 31 in Chapter 4, *Log Mechanisms* on how to provide your own log mechanisms.

Internationalization and Localization



The runtime system is internationalized, that is, all necessary mechanisms are in place to enable localization.

Error messages and predefined configuration key descriptions are the only localized items. The only currently available localization target is U.S. English.

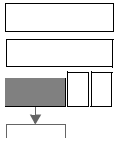
The intention is not to enable user-defined localizations, but to support localization requests from customers.

The internationalization mechanism supports character sets with a maximum of up to 32 bits per character.

3

Integrations

TRI Based Integrations



TRI is a specification that defines an interface between the test executable and the SUT. In TRI terminology, it is an interface between Rational Systems Tester (TE), the System Adaptor (SA) and the Platform Adaptor (PA).

The IBM Rational TRI compatible integration is provided both as a library and as source code. The library can be found in the `integrations/tri/<platform>` directory and the source code is the same as the example PL integration found in the `integrations/example`. Both are located in the Rational Systems Tester installation directory.

You have to implement the functions required on the SUT side of TRI (that is, the TE->SA and TE->PA functions defined in ETSI TR 102 043 V1.1.1 (2002-04)). This module should include the Rational Systems Tester TRI header file, `integrations/t3tri.h`, which contains the TRI definitions according to the TRI to ANSI-C-mapping.

Template and example implementations can be found in the `/integrations/tri` directory. They are are provided to make it easier to implement the necessary functions.

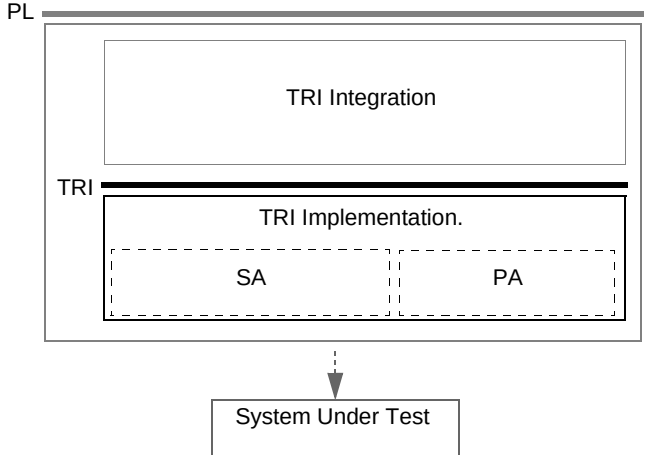


Figure 2: TRI Based Integration

Implementation Information/Hints

TRI, as defined in ETSI TR 102 043 V1.1.1 (2002-04), covers SUT communication, timer support, and external function calls. This is not sufficient for a complete integration in terms of PL functionality, hence it does not cover every need of the RTS. Therefore, the rest of the functionality is provided.

In the current TRI integration, all TTCN-3 components are executing in one and the same process, but each component executes in a separate thread. Communication between TTCN-3 components is handled within the provided integration.

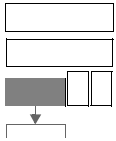
The timer implementation is based on real-time, leaving the TRI implementation no means of controlling the advancement time.

Note

Due to the non-blocking nature of the TRI, the TRI implementation (SA and PA) has to run in a separate thread other than the RTS. This thread should be created early (for example in the `triSAReset` *function) and should handle all the requests made by the RTS.*

PL Based Integrations

PL Integration Modules



This section describes the different part of the integration that is required to be implemented to complete the ETS. The definitions for these parts can be found in the `t3pl`-prefixed header files in the `include` directory in the Rational Systems Tester installation directory.

An example integration implementation is described in “Example Integration” on page 28.

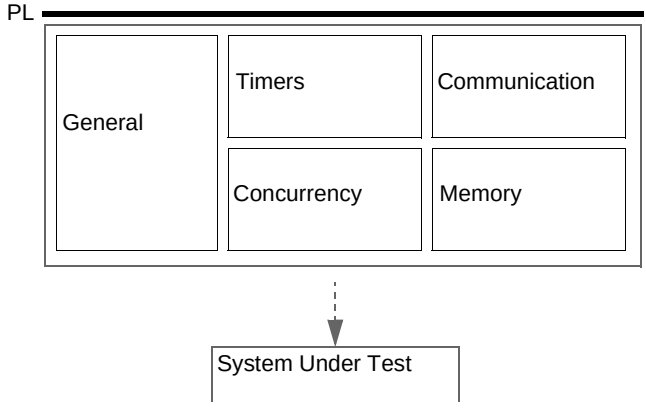


Figure 3: Non-TRI Based Integration

These are the sub-modules of a non-TRI based integration.

- “General” on page 15

  Covers the bits and pieces not covered by the other parts.
- “Timers” on page 15

  Handles timers and their operations.
- “Communication” on page 17

  Covers the handling of ports, and the communication primitives for these.

- "Concurrency" on page 20

  Provides the necessary functionality to execute components in parallel.
- "Memory" on page 22

  Implements the necessary memory primitives for memory handling.

General

This part currently contains pre- and postamble functions to prepare for and clean up after a test case. It also contains one function that should implement handling of external function calls.

General functions

These are the functions that are required by the RTS with respect to general behavior.

Function to implement	Description
`t3pl_general_prepare_testcase`	Prepares testcase execution.
`t3pl_general_postprocess_testcase`	Finalizes testcase execution (called within MTC) during testcase termination.
`t3pl_general_testcase_terminated`	Finalizes testcase execution (called within CPC) after testcase termination.
`t3pl_general_control_terminated`	Finalizes control part execution.
`t3pl_call_external_function`	Performs external function call.

See Also

"Platform Layer API" on page 306

Timers

Timers are a very important concept in a test suite and the actual timer implementation (representation and operations) is made in the integration. These services are used by the RTS to carry out the TTCN-3 semantics.

The timer API consists of functions for creating, controlling, and destroying timers. Timer are always created on demand from the RTS. Both TTCN-3 declared timers and implicit timers can be created.

Timer are declared locally in TTCN-3 components and are **never** shared between them.

Timer Functions

These are the functions that are required by the RTS with respect to timers.

Function to implement	Description
`t3pl_time_pre_initialize`	Initializes as much as possible to make the module work. The implementation of this function can **not** rely on configuration information through RTconf.
`t3pl_time_initialize`	Initializes (or re-initializes) the module. The implementation of this function can rely on configuration information through RTconf.
`t3pl_time_finalize`	Finalizes the module. The implementation of this function can rely on configuration information through RTconf.
`t3pl_timer_create`	Creates a timer, and returns a handle that will be used by the RTS to refer to it.
`t3pl_timer_delete`	Deletes the timer with the given handle.
`t3pl_timer_start`	Sets a given timer in the started state (“running”).
`t3pl_timer_stop`	Sets a given timer to the stopped state (not “running”).
`t3pl_timer_read`	Reads the state and elapsed time of a given timer.
`t3pl_timer_decode`	Returns binary string containing TriTimerId corresponding to provided internal RTS timer handle.

Active and Passive Timers

A timer can be either passive or active (not both), but a system can consist of a mix of both active and passive timers. The difference between them is how time-outs are detected. The time-out of a passive timer is only detected when

the timer's status is explicitly asked for by the RTS, while an active timer reports its time-out actively by telling the RTS when this happens. Effectively, potential time-outs are detected earlier when active timers are used.

By default, the RTS supports both passive and active timers simultaneously. During execution, when turning to the communication part of the integration for more data to process, a maximum time to wait is calculated from the states of the known running timers. This is for keeping the waiting time to a minimum. As a result from this, the RTS is real-time dependant. If more control over the advancement of time is desired, the RTS can be configured to only trigger on active timers and wait, in effect, until data is received or a time-out is actively triggered.

See Also

"PL Timer Functions" on page 310 in Chapter 7, *Runtime System APIs*

"Predefined Configuration Keys" in the online help for information on how to control the RTS.

Communication

All communications are made in terms of the ports concept of TTCN-3. Ports are the only means of transporting information between running components both in synchronous (that is the `send` operation) and asynchronous (the `call` operation) communication. The communication is highly oriented around the component, port, and binary data.

Ports are always created on demand from the RTS and are uniquely defined by a globally unique port address. Global uniqueness here means that no two ports may have the same address in the whole (arbitrarily distributed) test system. Ports and their addresses are allocated and maintained by the integration, as well as by the actual implementation for passing information between them.

Note

There are other ports than the declared ports of a TTCN-3 component type. Such ports are used as administration ports to control the components, and is a result of the design of the RTS rather then imposed by TTCN-3. They are referred to as "control ports" and requires very little special treatment.

Communication Functions

Function to implement	Description
`t3pl_communication_pre_initialize`	Initializes as much as possible to make the module work. The implementation of this function can **not** rely on configuration information through RTconf.
`t3pl_communication_initialize`	Initializes (or re-initializes) the module. The implementation of this function can rely on configuration information through RTconf.
`t3pl_port_create`	Creates and initializes a port, and returns a globally unique address.
`t3pl_port_create_control_port_for_cpc`	Creates and initializes a control port port for component control, and returns a globally unique address. Very similar to t3pl_port_create.
`t3pl_port_start`	Sets the given port in a started state.
`t3pl_port_halt`	Sets the given port in a halted state.
`t3pl_port_stop`	Sets the given port in a stopped state (disabled)
`t3pl_port_destroy`	Removes and deallocates the given port.
`t3pl_port_clear`	Discards any data contents in the port queue.
`t3pl_port_map`	Associates a given port to a given system port.
`t3pl_port_unmap`	Removes an association of a given port to a given system port.
`t3pl_port_component_send`	Sends encoded data between components.
`t3pl_port_sut_send`	Sends encoded data to the SUT.
`t3pl_port_sut_send_mc`	Sends encoded data to the multiple entities within SUT.
`t3pl_port_sut_send_bc`	Sends encoded data to all entities within SUT.
`t3pl_port_sut_call`	Performs procedure call to the SUT.

Function to implement	Description
t3pl_port_sut_call_mc	Performs procedure call to the multiple entities within SUT.
t3pl_port_sut_call_bc	Performs procedure call to all entities within SUT.
t3pl_port_sut_call_done	Finalizes successful procedure call to the SUT.
t3pl_port_sut_call_abort	Aborts procedure call to the SUT (e.g. due to timeout).
t3pl_port_sut_reply	Performs reply on a procedure call to the SUT.
t3pl_port_sut_reply_mc	Performs reply on a procedure call to the multiple entities within SUT.
t3pl_port_sut_reply_bc	Performs reply on a procedure call to all entities within SUT.
t3pl_port_sut_raise	Raises exception on a procedure call to the SUT.
t3pl_port_sut_raise_mc	Raises exception on a procedure call to the multiple entities within SUT.
t3pl_port_sut_raise_bc	Raises exception on a procedure call to all entities within SUT.
t3pl_port_sut_action	Performs a SUT action.
t3pl_port_retrieve_system_port	Retrieves the address of a named system port.
t3pl_port_release_system_port	Releases the given system port.
t3pl_component_get_system_control_port	Returns the port for controlling the (virtual) system component.

Function to implement	Description
`t3pl_component_set_system_component_type`	Makes the necessary initialization to set up the system "component" to be of the given type.
`t3pl_component_wait`	Waits for any incoming data or active timers to time out, and reports to the RTS when either event occurs.
`t3pl_component_control`	Similar to t3pl_component_wait but processes only control messages and doesn't block.

See Also

"PL Communication Functions" on page 314 in Chapter 7, *Runtime System APIs*

Concurrency

Since the RTS has a true concurrent execution model for the TTCN-3 components, it has to be able to create new execution threads. These so called tasks can be arbitrarily distributed, that is either as new threads within the same process, or as separate processes.

The concurrency integration must provide semaphores to enable the multi-threaded RTS (and integration) to function properly (with respect to thread synchronization). The semaphore services are defined by the `t3pl_sem_`-prefixed function.

The function needed to be implemented is declared in the `t3pl_concurrency.h` file in the `include` directory.

Concurrency Functions

Function to implement	Description
`t3pl_concurrency_pre_initialize`	Initializes as much as possible to make the module work. The implementation of this function can **not** rely on configuration information through RTconf.
`t3pl_concurrency_initialize`	Initializes (or re-initializes) the module. The implementation of this function can rely on configuration information through RTconf.
`t3pl_concurrency_finalize`	Finalizes the module. The implementation of this function can rely on configuration information through RTconf.
`t3pl_concurrency_start_separate_component`	Called when a new component has been started in a separate process. Takes the necessary steps to initialize its control port and returns its address to the creating component.
`t3pl_task_register_context`	Informs the integration of the runtime context that is associated with the current thread of execution.
`t3pl_task_create`	Creates a new thread of execution (possibly in a separate process) for a new component. Returns a fully functional control port to this component.
`t3pl_task_setup`	Setups whatever is necessary for the component to communicate through its ports.
`t3pl_task_kill`	Forces a given task to stop executing.
`t3pl_task_exit`	Exits the current task (called by the terminating task itself).
`t3pl_sem_create`, `t3pl_sem_wait`, `t3pl_sem_trywait`, `t3pl_sem_timedwait`, `t3pl_sem_post`, `t3pl_sem_destroy`	Semaphore primitives.

See Also

"PL Concurrency Functions" on page 337 in Chapter 7, *Runtime System APIs*

Memory

The RTS makes no assumption about the actual memory handling of an integration. It allocates, re-allocates, and deallocates memory only through the PL Memory interface.

The interface is based on the (ANSI-C) `malloc`/`realloc`/`free` paradigm. The functions that must be implemented are declared in the `t3pl_memory.h` file in the `include` directory of the Rational Systems Tester installation.

Important!

*It is **vital** that the memory module is working after the* `t3pl_memory_pre_initialize` *function has been called.*

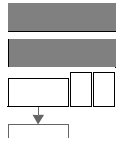
Memory Functions

Function to implement	Description
`t3pl_memory_pre_initialize`	Initializes as much as possible to make the module work. The implementation of this function can **not** rely on configuration information through RTconf. It is **vital** that the memory module is working after this function has been called.
`t3pl_memory_initialize`	Initializes (or re-initializes) the module. The implementation of this function can rely on configuration information through RTconf
`t3pl_memory_finalize`	Finalizes the module. The implementation of this function can rely on configuration information through RTconf
`t3pl_memory_allocate`	Allocates the requested number of bytes of memory.
`t3pl_memory_deallocate`	De-allocates (or releases) a block of memory.
`t3pl_memory_reallocate`	Reallocates (that is, resizes) a block of memory.

See Also

“PL Memory Functions” on page 334 in Chapter 7, *Runtime System APIs*.

Runtime System Details



There are some concepts of the RTS that have to be handled (or at least known) in any implementation using the RTL interface and the RTS. The objects are used to carry data around the system and to function as containers of all information during runtime.

Runtime Context

Throughout the execution of a test suite a wide range of environmental data is needed. For this purpose, a runtime **context** is passed around to (almost) every function in the RTS. A context is created for each new TTCN-3 component and is passed around throughout that component’s execution.

The context contains a diverse set of local information such as local verdict, log instances, a temporary memory area, and so on.

The context is normally neveraccessed but is usually just passed around as an opaque object.

Symbol Table

All the generated objects of the TTCN-3 test suite can be found by symbolic lookup in a symbol table. This is accessed by the RTS whenever symbolic information has to be mapped into relevant data. This can be the lookup of test case functions by name, finding types and constants of a specific module, and so on.

This table is a static entity in the generated code and cannot be made dynamic. However, this is not necessary since it contains statically declared symbols for a specific TTCN-3 module.

Functions for accessing this data can be found in “RTL Symbol Table Functions” on page 292 in Chapter 7, *Runtime System APIs*.

Control Part Component (CPC)

The phrase “executing a test suite” would be more correctly stated as “executing the control part of the root module of the TTCN-3 test suite modules”. In the RTS implementation, the control part always executes in the initial process thread of the ETS, that is, the thread calling the `t3rt_run_test_suite` function. The rest of the components can be distributed arbitrarily, it’s all up to the integration implementation.

For the purpose of executing the root module’s control part and to control all the running test components, a control part component (CPC) is created.

Component Distribution

An arbitrary number of parallel test components are created during the course of the ETS execution. The number depends entirely on what is written in the test suite.

The illustrations that follow show different component distribution topology scenarios. The boxes in the figures depicts process boundaries, and the circles depicts tasks (threads of execution within the process boundary) in which one component is executing. The process boundaries can also be machine boundaries. The arrows shows which component that creates another component.

The “Main process” box is the main process of the ETS executable.

Note

The sole responsibility for starting new tasks in the appropriate process lies on the `t3pl_task_create` *function. See* `“PL Concurrency Functions” on page 337 in Chapter 7, Runtime System APIs` *for further details.*

All illustrations show a possible distribution topology scenario of a test case (executed in an MTC) that has created two PTCs.

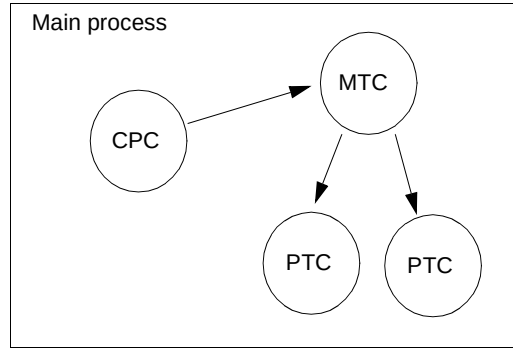


Figure 4: Single process ETS

The above scenario depicts a threading only integration where no component is run in a separate process. Unless special considerations must be made, this topology is recommended as it probably is the most resource efficient and best performing topology.

This is the scenario that both the provided TRI based integration and the PL based example integration is implementing.

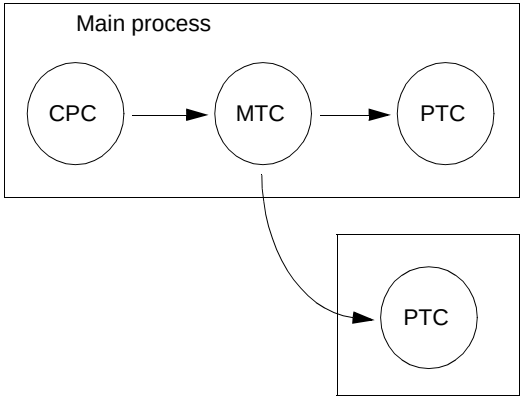


Figure 5: PTC running in separate process

In the above scenario, the major part of the test system is running in the main process but one (or more) PTCs are running in a separate thread. This kind of distribution might be required by some SUTs, perhaps in combination with the kind of tests being performed.

This topology requires a more complex task creation implementation in the integration, as well as support for inter-process communication.

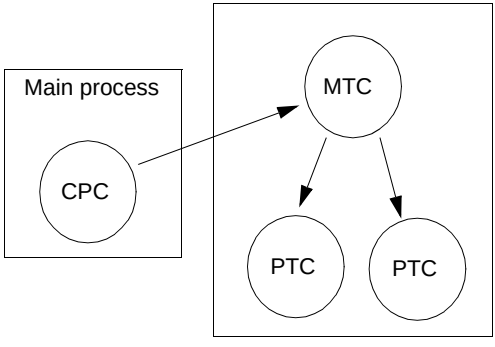


Figure 6: Running the test components in a separate process

In this scenario, the test components are all running in a separate process other than the control part components. This can be useful when the SUT is running in another environment (for example, machine and operating system) than the one the tester is running the tests from. Especially since the communication between the test components and the SUT is the most time critical for the test results.

Control Ports

All components that are running during the execution of a control part are controlled by the control part component (CPC). The controlling mechanism is message based, and all controlling messages are sent through control ports. Each component has one (and only one) control port to which it is always listening for control messages.

All the component control relies on this control port mechanism, from component startup, through execution, to shutting the system down when it has been finished or an error has occurred.

What is required from the integration implementation in this area is the creation of the control ports (very similar to normal ports) and the service to send binary data between such ports.

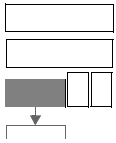
See Also

“Runtime Layer API” on page 48 in Chapter 7, *Runtime System APIs*

“Platform Layer API” on page 306 in Chapter 7, *Runtime System APIs*

“TRI API” on page 348 in Chapter 7, *Runtime System APIs*

Example Integration



This integration serves both as an example and as the template for developing an integration of your own. It is therefore released as source code. You can find it in the `/integrations/example` directory in the Rational Systems Tester installation directory.

Note

This example integration is delivered without support. Changes in future versions will be done without prior notice.

Note

No codecs system is provided by the example integration. At least one has to be provided. See the "Codecs Systems Overview" on page 42 in Chapter 5, Codecs Systems.

General

The functions in the general module are left un-implemented. The implementation can be found in the `example_general.c` file. What have to be added here is support for external function calls.

Timers

This example integration implements a passive timer scheme. The implementation can be found in the `example_time.c` file. If no special timer consideration (like non-linear or non-real-timers) is needed, this implementation should be directly reusable.

Communication

The communication part in the example integration, includes a representation of the currently created ports for the running components, as well as the ports of the current system component.

Since communication is made in a multi-threaded environment, FIFO-like queues are used for passing messages to components, instead of direct insertions into port queues. The messages are dispatched into the proper port input queues by the correct thread, when new information is checked by the receiving component (in the `t3pl_component_wait` function).

An implementation can be found in the `example_communication.c` file. The implementation is a pure “loop-back” (or “reflecting”) integration where all SUT messages sent on one port is sent back on the same port directly, as if the SUT had immediately responded with the same message on the same port.This is one of the fundamental things that has to be changed in order to make this a fully functional communication implementation.

Concurrency

The example integration is a threaded implementation which creates new tasks in new threads within the same process. Support for distributing components into separate processes are part of the RTS design, but the example integration does not implement this.

The implementation can be found in the `example_concurrency.c` file.

Memory

The example integration implements the required functions on top of the standard C memory primitives malloc, free, and so on. If this is sufficient, the implementation can be directly reused. The example implementation can be found in the `example_memory.c` file.

Components

This module contains the representation of a component and information about all its port and timers. Thread information is also stored here for each task. In the current implementation there is a logical one-to-one mapping between a task and component and a thread. (`example_components.c`)

Semaphores

The semaphores needed to protect the multi-threaded implementation is located here. Semaphores are used to protect the critical sections and to synchronize the threads. (`example_semaphore.c`)

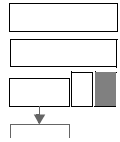
Event queue

Component are executed in separate threads and event queues are used for component-to-component communication. This is an indirection to make sure that a thread other than the components own thread inserts data into port queues. (`example_eventqueue.c`)

4

Log Mechanisms

Log Mechanisms Overview



If you have special requirements for log formats, destinations, behavior, and so on for your implementation, user defined log mechanisms can be developed and plugged into the RTS. This is considered optional since built-in log mechanisms are provided and used by default.

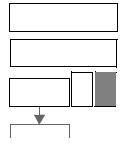
A log mechanism is registered to the RTS, and for each component a log instance will be created. These log instances is called each time something has to be logged. When a component is terminated, the log instance is closed.

Since components have their own execution threads, and possibly different memory address spaces, only events and information messages that occur in a specific component will be logged to any log instance.

Note

In the case where components execute in separate threads within the same process the log function have to be multi-thread safe. This is because the same log function can potentially be called from more than one thread simultaneously.

Implementing Log Mechanisms



A log mechanism consists of a small set of functions: initialize, finalize, open, close, and log event.

The intended place to register your own log mechanisms is the `t3ud_register_log_mechanisms` function. This file must then be compiled and linked with the ETS. Doing so will "shadow" the empty `t3ud_register_log_mechanisms` function already compiled into the RTS.

Init and Finalize

These two functions are called once. The `init` function is called in the initialization phase of the ETS, and the `finalize` function is called when the root module's control part has been executed.

Open and Close

The `open` and `close` functions are called to create and destroy log instances respectively. These instances are then used as a log "stream" object to the log function.

Log Event

The one (and only) logging function is called when something interesting has happened (that is, after the actual event) in the RTS. It is called with the type of the event and number of actual event parameters. There is a predefined set of event types and they are generated by the RTS, or occasionally by the generated code.

The same event and parameters will be sent to all event log instances so that no excessive event generation is performed.

Important!

It is crucial that the data is not tampered with (that is, modified or destroyed) since all log instances will reuse the same data.

See Also

"RTL Port Functions" on page 146 in Chapter 7, *Runtime System APIs*

Registering a Log Mechanism

This is an example of a working log mechanism that logs the verdict of an executed test case to the `stdout` stream.

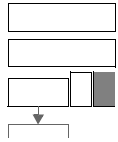
Example 1: Registering a Log Mechanism ----------

```
/* My log mechanism log function in my_log.c. */
static void
my_log( t3rt_log_t log,
        t3rt_log_event_kind_t event,
        t3rt_source_location_t source_location,
        t3rt_value_t params[],
        t3rt_binary_string_t origin,
        const char *origin_name,
        bool forwarded, /* Future extension! */
        t3rt_binary_string_t aux,
        t3rt_context_t ctx)
{
  t3rt_verdict_t verdict;
  char *tcname;

  if (event == t3rt_log_event_testcase_ended_c) {
    t3rt_log_extract_testcase_ended(params,
                                    NULL,
                                    &tcname,
                                    &verdict,
                                    ctx);

    printf("Testcase %s ended in %s\n",
        tcname, t3rt_verdict_string(verdict));
  }
}

...
/* Register my log mechanism in
t3ud_register_log_mechanisms.c. */

void
t3ud_register_log_mechanisms (void)
{
  t3rt_log_register_listener(
        "MyLog",
        t3rt_log_mechanism_version_1_c,
        NULL, NULL, NULL, NULL, my_log);

}
```

See Also

"RTL Log Functions" on page 149 in Chapter 7, *Runtime System APIs*

Pre-Defined Log Events



The following list covers the log events issued by the RTS during execution. The actual parameters to these events are available through the log event **extraction** functions (see “RTL Log Functions” on page 149 in Chapter 7, *Runtime System APIs*).

Log events that is prefixed with [sut] are two separate events that is either component-component based, or component-SUT based. The **Comment** column either states the TTCN-3 operation or statement that is the reason for the event, or just a general comment.

Log event	**Comment**
[sut] message sent, [sut] message sent failed	`send`
[sut] message sent mc, [sut] message sent failed mc	`send to (recipient1, recipient2)`
[sut] message sent bc, [sut] message sent failed bc	`send to all.component`
[sut] message detected	A message/call has been added to the destination port input queue.
[sut] message received	`receive`
[sut] message found	`check(receive)`
[sut] message discarded	`trigger` or as a result of a port being cleared.
sut action performed	`action`
[sut] call initiated, [sut] call failed	`call`
[sut] call initiated mc, [sut] call failed mc	`call to (recipient1, recipient2)`

Log event	Comment
[sut] call initiated bc, [sut] call failed bc	`call to all.component`
[sut] call timed out	An outgoing call failed to complete within specified period.
[sut] call detected	An incoming call has been added to the destination port input queue.
[sut] call received	`getcall`
[sut] call found	`check(getcall)`
[sut] reply sent, [sut] reply failed	`reply`
[sut] reply sent mc, [sut] reply failed mc	`reply to (recipient1, recipient2)`
[sut] reply sent bc, [sut] reply failed bc	`reply to all.component`
[sut] reply detected	An incoming reply has been added to the destination port input queue.
[sut] reply received	`getreply`
[sut] reply found	`check(getreply)`
[sut] exception raised, [sut] raise failed	`raise`
[sut] exception raised mc, [sut] raise failed mc	`raise to (recipient1, recipient2)`
[sut] exception raised bc, [sut] raise failed bc	`raise to all.component`
[sut] exception detected	An exception was added to the procedure based port's queue.
[sut] exception caught	`catch`
[sut] exception found	`check(catch)`

Log event	Comment
[sut] timeout exception detected	An timeout exception was added to the procedure based port's queue.
[sut] timeout exception caught	`catch(timeout).`
[sut] timeout exception found	`check(catch(timeout))`
timer started	`<timer>.start`
timer stopped	`<timer>.stop`
timer read	`read`
timer is running	`<timer>.running`
time-out detected	A time-out of a running timer has been detected.
time-out received, time-out mismatch	`timeout`
component created	`create`
component started	`<component>.start`
component is running	`<component>.running`
component is alive	`alive`
component stopped	`<component>.stop`
component killed	`kill`
component terminated	The component has been shut down by the RTS.
done check succeeded, done check failed	`done`
killed check succeeded, killed check failed	`killed`
port connected	`connect`
port disconnected	`disconnect`
port mapped	`map`

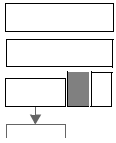
Log event	Comment
port unmapped	`unmap`
port enabled	`<port>.start`
port disabled	`<port>.stop`
port halted	`halt`
port cleared	`clear`
scope entered	A TTCN-3 testcase, function or test step has been entered.
scope left	A TTCN-3 testcase, function or test step has been exited.
local verdict changed	`setverdict`
local verdict queried	`getverdict`
alternative_activated	`activate`
alternative_deactivated	`deactivate`
variable modified	An assignment operation has been performed.
template match failed	`receive/trigger/getcall/getreply/catch` or a separate attempted match did not succeed.
template mismatch	Indicates the position where the template mismatched. (A template match failed event will follow.)
template match begin	Indicates the start of matching subtemplate
template match end	Indicates the end of matching subtemplate
sender mismatch	Indicates mismatch of the template specified in the `from` clause
testcase started	`execute`
testcase ended	The testcase was finished.
testcase timed out	The `execute` operation timed out.
testcase error	A test case error occurred. Reported by the component where the error occurred.

Log event	Comment
message encoded	A value was encoded successfully.
message encode failed	A value was not encoded successfully.
message decoded	A value was decoded successfully.
message decode failed	A value was not decoded successfully.
info message	Information message.
warning message	Warning message.
error message	Error message.
debug message	Debug message.
TTCN-3 message	`log`
alt entered	Indicates entering `alt` block statement
alt left	Indicates leaving `alt` block statement
alt rejected	Guard expression in alternative evaluated to false
alt else	`else` branch in alt block is selected
alt defaults	Indicates starting evaluation of activated defaults
alt repeat	`repeat`
alt wait	Indicates execution reached end of `alt` block statement while all alternatives failed to match, component wait for new events
function call	Indicates function call
external function call	Indicates external function call
altstep call	Indicates explicit altstep call

5

Codecs Systems

Codecs Systems Overview



Please refer to “Codecs Systems” on page 9 in Chapter 2, *ETS Architecture*, for introductions to encoder and decoder functions and codecs systems.

A codecs system is considered to be part of the integration since at least one codecs system is required to be present by the RTS.

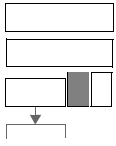
A codecs system has only one responsibility. That is to – when presented with a type of the TTCN-3 system – associate an encoder and decoder function for that type (if the codecs system is applicable for the given type). The encoder and decoder functions of the codecs system are subsequently responsible to encode and decode, respectively, when invoked from the RTS.

The codecs systems are activated by registering an **initialization** function and a **setup** function by calling the `t3rt_codecs_register` function. The initialization function should make sure that the encoders and decoder are able to work, and the setup function must associate encoder and decoder function if applicable.

See Also

“RTL Codecs Functions” on page 275 in Chapter 7, *Runtime System APIs*

Encoder and Decoder Functions



The RTS defines one function prototype for the encoder function, and one for the decoder function. All associated encoders and decoders must have this prototype.

An encoder function will be called when a value is sent to the SUT. The encoder function is retrieved from the value’s type. The encoder function is given a binary data “container” that must be filled with the encoded data.

A decoder function will be called whenever received data have to be converted into a TTCN-3 RTS value, to be able to perform matching algorithms (as well as other operations like, for example, logging). Typically, a decoder function will be called when a `receive` operation is encountered, and a decode attempt must be made.

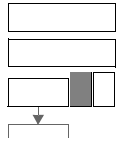
When decoding attempts are made, a decoder function must be able to handle data that it can not decode. For these cases it must report that the decoding failed, so that new attempts with other types's decode functions can be tried.

Please refer to "RTL Value Functions" on page 75 in Chapter 7, *Runtime System APIs* for information on how to access RTS value objects, and "RTL Type Functions" on page 49 in Chapter 7, *Runtime System APIs* for information on type information of how to access type information.

Note

The `"with encode"` *statement for encoding attributes of TTCN-3 types are accessed from the RTS type object (*`t3rt_type_t`*). This can be useful (or even necessary) to associate the correct encoder and decoder in the setup phase, and in the actual encoding and decoding.*

Registering a Codecs System



The intended location to register a codecs system is in the `t3ud_register_codecs` in the `t3ud_register_codecs.c` file. This file must be compiled and linked with the ETS. Doing so will "shadow" the empty `t3ud_register_codecs` function already compiled into the RTS.

Example 2: A Codecs System Embryo

```
/* t3ud_register_codecs.c
 *
 * This small example does not do any real encoding or
 * decoding. It sets up the encoder/decoder functions for
 * all types in the system. This can be done selectively
 * given, for example, the type's module name.
 */
#include "t3rts.h"


t3rt_codecs_result_t
ex_encode(t3rt_value_t value,
          t3rt_binary_string_t encoded_data,
          t3rt_context_t ctx)
{
  /* Perform encoding by filling in the binary
     data container*/
```

```
  return t3rt_codecs_result_succeeded_c;
}


t3rt_codecs_result_t
ex_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)
{
  *decoded_value = t3rt_type_instantiate(type, strategy,
ctx);
  /* Perform decoding */
  return t3rt_codecs_result_succeeded_c;
}


void
ex_codec_setup(t3rt_type_t type, t3rt_context_t ctx)
{
  t3rt_type_set_encoder(type, ex_encode, ctx);
  t3rt_type_set_decoder(type, ex_decode, ctx);
}


/* Registers user-provided codecs */
void
t3ud_register_codecs (t3rt_context_t ctx)
{
  /* This example doesn't have to do any initialization.
     This is why the init function is not registered.*/
  t3rt_codecs_register(NULL, ex_codec_setup, ctx);
}
```

6

Miscellaneous

Binary String Support

To satisfy the need for a binary data representation in the RTS for both encoding and decoding situations, as well as the representation of (component and port) addresses, the binary string support has been implemented.

The binary string type and functions are primarily intended to be used by encoders and decoders.

See Also

"RTL Binary String Functions" on page 261 in Chapter 7, *Runtime System APIs*

Wide String Support

To be able to internationalize the RTS and to provide a representation for the `universal charstring` type, the wide string type has been implemented. It is public since it has to be operated on by, for instance, log mechanisms.

See Also

"RTL Wide String Functions" on page 247 in Chapter 7, *Runtime System APIs*

7

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 302 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 3: 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 292

"t3rt_type_field_type" on page 59

"t3rt_type_template_base_type" on page 71

"t3rt_type_array_contained_type" on page 71

"t3rt_value_kind" on page 81

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 53

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 52

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 56

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 55

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 55

"t3rt_type_module, t3rt_type_definition_module" on page 56

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 275

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 275

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 275

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 275

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 52

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 97.

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 97.

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 58 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 50

"t3rt_value_copy" on page 76

"t3rt_value_assign_vector_element" on page 97

"t3rt_value_assign_string_element" on page 98

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 90

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 92

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 90

"t3rt_value_get_enum_name" on page 91

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 91

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 95

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 92

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 93

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 93

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 94

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 95

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 97

"t3rt_value_remove_vector_element" on page 107

"t3rt_value_vector_element" on page 87

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 106

"t3rt_value_assign_vector_element" on page 97

"t3rt_value_vector_element" on page 87

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 110.

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 58 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 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

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_status`	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_status`	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_status`	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 276 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 276 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 275 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 275). Doing so will cause a stack overflow.

Return Values

Compare with "t3rt_codecs_result_t" on page 275 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 275 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 275). Doing so that will cause a stack overflow.

Return Values

Compare with "t3rt_codecs_result_t" on page 275 for applicable values.

See also

"RTL Binary String Functions" on page 261

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 275 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 275 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 281 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 284.

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 288 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 287 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 288.

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 128 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 298.

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 4

```
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 triExecuteTestCase 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 triExecuteTestCase 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 charstring 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 milliseconds since midnight)
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

8

Glossary

A

Adaptation

Synonym: “Integration” on page 556

Analyze

To verify the syntactic and semantic correctness in selected test suites.

ASN.1

Abstract Syntax Notation 1.

An ITU standard – X.680-X.683, http://www.itu.int/ITU-T/study-groups/com07/asn1recs.html – commonly used to describe protocol data structures.

Asynchronous Communication

Synonym: “Message Based Communication” on page 557

ATS

Abstract Test Suite, a test specification written in an abstract notation (TTCN-3 for example) containing all definitions required to make an Executable Test Suite.

Synonym: “Test Suite” on page 564.

AST

Abstract Syntax Tree.

An Abstract Syntax Tree is the structural representation of the parser input, where each node represents the grammar terminals of the parsed code.

B

BER

Basic Encoding Rules. BER are the original rules for taking an ASN.1 data type, and turning it into a sequence of bits and bytes. BER uses a form of encoding commonly known as Tag-Length-Value. Each item is encoded as a tag, indicating what type it is, a length indicating the size of the object, and a value, which contains the actual contents of the object.

An ITU standard – X.690, http://www.itu.int/ITU-T/study-groups/com07/asn1recs.html – describes the ASN.1 basic encoding rules.

BNF

Backus-Naur Form, a context-free grammar defining the syntax of a language in terms of tokens and production rules.

See also: “EBNF” on page 554

Bookmark

Enables you to mark frequently accessed lines in your source file.

C

Codecs Systems

Conceptually, a set of encoder and decoder functions that implement certain encoding rules.

A codecs system is plugged into the runtime system during initialization to provide the necessary encoding and decoding services needed for any kind of communication.

Compiling

In Rational Systems Tester: Analyzing a test suite and generating ANSI-C code from it, using the options set in **Settings** on the **Project** menu.

Communication Port

TTCN-3 entity for communication between test components and SUT, or between test components. Ports must be defined as message-based, procedure-based or a mixture of the two.

Synonym: “Port” on page 559

Component Reference

Unique references to test components created during the execution of a test case including component type information.

Configuration

When you modify project settings (for analysis, code generation, execution, etc), they will be saved in the active configuration. A project may contain several configurations, associated with different settings.

Control Part

Synonym: “Module Control Part” on page 558

D

Data Types

There are four kinds of TTCN-3 data types: Basic types, Basic string types, User-defined structured types, and Special configuration types.

Synonym: Type.

See also: “Imported Data Types” on page 556

Decoding

Result of a conversion from a given transfer syntax to the tool specific value representation.

Definitions Part

Synonym: “Module Definitions Part” on page 558

DTD

Document Type Definition. Defines the legal building blocks of an XML document, and the document structure with a list of legal elements. A DTD can be declared within an XML document or in an external document, referenced from the XML document.

See also: “XML” on page 566

E

EBNF

The Extended Backus-Naur Form (ISO 14977, http://www.cl.cam.ac.uk/~mgk25/iso-ebnf.html).

The Extended BNF adds context sensitivity and regular expressions to BNF, making it more expressive.

See also: “BNF” on page 552

Editing Area

Area in which you can view and edit different files, such as TTCN-3 or HTML files for example.

Editor

There are two different editors available in Rational Systems Tester: one for TTCN-3 files with extended functionality, and one text editor for ASN.1 files, text files, and so on.

Encoding

Result of a conversion from a tool specific value representation to a given transfer syntax.

Entity

TTCN-3 language element such as types, variables, and so on.

Entity List

List of entities, displayed when you press CTRL + SPACEBAR in the TTCN-3 editor.

ETS

Executable Test Suite, the concrete application derived from the ATS.

See also: “Test Suite” on page 564

ETSI

European Telecommunications Standards Institute, http://www.etsi.org/.

ETSI is a non-profit organization whose mission is to produce telecommunications standards.

Event Channel

A conceptual entity in the runtime system, through which log events are distributed to active log mechanisms.

Execute Test

To execute test cases in an ETS.

Synonym: Run test

See also “ETS” on page 555

F

File View

View of the file structure of all files in a workspace.

Represented as a tab in the workspace window.

G

GCI Interface

Generic Compiler Interpreter Interface.

An API describing vendor and user specific responsibilities in the creation of TTCN-2 executable test suites.

Go To

Enables you to jump quickly to several different items, such as bookmarks or lines, in one or several TTCN-3 files.

Group

Different language elements can be grouped in the module definitions part. Groups may be nested, that is, groups can contain other groups.

All identifiers of declarations in a group, including nested groups, must be unique.

H

HTML

Hypertext Markup Language is the set of markup symbols or codes inserted in a file intended for display on a World Wide Web browser page.

I

ICS

Implementation Conformance Statement.

IDL

IDL (interface definition language) is a generic term for a language that lets a program or object written in one language communicate with another program written in an unknown language, ISO/IEC 14750:1999.

Imported Data Types

Types imported to a TTCN-3 module via the import statement.

Info Channel

A conceptual entity in the runtime system, through which text messages are distributed to active log mechanisms.

Integration

An implementation of the connection between the executable test suite and the SUT.

Synonym: Adaptation.

ITU

The International Telecommunication Union, http://www.itu.int/home/index.html, is an international organization within the United Nations System where governments and the private sector coordinate global telecom networks and services.

IUT

Implementation Under Test.

Synonym: SUT.

IXIT

Implementation eXtra Information for Testing.

J

K

L

Language Element

An entity in the language TTCN-3, for example module, data type, and test case.

M

Message Based Communication

The sending of a message is not related to the reception.

Synonym: “Asynchronous Communication” on page 551

Module

Top-level language element in TTCN-3. A module cannot be structured into sub-modules. A module can import definitions from other modules.

A TTCN-3 module can contain a definitions part and a control part. A test case is defined in the definitions part and called, executed, in the control part.

Module Control Part

Describes the execution order of the actual test cases.

Synonym: Control Part

Module Definitions Part

Contains all module definitions.

Synonym: Definitions Part

MSC

Message Sequence Charts.

An ITU standard – Z.120, http://www.itu.int/rec/recommendation.asp?type=items&lang=E&parent=T-REC-Z.120-199911-I – describing a graphical notation for the visualization of events in distributed systems.

MSC Viewer

Feature in Rational Systems Tester allowing you to use and view MSCs in your project.

MTC

Main Test Component.

Every test case contains one and only one MTC. Behavior defined in the body of a test case is the behavior of the MTC. The MTC type is defined in the test case header. An MTC is created automatically when a test case is executed.

N

O

Output Window

Window in which you can view logs, error messages, warnings, and so on.

Outline View

View of the structure of all language elements in a TTCN-3 file, displayed linearly.

Represented as an additional pane in the TTCN-3 editor.

P

PA

The Platform Adaptor is the part of the TRI implementation that is responsible for handling timers and external functions (that is, implements triStartTimer and so on). This is an entity that is provided by the customer.

PER

Packed Encoding Rules. PER use a different style of encoding than BER. Instead of using a generic style of encoding that encodes all types in a uniform way, the PER specialize the encoding based on the data type to generate much more compact representations.

An ITU standard – X.691, http://www.itu.int/ITU-T/studygroups/com17/languages/index.html – describes the ASN.1 packed encoding rules.

PICS

Protocol Implementation Conformance Statement.

PIXIT

Protocol Implementation eXtra Information for Testing.

Platform Layer

The adaptation implementation that needs to be implemented by the customer to provide necessary functionality for the platform dependent parts of the ETS execution

Port

Synonym: “Communication Port” on page 553

Procedure Based Communication

A remote procedure call, RPC, is performed between two components where the flow of control is transferred from the caller to the one being called (that is, the caller blocks until return of call) during execution of the call.

NOTE: Procedure based communication in TTCN-3 can be non-blocking as well. In this case the return value (if any) must be explicitly received by means of the `getreply` operation

Synonym: “Synchronous Communication” on page 562

Procedure Signature

Synonym: “Signature” on page 562

Project

A structured representation of all user files, such as TTCN-3, ASN.1, makefiles, and C files. A project is included in a workspace.

PTC

Parallel Test Component, a runtime entity executing parts of the testing behavior, communicating with the MTC and/or the SUT.

A PTC potentially runs on a different machine than the MTC and does not share memory with other components.

Q

R

Regular Expressions

A regular expression is an expression that defines a lexical string pattern in terms of a set of matching rules defined by a formal grammar.

Run Test

Synonym: “Execute Test” on page 555

Runtime Context

An opaque runtime system object that is passed around through the execution of the ETS in order to provide a necessary execution environment. Each component has its own runtime context object.

Runtime Engine

Synonym: “Runtime System” on page 561

Runtime Layer

Part of the runtime system API that provides necessary services to generated code and to an integration implementation.

Runtime System

Provides the operational functions needed by the ETS.

Synonym: Runtime Engine

Runtime System Objects

Entities that are instantiated during the execution of the ETS, which can be accessed by the different parts of an integration implementation.

S

SA

The System Adaptor is the part of the TRI implementation that is responsible for communication with the SUT (that is, implements triSend, etc). This is an entity that is provided by the customer.

SCC

Microsoft Common Source Code Control API.

Synonym: SCC API or SCCI

SDL

Specification and Description Language.

An ITU standard – Z.100, http://www.itu.int/rec/recommendation.asp?type=items&lang=E&parent=T-REC-Z.100-199303-S – describing a graphical notation for the specification and description of distributed real-time systems

Signature

Language element used to define the signature or format of function calls that are later used in testing that involves synchronous communication

Synonym: Procedure Signatures

Standard I/O

Standard input/output. The predefined input/output channels (stdin and stdout) that every process is initialized with.

Standard Toolbar

Displays a collection of easy-to-use buttons that represent well-known commands.

Status bar

Area in Rational Systems Tester where status and information, line number for example, is displayed.

stdout

Synonym: “Standard I/O” on page 562

Structured View

Hierarchical view of a test suite, sorted by language elements.

Represented as a tab in the workspace window.

SUT

System Under Test.

Synonym: “IUT” on page 557

Symbol Table

A statically generated symbol table that contains entries for the declared entities (for example types, constants, test cases, and so on) in the test suite. There is one symbol table generated for each TTCN-3 module.

Synchronous Communication

Synonym: “Procedure Based Communication” on page 560

T

Tcl

Tool Command Language

Tcl is an interpreted script language developed and maintained by Sun Laboratories. Tcl is comparable to JavaScript, Microsoft's Visual Basic, and similar. Used in the Script Wizard.

Templates

TTCN-3 construction that allows you to specify the instantiation of data in both synchronous and asynchronous communication.

In its simplest form a template can be compared to a value (send templates must contain real values). In a more advanced form, a template allows you to specify matching information for received data (only allowed for receive constraints). Templates can in its most powerful form also be parameterized to provide maximum expression power.

Test Case

Definition of the intended test, a defined limited set of actions performed to verify certain requirements set on the SUT.

In TTCN-3 terms, a test case is a special kind of function.

Test Component

Entity on which test behavior is executed. A test component often contains local declarations and a list of ports used by the component. Test components can be connected with other components and with a test system interface.

The Main Test Component (MTC) is created automatically when a test case starts, all other test components, that is, Parallel Test Components (PTC), are created explicitly by using the operation create.

Test Configuration

Consists of a set of inter-connected test components with well-defined communication ports and an explicit test system interface which defines the borders of the test system.

A test configuration can consist of one, and only one, MTC. At the start of each test case the test configuration is reset.

Test Schedule

List of test cases, and order in which they are executed.

Test Suite

The informal name for ATS or ETS. If the term denotes ATS or ETS depends on the context, but usually it is ATS.

Synonym: “ATS” on page 551

See also: “ETS” on page 555

TRI

TTCN-3 Runtime Interface.

An ETSI technical report, ETSI ES 201 873-5 V3.2.1, describing the vendor and user specific responsibilities in the creation of TTCN-3 executable test suites.

TSI

Test System Interface.

Test component that provides mapping of the ports available in the (abstract) TTCN-3 test system to those offered by a real test system.

TTCN-2

Tree and Tabular Combined Notation.

A formal language for specification of test cases. The latest version of TTCN is TTCN-3.

TTCN-3

Testing and Test Control Notation, the latest version of TTCN.

Type

Synonym: “Data Types” on page 553

U

UML

Unified Modeling Language.

A language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems.

V

Verdict

Outcome of the execution of a test. The verdict can have five different values: pass, fail, inconclusive, none, and error.

Pass: Expected behavior is correct.

Fail: Expected behavior is incorrect.

Inconclusive: Expected behavior is neither correct nor incorrect.

None: When a test component is instantiated, its local verdict object is created and set to this value. This verdict cannot be set by the user.

Error: A runtime error has occurred. This verdict cannot be set by the user.

Views

Graphical representation of data in Rational Systems Tester. In the workspace window, the structured view and the file view are displayed as tabs.

In the text editor, the outline view is available when editing TTCN-3.

W

Workspace

A container where you store and work with projects.

Workspace Window

Window in which the File View and Structured View facilitates the work with files, projects, and workspaces.

X

XML

Extensible Markup Language. XML is a simplified version of, and compatible with, SGML. The XML syntax is also similar to HTML, but while HTML describes how a page looks and acts, XML describes what the words in a document are.

An XML document that conforms to a DTD is referred to as a valid XML document. A well-formed XML document is an XML document that simply follows the XML syntax rules and no DTD is involved.

See also: “DTD” on page 554

Index of Functions

Index of Types

Index

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

X