Rational. SDL and TTCN Suite

TTCN Suite Getting Started

Getting Started

IBM Rational TTCN Suite 6.3

Getting Started

This edition applies to IBM Rational SDL Suite 6.3 and IBM Rational TTCN Suite 6.3 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, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions. Therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time

without notice.

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

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.

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

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

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

Acresso and FLEXnet are registered trademarks or trademarks of Acresso Software Inc.

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

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

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

Pentium is a trademark of Intel Corporation.

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

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

Introduction

About this Manual

This volume, *Getting Started*, contains a guide that will assist you in getting familiar with the TTCN Suite environment.

We start by giving an introduction to the TTCN Suite and the supported languages. Then follows a number of chapters with hands-on tutorials on how to use the tools included in the TTCN Suite, including different ways to build TTCN test suites from an SDL specification. These tutorials can be followed without any previous knowledge of the TTCN Suite.

The basic tutorials have separate chapters for the UNIX and Windows version of the TTCN Suite. The reason for this is the differences – in how to use the TTCN Suite and the TTCN Suite user interface – between the two versions.

All tutorials build on the knowledge gained in the previous tutorials, so you should practice them in the order they are included.

Documentation Overview

A general description of the documentation can be found in "Documentation" on page viii in the Release Guide.

Typographic Conventions

The typographic conventions that are used in the documentation are described in "Typographic Conventions" on page x in the Release Guide.

How to Contact Customer Support

Detailed contact information for IBM Rational Customer Support can be found in "How to Contact Customer Support" on page iv in the Release Guide.

Chapter

1

Introduction to Languages and Notations

This chapter describes the benefits of formal methods. It also gives a brief introduction to the TTCN, the ASN.1 and the MSC language.

Note that this chapter is not a tutorial on TTCN, ASN.1 or MSC. In Methodology Guidelines you can find more information about TTCN and how to use it.

If you want to know more about the languages supported in the SDL Suite, you should read chapter 1, *Introduction to Languages and Notations, in the Getting Started*.

Standardized Formal Methods

It is getting increasingly accepted within a steadily growing range of industrial segments that the only true way for software engineering to achieve higher quality, deliver on time and decrease development costs, is to use formal methods. Furthermore, as the international market grows, equipment from different manufacturers must be able to communicate with each other. Therefore it is obvious that the formal method to be used should be internationally standardized.

There are a number of different formal standardized languages and methods available today. The one you select, however, should fulfill a few more important requirements.

One is the availability of professional development tools. Another is clarity. The notation should be understood by a general audience, from experts to end users. Ideally it should have a graphical syntax.

Formal standardized graphical languages ...

- ... enforce precision during specification, since ambiguities and unclear statements are impossible to make.
- ... allow tool support. Tools can help to perform all kind of analyses like syntax/semantics check, simulation, test generation, code generation, etc.
- ... attract more than one tool-builder. The second-source possibility creates vendor independence with all its advantages.
- ... are more often the subject for courses, seminars, and text-books.
- ... are more likely to be maintained. As with natural languages, formal languages need to evolve to stay modern.
- ... promotes efficient verbal and written communication within development teams, between manufactures and between suppliers and customers, due to the conceptually formalized means of communication.
- ... have a graphical syntax that makes it simple and efficient to exchange information between different players within an organization.

The Test Suite Framework Standard

As the use of standards within the world of Information Technology and Telecommunications has increased tremendously during the last decade, so has the need for methods and tools that support the verification and validation of both the standards and their implementations.

This need has been addressed by ISO and CCITT (ITU-T) in the "Framework and Methodology for Conformance Testing of Implementations of OSI and CCITT Protocols". The framework has now reached the status of an International Standard as ISO/IEC 9646 (or X.290).

- The standard introduces the concept of *Abstract Test Suites* (consisting of *Abstract Test Cases*), a description of a set of tests that should be executed for a system. The tests should be described using a black-box model, i.e. only control and observe using the available external interfaces.
- The abstract tests are to be described using a formal language rather than using informal natural language. As part of the standard, the language TTCN is defined in order to describe the abstract tests.
- The possibility to copy the ASN.1 definitions from the protocol specification into the test suite in TTCN assures consistency between the information transferred in system specification and the test specification.

Conformance Testing

Conformance testing is the process of verifying that an implementation performs in accordance with a particular standard/specification/environment.

Conformance testing is exclusively concerned with the external behavior of an implementation. Service and functional behavior is tested in order to find logical errors and prerequisites for interoperability. Conformance testing is not intended to be exhaustive and a successfully passed test suite does not imply a 100% guarantee. But it does ensure, with a reasonable degree of confidence, that the implementation is consistent with its specifications, and it does increase the probability that implementations will interwork.

System Testing

- *Conformance testing* verifies whether an implementation performs according to the stated standard/specification/environment.
- *Interoperability testing* checks the ability of different implementations to interact in a prescribed manner, achieving predictable results.
- *Regression testing* is performed after functional improvements or corrections, to confirm that nothing unintentional has been introduced.

The TTCN Language

TTCN (Tree and Tabular Combined Notation, ISO/IEC 9646-3) is a language standardized by ISO for the specification of tests for real-time and communicating systems. TTCN has been developed within the framework of standardized conformance testing (ISO/ IEC 9646).

With TTCN a test suite is specified. A test suite is a collection of various test cases together with all the declarations and components it needs. Each test case is described as an event tree. In this tree, behaviors such as “First, we send A, then either B or C is received; if it was B we will send D...” are described. Concurrent TTCN allows several event trees to run concurrently.

TTCN is abstract in the sense of being test system independent. This means that a test suite in TTCN for one application (e.g. protocol, system, etc.) can be used in any test environment for that application.

The use of TTCN has increased tremendously during the last few years. This has been augmented by the significant amount of test suites released by various standardization bodies. TTCN is not only used in standardization work. The language is very suitable for all kinds of functional testing for real-time and communicating systems. This has led to a wide usage throughout the industry.

The specifications of the messages being sent and received can be defined using either the native form of TTCN or by using ASN.1 (Abstract Syntax Notation One).

Theoretical Model

A TTCN specification describes an abstract test suite (ATS) that is independent of test system, hardware and software. The ATS defines the test of the *implementation under test* (IUT), which is treated in a black box model, i.e. only its exterior interface is of concern. The IUT is stimulated by sequences of test events and its response is inspected.

A TTCN abstract test suite can be transformed into an *executable test suite* (ETS) using the TTCN Suite. This ETS is downloaded into the test system (the system performing the test).

The test system performs the test by executing the ETS against the *system under test* (SUT) which contains the implementation under test.

During execution the ETS will report any errors and log events for on-line or post-test evaluation.

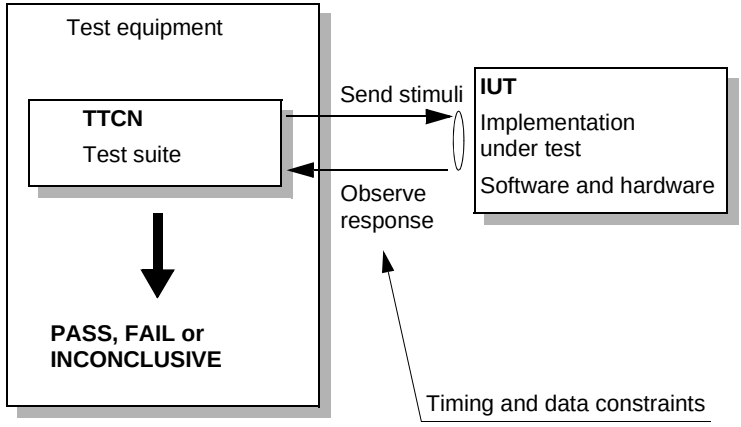


Figure 1: Test System with Executable Test Suite (ETS) connected to the system under test (SUT)

TTCN Specification Structure

A TTCN specification is similar to a Pascal or C program. (Being cleaner and more comprehensive, TTCN is easier to learn.) Just like Pascal and C, TTCN requires type and data declarations and it uses concepts like modules and subroutines. Since TTCN is designed for testing, it contains test specific concepts such as:

- Powerful pattern matching constructs for complex data structures using both TTCN and ASN.1.
- Verdicts and preliminary verdicts to define the outcome of test cases.
- Possibilities to handle alternative outcomes in a test case.
- Pre-ambles and post-ambles to show how to compose test cases.
- The “modules” concept supporting multi-user test development.
- The “modules” concept supporting the re-use of test components and data structures.
- Constructs for parallel test component execution including synchronization primitives.

A TTCN specification has a standardized layout that produces comprehensive and unambiguous paper printouts. This greatly improves clarity and readability. A test suite is divided into the following four major parts:

- The *overview part*, containing a table of contents and a description of the test suite. Its purpose is mainly to document the test suite to increase clarity and readability.
- The *declarations part*, declaring all messages, variables, timers, data structures and black box interface towards the Implementation Under Test.
- The *constraints part*, assigning values and creating constraints for inspection of responses from the implementation under test.
- The *dynamic part*, containing all test cases, test steps and default tables with test events and verdicts, i.e. it describes the actual execution behavior of the test suite.

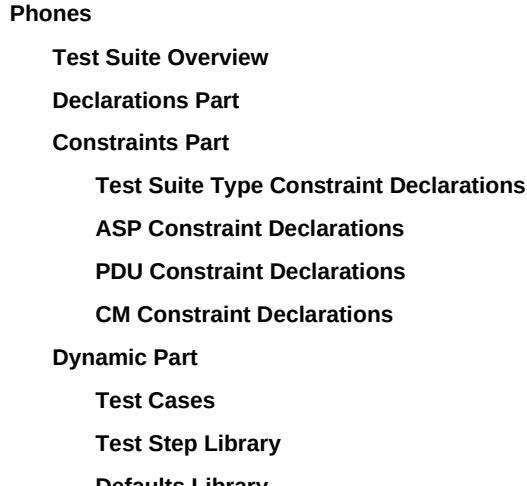


Figure 2: The basic structure of a TTCN Test Suite

Test Suite Dynamic Structure

The dynamic part of a TTCN abstract test suite is created in a hierarchical and nested manner. The building blocks are test groups, test cases, test steps and test events. There are no limitations as to how many test groups may be contained in a test suite, how many test events may be contained in a test step, etc.

Test component explanation:

- *Test event:* The smallest, indivisible unit of a test suite. Typically, it corresponds to a signal, interrupt, message, data or timer expiration.
- *Test step:* A grouping of test events, similar to a subroutine or procedure in other programming languages.
- *Test case:* The main fundamental building block in a test suite. A test case tests a particular feature or function in the implementation under test (IUT). A test case has an identified test purpose and it assigns a verdict that depends on the outcome of the test case.
- *Test group:* A grouping of test cases. It might for example be convenient to group all test cases concerning connection establishment, and to put all test cases concerning transport into a separate test group.
- *Test suite dynamic part*: The highest level, encompassing all test components and serving as the root of the tree. A test suite can range from a large number of test groups and test cases to a single test event contained in a test case.

Dynamic Part

Test Cases

BasicCall

CallW

Test Step Library

ConnectPhones

TestCallWaiting

HangUpAllPhones

Defaults Library

Figure 3: The TTCN dynamic part structure

Communication Mechanisms

TTCN uses the concepts of *points of control and observation* (PCOs), *abstract service primitives* (ASPs) and *protocol data units* (PDUs) in order to create an abstract interface towards the implementation under test (IUT). A PCO is a point in the abstract interface where the IUT can be stimulated and its responses can be inspected. An ASP or a PDU is either a stimuli or a response that carries information, i.e. parameters and data.

Each PCO has two *first in first out* queues for temporary storage of ASPs and PDUs: One queue for send and one queue for receive. These queues are infinite, i.e. they can store any number of ASPs and PDUs.

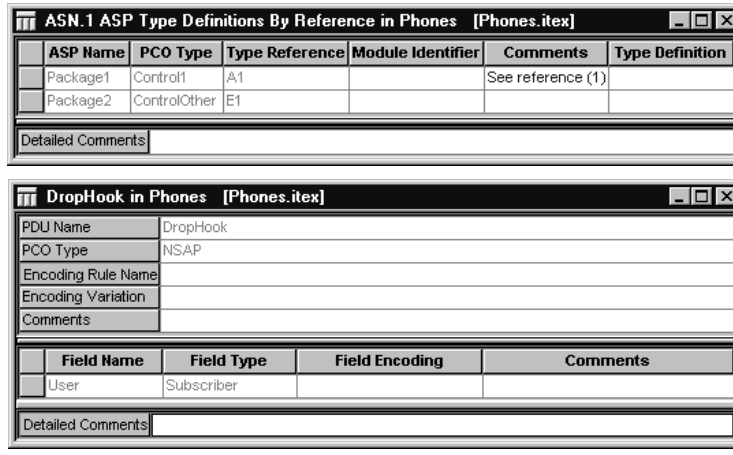


Figure 4: PCOs together with ASPs and PDUs create an abstract interface towards the IUT

Event Trees, Constraints and Verdicts

TTCN uses event trees with test events to express the behavior of test steps and test cases.

All the leaves in the event tree are assigned a verdict that can be PASS, FAIL or INCONCLUSIVE. PASS means that the test case completed without detecting any error. FAIL means that an error was detected, that is, the behavior of the IUT did not conform with the pre-defined specification. INCONCLUSIVE means that there was insufficient evidence

for a conclusive verdict to be assigned, but that the behavior of the IUT was valid.

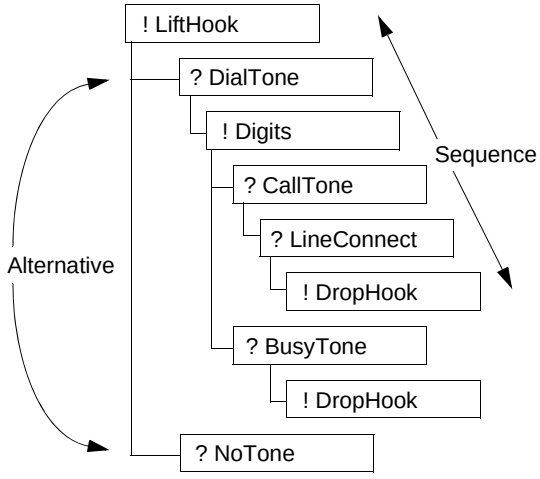


Figure 5: An event tree and the corresponding execution order

A verdict can be either preliminary or final, allowing for flexibility in the specification. A final verdict will terminate the active test case and return its verdict. A preliminary verdict will not terminate test case execution but it will flag either PASS, FAIL or INCONCLUSIVE. This preliminary verdict can be inspected during test execution, like any variable.

TTCN uses the comprehensive format shown in Figure 6. The indentation level of statements in the Behaviour Description column indicates where in the event tree an event belongs. The leaves of the tree hold verdicts that define test case outcome.

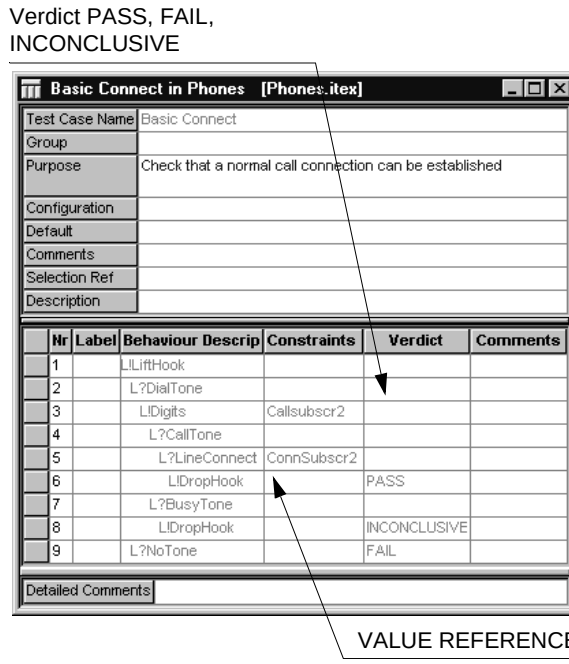


Figure 6: The TTCN test case corresponding to the event tree in Figure 5

Data and Value Model

The TTCN data and value model is somewhat different from the one in traditional programming languages. It allows for the creation of complex data structures and types, and has the concept of constraints to do value assignments. Constraints are more powerful than values in that they also allow the use of patterns. A pattern can contain wild cards and define allowable value ranges for complex data structures. This is very useful when inspecting responses from the implementation under test.

TTCN has two alternative data and constraint representation formats: the TTCN native tabular form and ASN.1 (Abstract Syntax Notation One). ASN.1, which is a purely textual notation, provides a more flexible platform for describing complex data structures. ASN.1 also allows data descriptions to be shared between an SDL (Z.105) specification and a TTCN test suite.

Modular TTCN

With modular TTCN, it is possible to define test suite components for re-use. This facilitates test component re-use, and provides a language platform for multi-user test development projects. See also "Distributed Development" on page 118 in chapter 1, *The TTCN Introduction, in the Methodology Guidelines*.

Concurrent TTCN

Concurrent TTCN introduces a parallel architecture for simultaneous execution of several test components, allowing many interfaces to be tested concurrently. There are several benefits:

- Test components become more cohesive/modular since they focus on the test of a specific interface of the IUT. Each interface is isolated in specific test components.
- Module and integration testing is easier and it facilitates test component re-use.
- Test suite maintenance becomes easier since test components are less monolithic. If one interface of the IUT changes, it will not influence any test components but the ones specifically dedicated to this interface.
- Each test component becomes smaller and simpler since it deals with fewer alternatives.

In concurrent TTCN, each test case consists of several *parallel test components* (PTCs) that execute autonomously, performing concurrent tests. A *master test component* (MTC) starts the execution of the PTCs and controls the final verdict. The PTCs can only set preliminary verdicts and the test case is completed and its verdict is decided when all the PTCs are finished. The PTCs are synchronized through *co-ordination points* (CPs) and *co-ordination messages* (CMs).

Graphical and Textual Notations

The TTCN language supports two notations that are equivalent. The graphical notation (TTCN-GR) and a textual notation (TTCN-MP).

Application Areas

Currently, TTCN is mainly known within the telecommunications industry. However, it has broader areas of application, which can be summarized as follows:

- Any protocol conformance testing.
- Any communicating systems testing, e.g. interactive, message-driven, real-time or distributed.
- Any system with a well defined interface that can be stimulated and observed.

The ASN.1 Language

Abstract Syntax Notation One is a language specifically designed for describing structured information that is conveyed across some interface or communication medium. ASN.1 is standardized internationally (ISO/IEC 8824) and it is a key ingredient of Open Systems Interconnection (OSI).

In the presentation layer of the OSI hierarchy, data values of quite complex types, such as character strings, intricate structures or arrays of values, need to be determined in a unique way without saying anything about the representation. ASN.1 is developed to fill this need.

ASN.1 is a generic notation for the specification of data types and values. The basic principle is to define a small number of simple types by defining their possible values, and give rules for combining these into increasingly complicated types. The original use of ASN.1 was in the information description of high-level protocols (FTAM, CMIP, MHS etc.), but today it is widely used in the telecommunications industry for protocols and applications.

```
AtmInterfaceTCEntry     ::= SEQUENCE {
atmInterfaceOCDEvents       Counter32,
atmInterfaceTCAlarmState    INTEGER
}
```

Figure 7: A sample ASN.1 type definition

ASN.1 Encoding and Transfer Syntax

ASN.1 requires a transfer syntax in order to pass data between two entities. *Basic encoding rules* (BER, ISO 8825) is a standardized transfer syntax of OSI. Others exist as well: *canonical encoding rules* (CER) for security applications, *distinguished encoding rules* (DER) for digital signatures, traditional C/C++, etc. Any transfer syntax can be used for ASN.1 descriptions.

ASN.1, SDL and TTCN – a Powerful Combination

TTCN includes ASN.1, i.e. ASN.1 is used for creating data descriptions and constraints in test suite specifications. Through the new standard of Z.105, ASN.1 is merged with SDL (Specification and Description Language) to create an extremely powerful language environment for specification of real-time, interactive and distributed systems.

Data descriptions made in ASN.1 can be used for both SDL and TTCN specifications, thus making a tight integration between implementation and test, and promoting re-use.

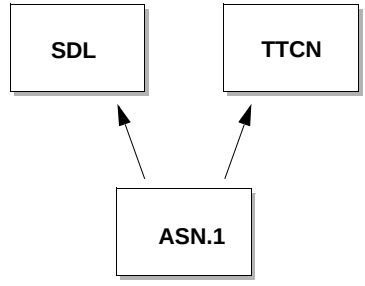


Figure 8: ASN.1 specifications can be shared between SDL and TTCN specifications

The Message Sequence Chart Language

History

During the last years, ITU has made a considerable effort in standardizing a formal language that defines *message sequence charts* (MSCs). A

first version of the MSC recommendation was published in the summer of 1992.

As defined in the recommendation Z.120, the MSC language offers a powerful complement to SDL in describing the dynamic behavior of an SDL system. Its graphical representation is well suited for presenting a complex dynamic behavior in a clear and unambiguous way that is easy to understand.

Theoretical Model

An MSC describes one or more traces from one node to another node of an abstract communication tree generated from an SDL specification. Basically, the information interchange is carried out by sending messages from one instance to another. In an SDL specification, those messages would coincide with the signals that are sent from one process and consumed in another process. The instances would correspond to any part of the specification (an SDL system, a block or a process).

Graphical and Textual Notations

The MSC language supports two notations that are equivalent. Beside the graphical notation (MSC/GR), a textual notation (MSC/PR) is standardized.

Application Areas

Among several application areas, we have selected the following:

- Producing documents with the purpose of defining the requirements on a system.
- Facilitating the design phase, by identifying and documenting a multitude of dynamic cases before starting designing with SDL.
- Presenting the execution of a simulation as a graphical output. This output is easy to follow and can later be verified against a reference. MSCs can be verified against an SDL system using the SDL Suite.
- Presenting the execution trace of an SDL system during an interactive simulation and generation of reports.
- A convenient way to define test purposes, particularly in conjunction with the Autolink test generation features.

Chapter 1 Introduction to Languages and Notations

Chapter

2

Introduction to the TTCN Suite (in Windows)

This chapter gives an introduction to the TTCN Suite and its functionality. It also includes a list of file types supported by the TTCN Suite.

When you have read this chapter and want to get more familiar with the TTCN Suite, you should read and practice chapter 4, *Tutorial: TTCN Suite Basics (in Windows)*.

If you want to know more about what is new in this release of the TTCN Suite, you should read chapter 2, *Release Notes, in the Release Guide*.

> **Note: Windows version**
>
> **This is the Windows version of the chapter. The UNIX version is chapter 3, *Introduction to the TTCN Suite (on UNIX)*.**

Introduction to the TTCN Suite

The TTCN Suite is a family of tools for the development, specification and compilation of executable tests for the TTCN language. You can find more information about the TTCN Suite tools in "The TTCN Suite Toolset" on page 19.

With the TTCN Suite and TTCN, tests are specified in a way that is independent of implementation software and hardware, that is, the tests are only based on the system to be tested and not on the system performing the test. This has encouraged standards organizations to use the technology. Test suites for various standards, supplied by standardization authorities such as ETSI, can therefore be used by all manufacturers and users of that standard. The test suites are created in a graphical editor environment and checked for syntactic and semantic errors by means of a TTCN analyzer.

One of the TTCN Suite tools is TTCN Access. It is a general tool for developing translators, code generators for different languages, automatic encoders and decoders, interpreters, reporters and analysis tools for TTCN test specifications.

The TTCN to C compiler produces code for both concurrent and non-concurrent TTCN as well as a limited subset of ASN.1 (see "TTCN ASN.1 BER Encoding/Decoding" on page 40 in chapter 1, *Compatibility Notes, in the Release Guide* for further details on the restrictions that apply). To complete the production of an ETS, the generated code must be adapted to the specific target environment.

The generated C code forms an ETS together with the Generic Compiler Interpreter (GCI) Interface. The GCI Interface standardizes the communication between a TTCN component supplied by a vendor and other test system components supplied by the customer, thus forming a Means Of Testing (MOT). GCI is a minimal interface for the adaptation of the generated code to a specific target environment. It makes ETS adaptation easy.

The TTCN Suite Toolset

The TTCN Suite is an integrated set of tools used for creating, editing and maintaining TTCN documents. The TTCN Suite will be described below.

The Organizer

You start the TTCN Suite by adding and opening TTCN documents in the *Organizer*. The Organizer is the main window in TTCN Suite. It integrates and co-ordinates the individual tools as required. Several tools may be used simultaneously through the Organizer. For instance, one part of a design may be analyzed at the same time as another is edited.

The Organizer features a graphical view of all the diagrams and documents you are working with, making them an integrated part of the development process. This may include TTCN documents, SDL hierarchies, Messages Sequence Charts, Object Model diagrams, State Charts, Highlevel MSCs and text documents. The view may be freely organized according to your preference.

Different Views of the Test Suite

You can view and edit the contents of a TTCN test suite in the *Browser*, the *Table Editor* and the *Finder*. The Browser gives information about the overall structure while the Table Editor is used for viewing and editing the contents of the TTCN tables. The Finder is used for displaying parts of the test suite, based on different search criteria. Some operations in the TTCN Suite produce logs and they are displayed in the *Log Manager*.

The Browser

The Browser presents the view of the overall structure of a test suite. You can manipulate the test suite by adding and deleting items. It is also possible to control the amount of information displayed in the Browser by collapsing and expanding the test suite.

Several Browsers, displaying different test suites, can be opened at the same time. You may also find it useful to create a sub Browser, in which you can select to display only a part of the test suite.

The Table Editor

When you double-click a table that is included in a test suite, it will be opened in the Table Editor. In the Table Editor you can edit the standardized TTCN-GR tables, including the concurrent TTCN tables.

The Finder

By using the Finder, you can display the TTCN tables in a list, without any ordering restrictions. You can sort out the amount of tables included in the list by different criteria, for example name, type, modification time and analysis status. The tables can also be opened and analyzed from the Finder.

The Log Manager

The Log Manager presents information about different operations in the TTCN Suite. For example, log information will be generated when you analyze, simulate or export a file.

Building a Test Suite

You can edit the declarations, constraints and dynamic tables manually in the Table Editor, but there are also other ways for editing which reduce the manual work and the risk of typing-errors:

Data Dictionary

An alternative to manually writing behavior lines is to use the *Data Dictionary*. By using the Data Dictionary, you can select system components – for example PCOs, types, constraints and timers – that are already declared. You can then use these to build behavior statements.

SDL to TTCN Link

SDL to TTCN Link is similar in use and appearance to the Data Dictionary. The difference is that with SDL to TTCN Link, you generate TTCN declarations and interactively build behavior tables based on an SDL specification.

Autolink

Autolink supports the automatic generation of declarations, constraints and dynamic behavior tables in a TTCN test suite. The test generation is based on an SDL specification and a number of MSC diagrams. You may create these MSCs manually in the MSC Editor or automatically in

the SDL Simulator or the SDL Explorer. The output from Autolink is a test suite in MP file format which can be opened in the TTCN Suite and refined subsequently.

In general, it is easier to describe test cases by MSCs than to specify them in TTCN directly. Moreover, Autolink provides a number of facilities to control the format and enhance the readability of a generated test suite.

Generation of the Test Suite Overview

The *test suite overview tables* includes the test suite structure, test case index and default index tables. These tables only need to be generated once, for example when the test suite is to be printed or exported. After that, the overview will be updated automatically as soon as you edit the test suite.

Analyzing, Verifying and Executing a Test

The Analyzer

The *Analyzer* performs a complete syntax check on TTCN and ASN.1 (as used in TTCN), as well as a number of static semantic checks, focusing on the existence and uniqueness of identifiers and the way they are being used.

It is possible to analyze the entire Browser structure or selected parts of it, as well as an entire modular TTCN system. If any tables are found to be erroneous after analysis, the names of those tables will be displayed in the Log Manager. An easy way to find and open an erroneous table, is to left-click the table name in the log and then `<Ctrl>`-right-click the table name again.

The TTCN to C Compiler

The *TTCN to C compiler* translates TTCN to ANSI-C. The TTCN to C compiler reduces the time for code generation as well as the volume of the generated code. Automatically generated code is also free from the potential errors of manual coding. The generated code is independent of the target operative system and protocol. In order to be executable, the generated C code must be linked to a library containing the pertinent OS functions.

The Generic Compiler Interpreter Interface

The *Generic Compiler Interpreter* interface (GCI) is an interface for the adaptation of the generated code to a specific target environment. The GCI interface focuses on what an abstract test suite needs in order to execute in terms of functionality, and on what is needed to integrate TTCN with a larger system.

The SDL and TTCN Integrated Simulator

The *SDL and TTCN Integrated Simulator* allows execution of a TTCN test suite in a host environment. The system under test consists of a simulated SDL system. When the SDL and TTCN Integrated Simulator is connected with the SDL Simulator, the combined system is simulated. The test cases are executed one by one or in batch. After the execution, the test coverage information can be visualized by the SDL Coverage Viewer.

TTCN Access

TTCN Access makes the internal data structures representing the test suite, available for application programs. It can be seen as a platform for writing applications related to an abstract test suite, for example code generators, interpreters, report tools and analysis tools.

Access provides a default mechanism for accessing the information, but gives you total freedom to override this for one better fitted to your needs.

Input and Output Formats

The TTCN language supports two notations that are equivalent: TTCN-GR – the graphical notation – and TTCN-MP – the textual notation.

TTCN-GR

TTCN-GR is the format used when you edit a test suite in the TTCN Suite. The TTCN Suite also prints a TTCN document in the TTCN-GR format according to ISO/IEC 9646-3. You may select to print an entire test suite or just parts of it. A preview feature allows you to see what the printed material will look like before printing. The document may be printed as a hard copy or a PostScript file.

TTCN-MP

It is possible to export TTCN documents to TTCN-MP format. Selected items in the Browser – a single table or sets of tables – may also be exported to TTCN-MP format. The test suite that is to be exported, does not necessarily have to be correct, analyzed or complete.

It is also possible to open a document in TTCN-MP format. There is a high tolerance for errors in the MP file and you can use the Analyzer for finding the errors and then correct them in the TTCN Suite rather than in the original MP file.

Files in the TTCN Suite

File Types

The TTCN Suite supports a number of different file types with various suffixes:

Filename	Explanation
`filename.itex`	TTCN document main data base file
`filename.itex#0`	TTCN document revert file
`filenJAAa002l89,s`	TTCN document working structure file
`filenJAAa002l89,t`	TTCN document working table file
`filename.itex-read_lock`	TTCN document read lock file
`filename.itex-lock`	TTCN document lock file
`filename.mp`	TTCN-MP file
`filename.log`	TTCN Suite log file
`filename.c`	ANSI-C file generated by the TTCN to C compiler
`filename.h`	Header file generated by the TTCN to C compiler

The working *structure* and *table* files are the internal TTCN Suite files and will be created temporarily in the temporary directory.The TTCN Suite checkpoints automatically to these files during editing but the main data base files are updated only on save.

The revert file is created in the session directory (which is the same as the target directory unless the `ITEX_SESSION_DIR` environment variable is set). It has a name consisting of the name of the main data base file (including extension), a hash sign (#) and a sequence number. The revert file is always stored in the TTCN-GR format even if it has a name like `filename.mp#0`.

Locking of TTCN Documents

When a TTCN document is opened in the TTCN Suite Browser, a read-lock file for it (which has the additional suffix '-read_lock') is automat-

ically created. The lock file is created in the same directory as the document. When the TTCN Suite Browser exits, the read-lock file is removed. If any another user (or the same user but using another TTCN Browser) try to open the same TTCN document, he will get a question informing that the file is opened by another user and asking to open it in read-only mode. The command-line TTCN2C compiler and other access applications always open TTCN documents in read-only mode. When the TTCN document is opened in read-only mode, no read-lock file is be created.

When a TTCN document is saved, a lock file for it (which has the additional suffix `-lock`) is automatically created. This means that no other user can access it during the save operation. The lock files are created in the same directory as where the document is saved.

Transferring TTCN Suite TTCN Documents

The entire TTCN Suite data bases may be transferred between storage areas. This is useful in order to preserve information that is lost in the TTCN-MP format. Use the usual file commands to achieve this.

Chapter

3

Introduction to the TTCN Suite (on UNIX)

This chapter gives an introduction to the TTCN Suite and its functionality. It also includes a list of file types supported by the TTCN Suite.

When you have read this chapter and want to get more familiar with the TTCN Suite, you should read and practice chapter 5, *Tutorial: TTCN Suite Basics (on UNIX)*.

If you want to know more about what is new in this release of the TTCN Suite, you should read chapter 2, *Release Notes, in the Release Guide*.

> **Note: UNIX version**
>
> **This is the UNIX version of the chapter. The Windows version is chapter 2, *Introduction to the TTCN Suite (in Windows)*.**

Introduction to the TTCN Suite

The TTCN Suite is a family of tools for the development, specification and compilation of executable tests for the TTCN language. You can find more information about the TTCN Suite tools in "The TTCN Suite Toolset" on page 29.

With the TTCN Suite and TTCN, tests are specified in a way that is independent of implementation software and hardware, that is, the tests are only based on the system to be tested and not on the system performing the test. This has encouraged standards organizations to use the technology. Test suites for various standards, supplied by standardization authorities such as ETSI, can therefore be used by all manufacturers and users of that standard. The test suites are created in a graphical editor environment and checked for syntactic and semantic errors by means of a TTCN analyzer.

One of the TTCN Suite tools is TTCN Access. It is a general tool for developing translators, code generators for different languages, automatic encoders and decoders, interpreters, reporters and analysis tools for TTCN test specifications.

The TTCN to C compiler produces code for both concurrent and non-concurrent TTCN as well as a limited subset of ASN.1 (see "TTCN ASN.1 BER Encoding/Decoding" on page 40 in chapter 1, *Compatibility Notes, in the Release Guide* for further details on the restrictions that apply). To complete the production of an ETS, the generated code must be adapted to the specific target environment.

The generated C code forms an ETS together with the Generic Compiler Interpreter (GCI) Interface. The GCI Interface standardizes the communication between a TTCN component supplied by a vendor and other test system components supplied by the customer, thus forming a Means Of Testing (MOT). GCI is a minimal interface for the adaptation of the generated code to a specific target environment. It makes ETS adaptation easy.

The TTCN Suite Toolset

The TTCN Suite is an integrated set of tools used for creating, editing and maintaining TTCN documents. The tools and functionality of the TTCN Suite will be described below.

The Organizer

You start the TTCN Suite by adding and opening TTCN documents in the *Organizer*. The Organizer is the main window in TTCN Suite. It integrates and co-ordinates the individual tools as required. Several tools may be used simultaneously through the Organizer. For instance, one part of a design may be analyzed at the same time as another is edited.

The Organizer features a graphical view of all the diagrams and documents you are working with, making them an integrated part of the development process. This may include TTCN documents, SDL hierarchies, Messages Sequence Charts, Object Model diagrams, State Charts, Highlevel MSCs and text documents. The view may be freely organized according to your preference.

Different Views of the Test Suite

You can view and edit the contents of a TTCN test suite in the *Browser* and the *Table Editor*. The Browser gives information about the overall structure while the Table Editor is used for editing the contents of the TTCN tables.

The Browser

The Browser presents the view of the overall structure of a test suite. You can manipulate the test suite by adding and deleting items. It is also possible to control the amount of information displayed in the Browser by collapsing and expanding the test suite.

Several Browsers, displaying different test suites, can be opened at the same time. You may also find it useful to create a sub Browser, in which you can select to display only a part of the test suite.

The Table Editor

When you double-click a table that is included in a test suite, it will be opened in the Table Editor. In the Table Editor you can edit the standardized TTCN-GR tables, including the concurrent TTCN tables.

Building a Test Suite

You can edit the declarations, constraints and dynamic tables manually in the Table Editor, and the menu choice *Replace* allows searching for and replacing of test strings. There are also other ways for editing which reduces the manual work and the risk of typing-errors: you can use the *Data Dictionary*, the SDL to TTCN Link and the *Autolink*.

The test suite overview tables can be generated by the menu choice *Generate SO*.

Data Dictionary

An alternative to manually writing behavior lines is to use the Data Dictionary. The Data Dictionary is included in the Table Editor, and by using it you can select system components – for example PCOs, types, constraints and timers – that are already declared, to build behavior statements.

SDL to TTCN Link

SDL to TTCN Link is similar in use and appearance to the Data Dictionary. The difference is that with SDL to TTCN Link, you generate TTCN declarations and interactively build behavior tables based on an SDL specification. This ensures consistency between the test suite and the specification.

Autolink

Autolink supports the automatic generation of declarations, constraints and dynamic behavior tables in a TTCN test suite. The test generation is based on an SDL specification and a number of MSC diagrams. You may create these MSCs manually in the MSC Editor or automatically in the SDL Simulator or SDL Explorer. The output from Autolink is a test suite in MP file format which can be imported into the TTCN Suite and refined subsequently.

In general, it is easier to describe test cases by MSCs than to specify them in TTCN directly. Moreover, Autolink provides a number of facilities to control the format and enhance the readability of a generated test suite.

Generate Overview

To complete the test suite, you have to generate the *test suite overview tables*, that is, the test suite structure, test case index and default index tables. The page numbers of the tables referenced in the index tables will also be calculated. No selection is required when you generate the overview and you can do it both from the Browser and the Table Editor.

Other Test Suite Operations

The following operations can be performed on the entire test suite or selected parts of it:

Select

You can create a selection in the Browser by specifying various predicates. They could be name or content patterns, cross reference predicates, type information or analysis results. The selections may later be used for presenting status information on the items selected. It is also possible to create sub Browsers from selections.

Report

It is possible to produce a configurable textual report – status information as analysis status or modification date – on a test suite. The output may also be used for further processing in an awk script or Microsoft Excel.

Compare

You may compare two TTCN documents. This is often useful when you want to resolve conflicts before merging one of the documents into the other. When you want to compare two documents, you make selections in one test suite. These are then compared with the other test suite – the destination test suite. Any similarities found will be logged and reported.

Merge

You can merge an entire TTCN document or selected parts of it into another TTCN document. This only works if there is no conflict, that is, no object in the source test suite has the same name as an object in the destination test suite.

It is also possible to merge the contents of an MP file into a TTCN document.

Search

You may search for occurrences of a specified pattern, which can be either a literal string, wildcard or regular expression. If you want to, you can also replace the matches by another pattern. It is possible to search both from the Browser and the Table Editor and to select, for example, the scope and direction for the search.

Find Tables

You can search for and display named tables. No selection is required and you can to it both from the Browser and the Table Editor.

Analyzing, Verifying and Executing a Test

The Analyzer

The *Analyzer* performs a complete syntax check on TTCN and ASN.1 (as used in TTCN), as well as a number of static semantic checks, focusing on the existence and uniqueness of identifiers and the way they are being used.

When the Analyzer is started from a Browser, it will analyze selected items in the test suite. When the Analyzer is started from a Table Editor it will only analyze that table.

If any tables are found to be erroneous after analysis, the names of those tables will be displayed in the log. An easy way to find and open an erroneous table is to select the name of the table in the log and then select *Find Table* from the *Tools* menu in the Browser or the Table Editor.

The TTCN to C Compiler

The *TTCN to C compiler* translates TTCN to ANSI-C. The TTCN to C compiler reduces the time for code generation as well as the volume of

the generated code. Automatically generated code is also free from the potential errors of manual coding. The generated code is independent of the target operative system and protocol. In order to be executable, the generated C code must be linked to a library containing the pertinent OS functions.

The Generic Compiler Interpreter Interface

The *Generic Compiler Interpreter interface* (GCI) is an interface for the adaptation of the generated code to a specific target environment. The GCI interface focuses on what an abstract test suite needs in order to execute in terms of functionality, and on what is needed to integrate TTCN with a larger system.

The SDL and TTCN Integrated Simulator

The *SDL and TTCN Integrated Simulator* allows execution of a TTCN test suite in a host environment. The system under test consists of a simulated SDL system. When the SDL and TTCN Integrated Simulator is connected with the SDL Simulator, the combined system is simulated. The test cases are executed one by one or in batch. After the execution, the test coverage information can be visualized by the SDL Coverage Viewer.

TTCN Access

TTCN Access makes the internal data structures representing the test suite, available for application programs. It can be seen as a platform for writing applications related to an abstract test suite, for example code generators, interpreters, report tools and analysis tools.

Access provides a default mechanism for accessing the information, but gives you total freedom to override this for one better fitted to your needs.

Input and Output Formats

The TTCN language supports two notations that are equivalent: TTCN-GR – the graphical notation – and TTCN-MP – the textual notation.

TTCN-GR

TTCN-GR is the format used when you edit a test suite in the TTCN Suite. The TTCN Suite prints a TTCN document in the TTCN-GR format according to ISO/IEC 9646-3.

When you print, configurable headers and footers, double or single sided printing and partial prints are supported. The output format is PostScript, which also makes it possible to print the test suite to a file.

TTCN-MP

It is possible to export TTCN documents to TTCN-MP format. Selected items in the Browser – a single table or sets of tables – may also be exported to TTCN-MP format. The test suite that is to be exported, does not necessarily have to be correct, analyzed or complete.

It is also possible to import a document in TTCN-MP format. There is a high tolerance for errors in the MP file and you can use the Analyzer for finding the errors and then correct them in the TTCN Suite rather than in the original MP file.

In addition, you can simply open MP files and work with them as with TTCN-GR files without having to convert them first. It is also possible to use *Save As* for changing the format.

Files in the TTCN Suite

File Types

The TTCN Suite supports a number of different file types with various suffixes:

Filename	Explanation
`filename.itex`	TTCN document main data base file
`filename.itex#0`	TTCN document revert file
`filenJAAa002l89,s`	TTCN document working structure file
`filenJAAa002l89,t`	TTCN document working table file
`filename.itex-read_lock`	TTCN document read lock file
`filename.itex-lock`	TTCN document lock file
`filename.mp`	TTCN-MP file
`filename.log`	TTCN Suite log file
`filename.rpt`	TTCN Suite reporter file
`filename.c`	ANSI-C file generated by the TTCN to C compiler
`filename.h`	Header file generated by the TTCN to C compiler

The working *structure* and *table* files are the internal TTCN Suite files and will be created temporarily in the temporary directory. The TTCN Suite checkpoints automatically to these files during editing but the main data base files are updated only on save.

The revert file is created in the session directory (which is the same as the target directory unless the `ITEX_SESSION_DIR` environment variable is set). It has a name consisting of the name of the main data base file (including extension), a hash sigh (#) and a sequence number. The revert file is always stored in the TTCN-GR format even if it has a name like `filename.mp#0`. You can specify the compression of the revert files by using the resource `revertFileCompressionMethod`.

UNIX commands such as `mv`, `cp`, `rm` etc. can be used on these files (but use with caution!).

Locking of TTCN Documents

When a TTCN document is opened in the TTCN Suite Browser, a read-lock file for it (which has the additional suffix '-read_lock') is automatically created. The lock file is created in the same directory as the document. When the TTCN Suite Browser exits, the read-lock file is removed. If any another user (or the same user but using another TTCN Browser) try to open the same TTCN document, he will get a question informing that the file is opened by another user and asking to open it in read-only mode. The command-line TTCN2C compiler and other access applications always open TTCN documents in read-only mode. When the TTCN document is opened in read-only mode, no read-lock file is be created.

When a TTCN document is saved, a lock file for it (which has the additional suffix `-lock`) is automatically created. This means that no other user can access this file during the save operation. The lock files are created in the same directory as where the document is saved.

Transferring TTCN Suite TTCN Documents

The entire TTCN Suite data bases may be transferred between storage areas. This is useful in order to preserve information that is lost in the TTCN-MP format. Use the usual UNIX commands to achieve this.

Chapter

4

Tutorial: TTCN Suite Basics (in Windows)

This tutorial is intended as an easy introduction to the TTCN Suite for the newcomer. It is assumed that you have some basic knowledge about Windows. In addition, to find the TTCN Suite meaningful to use, you have to understand TTCN.

> **Note: Windows version**
>
> **This is the Windows version of the tutorial. The UNIX version can be found in chapter 5, *Tutorial: TTCN Suite Basics (on UNIX)*.**

> **Note:**
>
> **This document is a TTCN Suite primer and is not intended to provide a tutorial on TTCN. The volume Methodology Guidelines introduces some basic TTCN features presented with the aid of a simple example.**

Purpose of This Tutorial

The purpose of this tutorial is to make you familiar with the basic functions of the TTCN Suite. You will start by creating a test suite and you will learn various ways of editing tables.

It is assumed that you know how to use Windows but that you have no or little previous knowledge about the TTCN Suite. However, to understand the full use of the TTCN Suite, you must have knowledge of TTCN. Methodology Guidelines introduces some essential TTCN features, presented with the aid of a simple example.

Starting the TTCN Suite

It is assumed that the TTCN Suite has been installed correctly and that the installation directory is:
`C:\IBM\Rational\SDL_TTCN_Suite6.3`

To start TTCN Suite:

- Select the TTCN Suite icon from the *Start* menu.
  - You can also double-click the executable file `sdt.exe`, which should be located in:
    `C:\IBM\Rational\SDL_TTCN_Suite6.3\bin\wini386`.

After a few seconds, the *Organizer* is started. The Organizer is the main window from which you have access to the tools in the TTCN Suite environment.

A welcome window, where you may read the license agreement for SDL Suite and TTCN Suite, will also be displayed. The window disappears as soon as you click the *Continue* button or perform any action in the Organizer.

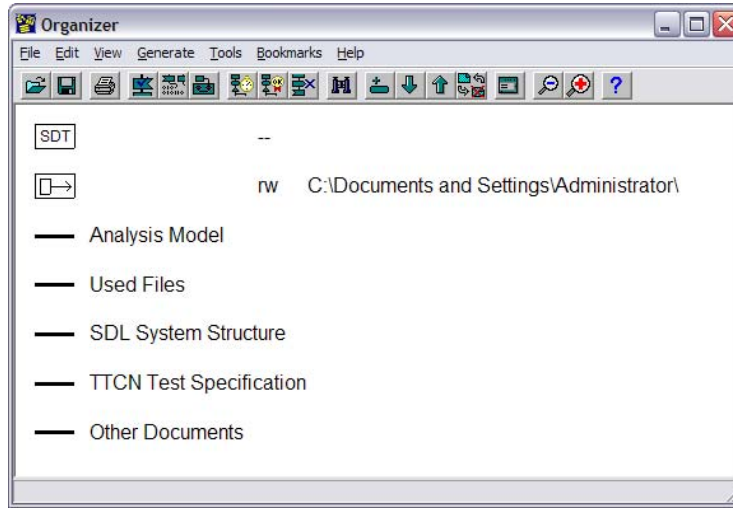


Figure 9: The Organizer main window

The Organizer consists of a main window and a log window. In the main window, five areas – known as *chapters* – are displayed by default:

- *Analysis Model*
- *Used Files*
- *SDL System Structure*
- *TTCN Test Specifications*
- *Other Documents*

You may freely use these chapters to hold a number of documents and chapters may also be renamed, deleted and created – the actual use is a matter of personal taste.

More information about customizing the chapters can be found in "Customizing the Organizer Chapters" on page 51 in chapter 3, *Tutorial: The Editors and the Analyzer, in the Getting Started*.

Creating a TTCN Test Suite Document

What You Will learn

- To set the source and target directory
- To create a TTCN test suite

Setting the Source and Target Directory

You will begin by setting the source and target directory in the Organizer. The source directory is where your new documents will be saved by default. The target directory is where generated files are put. Both of these directories must already exist – you cannot create them in the Organizer.

1. Double-click the source directory icon in the Organizer window.

   – You may also select the source directory icon and then select *Edit* from the *Edit* menu in the Organizer or you may select *Set Directories* from the *File* menu.

   The *Set Directories* dialog is opened, and you may change the source and target directories.

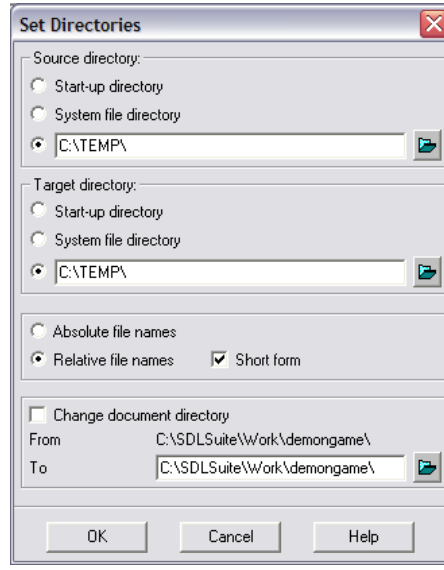


Figure 10: The Set Directories dialog

2. Select the source directory by writing the path in the text field or by browsing in the dialog that is opened when you click the folder button. Make sure that you have write access to the directory.
3. In the same way, you can change the target directory.
4. Click *OK* in the *Set Directories* dialog.

Creating a New Test Suite

When you have set the source and target directories, you should create a new TTCN document.

1. Select the chapter *TTCN Test Specification* in the Organizer.
2. Select *Add New* from the *Edit* menu in the Organizer.

   The *Add New* dialog is opened. In the dialog you should set the document type and name.

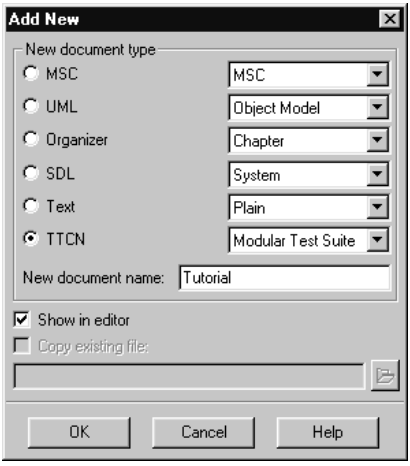


Figure 11: Add New dialog

3. Select *TTCN* and *Test Suite* or *Modular Test Suite* in the *New document type* list.
4. Type `Tutorial` in the *New document name* field.
5. Make sure that the option *Show in editor* is selected.
6. Click *OK*.

   A new TTCN document is created in the source directory. At the same time, a TTCN icon is created in the Organizer. After that, the TTCN Suite is started. The TTCN Browser will show the new test suite in collapsed format.

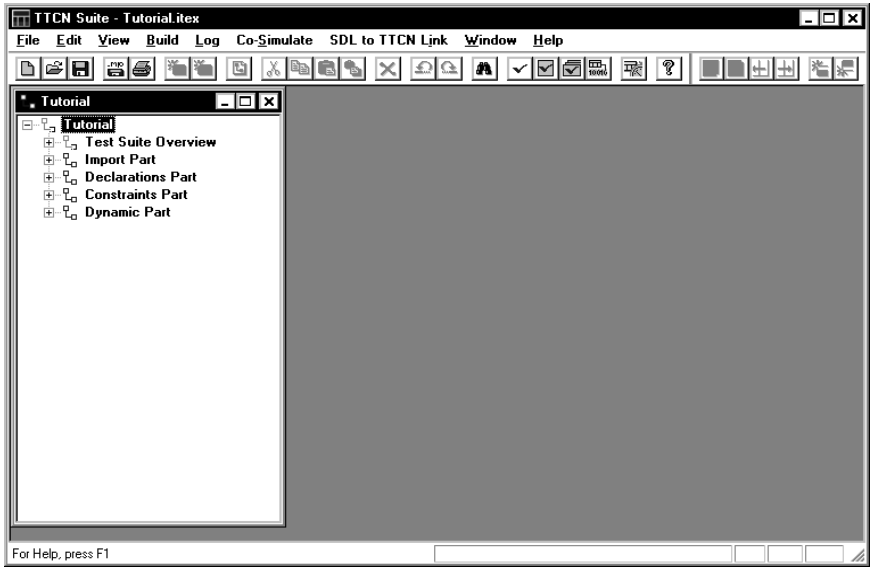


Figure 12: The TTCN Suite is started and the Browser displays the test suite

Using the Browser

What You Will Learn

- To expand and collapse nodes
- To build a test suite

Expanding and Collapsing Nodes

The Browser displays the test suite in a collapsed format, which is indicated by plus signs in front of the nodes. You expand and collapse the nodes the same way that you expand and collapse directory levels in the Windows Explorer.

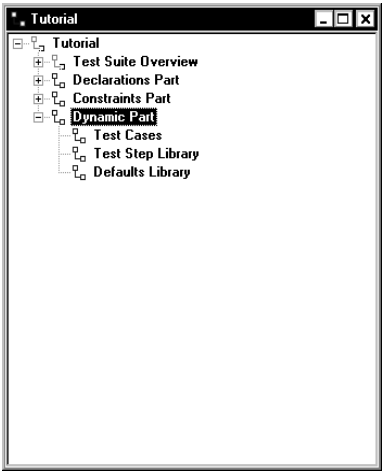


Figure 13: The Dynamic Part has been expanded

Some nodes are not marked with plus or minus signs. Those node are tree nodes without children. The node Test Cases in Figure 13 is an example of this. Such nodes represent the static (structural) parts of the test suite that cannot be opened or edited.

Building a Test Suite

You are now going to add a TTCN table to your test suite.

To add a PCO type:

1. Expand the Declarations Part.

   To be able to see the entire test suite, you may also have to enlarge the Browser window.

2. Select the PCO Type Declarations node.

3. Select *Add in* from the *Edit* menu.

   A PCO Type with the temporary name *NoName* is added in the *PCO Type Declarations* list. The table icon looks like a small graphical table, which means that it can be opened and edited.

4. Rename the PCO type *NoName* by clicking on it when it is selected, that is, it becomes highlighted. Name the PCO type **`LOWER_PCO`**.

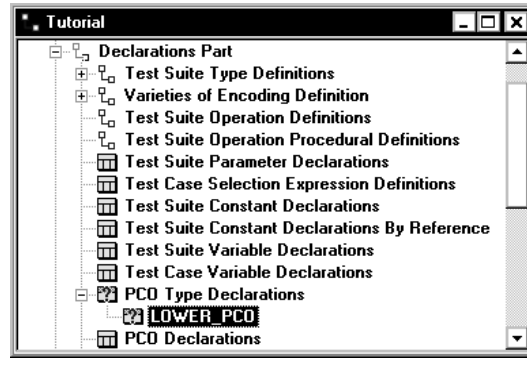


Figure 14: The test suite when a PCO Type has been added

Note that the new table is marked with a question mark to indicate that it has not yet been analyzed.

5. You should also add the following items:

   – A TTCN PCO called **`L`**
   (*Declarations Part > PCO Declarations*)

   – A TTCN PDU called **`SEND`**
   (*Declarations Part > PDU Type Definitions > TTCN PDU Type Definitions*)

   – A TTCN PDU called **`RECEIVE`**
   (*Declarations Part > PDU Type Definitions > TTCN PDU Type Definitions*)

   You can also copy and paste SEND and change the name to RECEIVE.

   – A TTCN PDU constraint on the SEND PDU called **`S1`**
   (*Constraints Part > PDU Constraint Declarations > TTCN PDU Constraint Declarations*)

   – A TTCN PDU constraint on the RECEIVE PDU called **`R1`**
   (*Constraints Part > PDU Constraint Declarations > TTCN PDU Constraint Declarations*)

   You can also copy and paste S1 and change the name to R1.

   – A test case called **`TEST_CASE_1`**
   (*Dynamic Part > Test Cases*)

   – A test step called **`TEST_STEP_1`**
   Dynamic Part > Test Step Library)

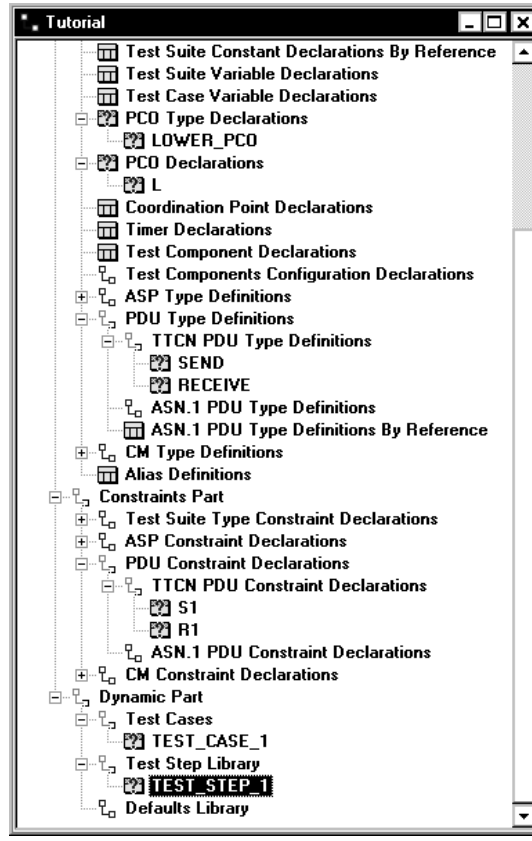


Figure 15: The test suite when the tables have been added

The next step is to edit the tables that you just added.

Using the Table Editor

What You Will Learn

- To open the Table Editor
- To resize the window parts
- To edit the contents of a table

Opening the Table Editor

The Table Editor is opened when you double-click a table icon in the Browser:

- Double-click the test case table *TEST_CASE_1*.

  The test case table is opened in the Table Editor.

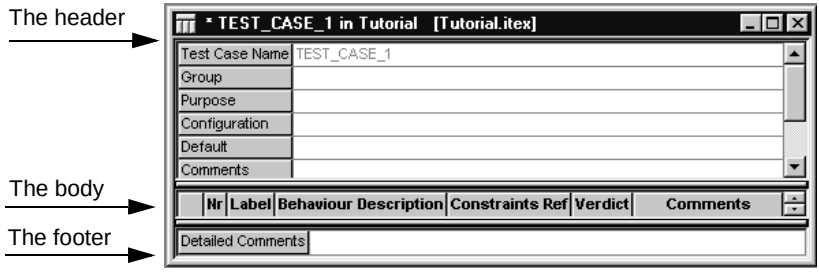


Figure 16: The test case TEST_CASE_1 which has not yet been edited

Resizing the Table

As you can see, the Table Editor window is divided into three parts: the header, the body and the footer. These parts can be resized and hidden.

- Drag the horizontal bars that separates the table parts to change the relative size.
- Drag a row separator in the left most column to change the height of a row.
- Drag a column separator in a header of the body part of the table to change the column width.

Editing the Test Case Table

As you can see, TEST_CASE_1, or any other table, consists of fields where text may be inserted and edited. This is what you are going to do now:

1. Click in the Purpose field and type some text, for example: **`This is an example test case for the TTCN Suite tutorial`**.
2. Type some text in the Description field: **`Example test case`**.
   - Instead of clicking the mouse to set the input focus, you can press `<down arrow>` until the cursor has reached the Description field.

   It is also possible to type text in the other fields but you do not have to do that in this tutorial. Note that the Group field is not editable. The contents of this field is always kept updated from the Browser structure.
3. Select *Insert New Row After* from the *Edit* menu.
   - You can also press `<Ins>` or `<Insert>`.

   A new empty line is added to the body of the table. Note that the line is automatically numbered.
4. Type **`L! SENT`** in the Behaviour description field of the new line.

> **Note:**
> The misspelling of “SEND” is intentional!

5. Type **`S1`** in the Constraints Ref field.
6. Press `<Ins>` or `<Insert>` to add another new line.
7. Type **`+TEST_STEP_1`** in the Behaviour Description field of row 2

   Note that the text is automatically indented.

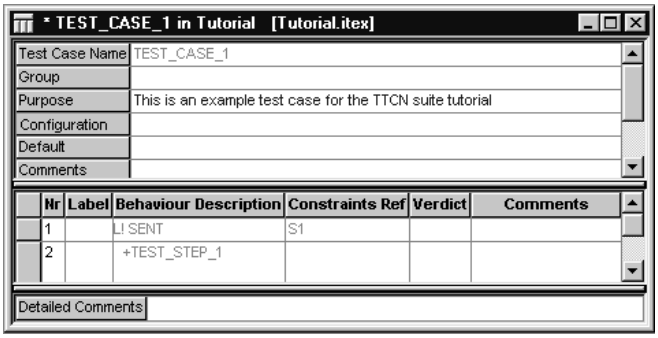


Figure 17: The test case TEST_CASE_1 when it has been edited.

The misspelling of "SEND" is intentional.

When you have edited TEST_CASE_1, you should close the table.

Completing the Test Suite

You should now edit the other tables that you have already added to the test suite. Use the tables in the following figures as models.

1. Edit the tables in the *Declarations Part* of the test suite, that is the PCO type *LOWER_PCO*, the PCO called *L* and the PDUs called *SEND* and *RECEIVE*:

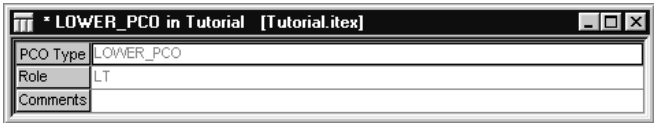


Figure 18: The PCO type LOWER_PCO

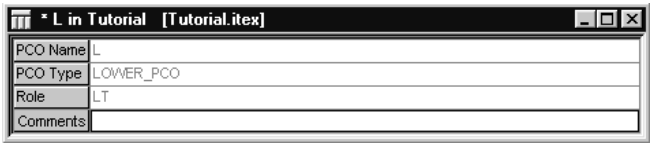


Figure 19: The PCO L

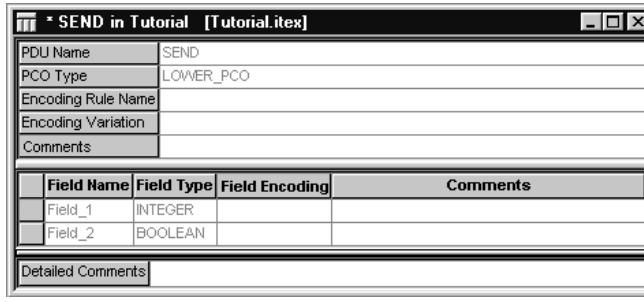


Figure 20: The PDU SEND

Since the contents of SEND and RECEIVE are identical, you can copy the text and rows from SEND and paste them in RECEIVE:

– Select text in the usual way with the mouse.

– Select an entire row in the **body** of a table by clicking the left most field.

When you are going to paste, note the following:

– Text can only be pasted in text edit mode, that is, when a field contains a text pointer.

– A body row can only be pasted when a body row is selected or when a body field is highlighted.

Note:

This means that there are two paste buffers: one for plain text and one for body rows.

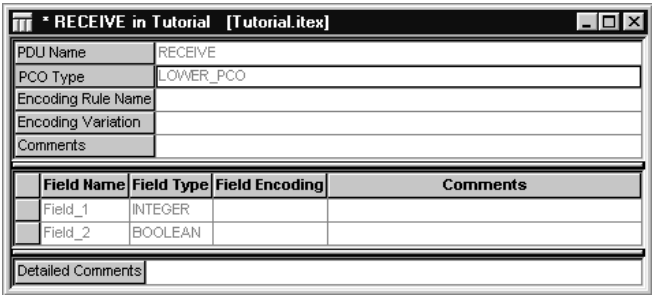


Figure 21: The PDU RECEIVE

2. Edit the tables in the Constraints Part, that is the PDU constraints called S1 and R1:

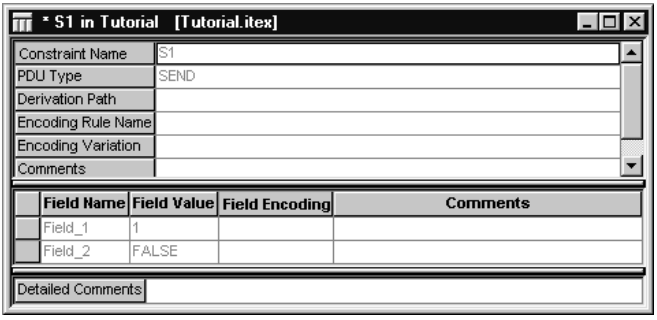


Figure 22: The PDU constraint S1

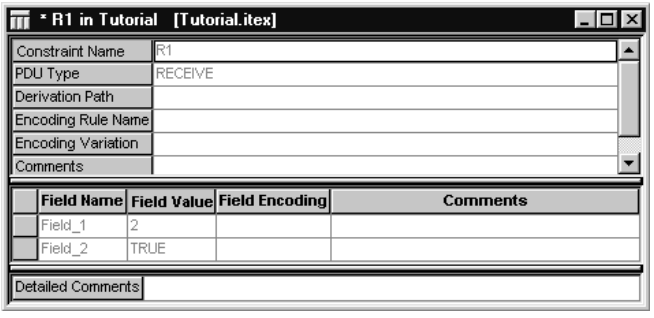


Figure 23: The PDU constraint R1

3. Edit the test step called *TEST_STEP_1* in the *Dynamic Part*:

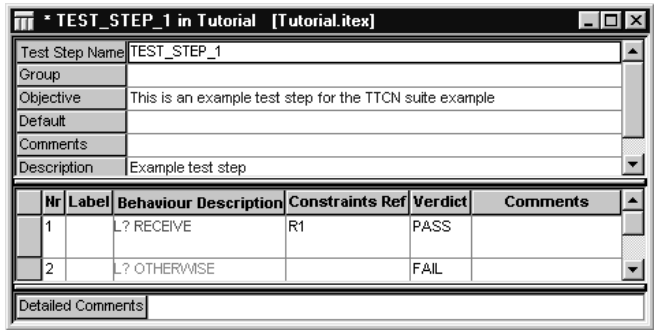


Figure 24: The test step TEST_STEP_1

When you add row 2 it is automatically indented, however, in this case it should not be. To decrease the indention, put the cursor in row 2 and then either:

- Press `<Alt+->` (`<Alt>` and the minus key).
- Click the minus indention quick-button.
- Select *Edit > Decrease Indention Level*.

4. Close all tables when you have finished editing.

5. Save the test suite by selecting *Save* in the *File* menu.

Analyzing the Test Suite

What You Will Learn

- To analyze a test suite
- To find errors

Analyzing the Test Suite

You are now going to analyze the test suite:

1. Select *Analyze Suite* from the *Build* menu.

   – You can also select the top node in the Browser and then select *Analyze*. *Analyze* only works on selected parts of a test suite (including sub-trees) so in this case it will have the same effect as *Analyze Suite*.)

   The *Analyzer/TTCN to C Compiler Settings* dialog is opened. You do not have to change the settings in this tutorial.

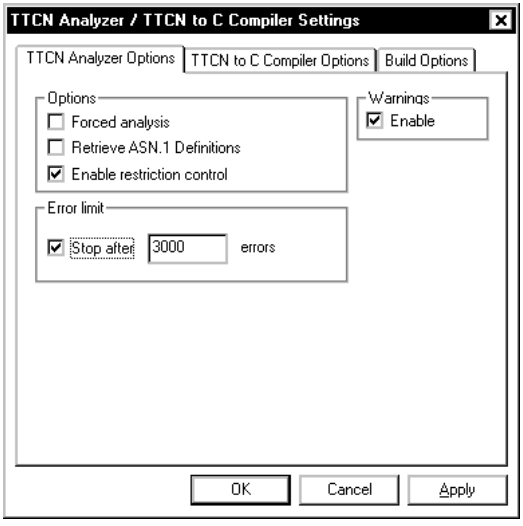


Figure 25: The Analyzer/TTCN to C Compiler Settings dialog

Note that if you analyze the test suite again by using a quick-button or a shortcut, the same options will be used but the dialog will not be displayed.

2. Click *OK*.

   If you have edited the test suite as described, the TTCN Suite Log Manager is opened with a textual log that should show a single error message:

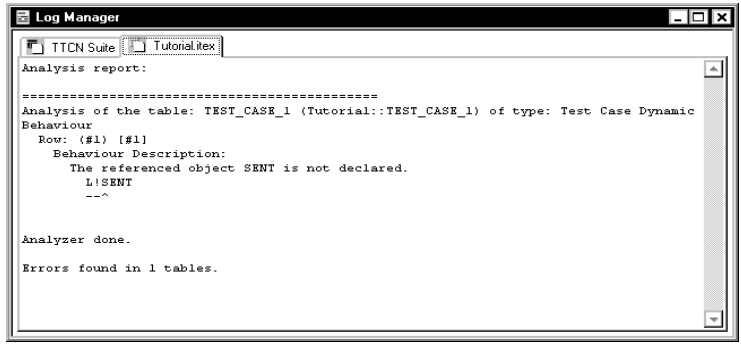


Figure 26: The TTCN Suite Log Manager window showing one error

The log text means that there is an error in line 1 of the behaviour description of the test case dynamic behaviour table called *TEST_CASE_1*.

Also note that the parent nodes to the erroneous table are marked with a red arrow and the table is marked with a red cross in the Browser.

Finding and Correcting the Error

To find the erroneous table, you use the popup menu:

1. In the Log Manager, click the table identifier, that is, the name of the table: TEST_CASE_1.
2. `<Ctrl>`-right-click the name of the table again.

   A popup menu is opened.

3. Select the top command in the popup menu: *Test Case Name: TEST_CASE_1*.

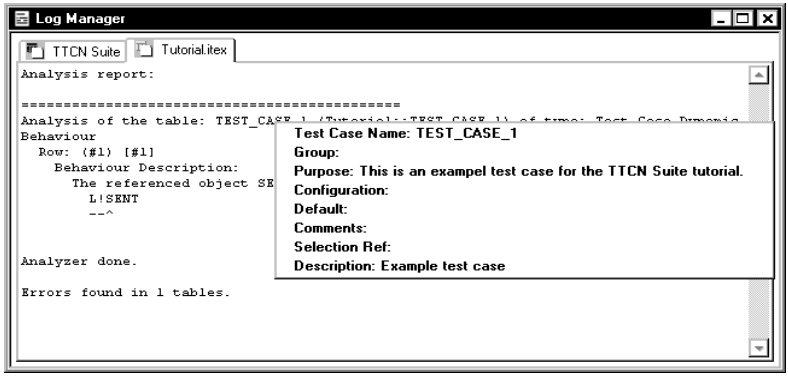


Figure 27: Finding a table by using the popup menu in the Log Manager

The Table Editor window is opened, displaying the *TEST_CASE_1* table. Note that the error is marked in red, in this case *L! SENT* in the *Behaviour Description* column.

You should now correct the error and analyze again:

1. Correct the error, i.e. change *SENT* to *SEND*.
2. Analyze the test suite again. You can do this while having the Table Editor window still opened since the Analyzer works on any level of the test suite.

   You will now have a small test suite that is syntactically correct. This means that the red markings will disappear.

Creating TTCN Tables in Other Ways

The methods described below make it faster and more efficient to create TTCN tables. They also avoid annoying spelling errors and reduces the amount of text that needs to be typed.

What You Will Learn

- To create a table by copying and pasting.
- To create a table by using the Data Dictionary.
- To delete a table.

Copying and Pasting

You are now going to practice another way of creating a TTCN table, namely by copying and pasting it.

1. Copy the PDU called SEND in the Browser.
2. Select S1 and then paste the table you have copied.

   The SEND table is pasted before S1.

Editing the New Table

1. Open the new SEND table.

   The Table Editor window is opened displaying the table. As you can see, the *PDU Type* field and the *Field* names are filled in. Now you just have to change the name and insert values in *Field_1* and *Field_2*.

2. Type a new name in the *Constraint Name* field: `S2`.
3. Give *Field_1* the value `3` and *Field_2* the value `TRUE` in the *Field Value* column.
4. Analyze the entire test suite as described in "Analyzing the Test Suite" on page 54.

Using the Data Dictionary

You can use the Data Dictionary to interactively build behaviour lines in behaviour tables. In the Data Dictionary dialog, you can compose sends, receives, timeouts and attachments, and at the same time see how it turns out in the dialog.

In this simple example, you will use the Data Dictionary to create a new test case with the same contents as TEST_CASE_1, and as a comparison you may have that table opened too:

1. Create a new test case and open it.
2. Name the test case **`TEST_CASE_2`**.
3. Open the *Data Dictionary* from the *View* menu.

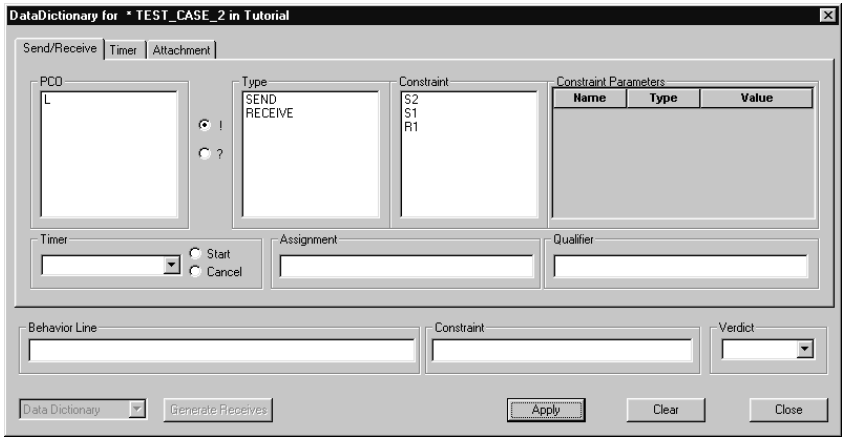


Figure 28: The Data Dictionary dialog

4. In the *Send/Receive* tab in the dialog, make the following settings:
   - Select the PCO *L*.
   - Select the *!* (exclamation mark) radio button, which means that you will add a send statement.
   - Select the type *SEND*.
   - Select the constraint *S1*.
   - No verdict should be selected.

   The *Behaviour Line*, the *Constraint* and the *Verdict* fields show the contents of the behaviour line that is inserted in the table.
5. Click *Apply*.

   The behaviour line you have composed is inserted in the table.
6. In the *Attachment tab*, select *TEST_STEP_1*.

   The *Behavior Line* will read `+TEST_STEP_1`.
7. Click the *Apply* button.

   The behaviour line is inserted and TEST_CASE_2 should now be identical to TEST_CASE_1.
8. Close the *Data Dictionary* dialog.

Deleting the Tables

You will not need the newly created tables in the rest of this tutorial. You should therefore delete them:

1. Select S2 and TEST_CASE_2 in the Browser.
2. Select *Delete* from the *Edit* menu or press the `<Delete> key`.

   The tables are deleted.

Exporting a Test Suite

What You Will Learn

- To save the test suite as TTCN-MP
- To generate the test suite overview
- To save the test suite as HTML

Saving a Test Suite to MP Format and Generating the Overview

You should now convert the test suite to TTCN-MP format. However, before the actual conversion, the test suite overview has to be generated. How to do this will also be described below.

1. Select the top node of the test suite in the Browser.
2. Select *Save As* from the *File* menu.

   Since the test suite overview has not been generated before, it has to be done now. A dialog is opened where you can confirm this operation.
3. Click *Yes* in the confirmation dialog.

   Since the test suite you have created is small, the generation will not take long. Also, once it has been generated, it will be updated automatically.

   A file dialog is opened.
4. Specify where the file is to be saved, ensure that you save as type TTCN-MP and give the file a name (**`example`**).

   The test suite will now be renamed to `example.mp`.

Hint:

Optionally, you can use the *Export to MP* button for exporting the `.itex` file to MP, while keeping the `.itex` file open.

A TTCN-MP file can be opened like an ordinary `.itex` file.

Saving a Test Suite as HTML

To save the entire test suite, or selected parts of it, as HTML, you proceed as described in "Saving a Test Suite to MP Format and Generating the Overview" on page 60. But instead of TTCN-MP, select HTML as the file type.

It is also possible to only export an opened table. To do this:

1. Open a table.
2. Select *Generate HTML* from the pop-up menu in the table.
3. Select a name of the file and where it should be saved.

An HTML document cannot be opened in the TTCN Suite again, but you can view it in an HTML browser.

Printing the Test Suite

You should by now have a test suite that is syntactically correct.

If you have saved the test suite as MP, as described earlier, the test suite overview should also have been generated and it does not have to be done again. The overview will be automatically updated as soon as you make any changes in the test suite after the generation.

If you did not save the test suite as MP, you will have to confirm the updating of the overview before printing is possible.

You can print a test suite both from the TTCN Suite and the Organizer. The main differences are that in the TTCN Suite there is a print preview, and in the Organizer, you can print several test suites at the same time.

Printing from the TTCN Suite

To print from the TTCN Suite:

1. Ensure that the Browser window is active.
2. Select *Print* from the *File* menu.

   The *Print* dialog is opened.
3. If necessary, change some settings.
4. Click *OK*.

   The test suite is printed.

Printing from the Organizer

To print from the Organizer:

1. Select the TTCN document icon in the Organizer.
2. Select *Print* from the *File* menu.

   The *Print TTCN* dialog is opened.
3. If necessary, change some settings.
4. Click *Print*.

   The test suite is printed.

So Far...

You should now have learned how to create a test suite and how to edit it in the Browser. You have also learned how to create and edit tables and how to work with the Table Editor. Other things you should have learned is how to analyze the test suite, how to find errors and how to save as MP and HTML.

By practicing this tutorial you have probably not achieved any knowledge about TTCN. If you want to know more about the basics of TTCN you may read the Methodology Guidelines. That volume is, however, not a tutorial on TTCN.

The following tutorial is chapter 6, *Tutorial: SDL and TTCN Integrated Simulator (W)*. Before you start practising that, you should know how to use the SDL Simulator. A tutorial on that can be found in *chapter 4, Tutorial: The SDL Simulator, in the Getting Started*.

Chapter 4 Tutorial: TTCN Suite Basics (in Windows)

Chapter

5

Tutorial: TTCN Suite Basics (on UNIX)

This tutorial is intended as an easy introduction to the TTCN Suite for the newcomer. It is also assumed that you have some basic knowledge about UNIX. In addition, to find the TTCN Suite meaningful to use, you have to understand TTCN.

Note: UNIX version

This is the UNIX version of the tutorial. The Windows version can be found in chapter 4, *Tutorial: TTCN Suite Basics (in Windows)*.

Note:

This document is a TTCN Suite primer and is not intended to provide a tutorial on TTCN. The volume Methodology Guidelines introduces some basic TTCN features presented with the aid of a simple example.

Purpose of This Tutorial

The purpose of this tutorial is to make you familiar with the basic functions of the TTCN Suite. The sections in this chapter are to be read sequentially. You will start by setting up the TTCN Suite environment. Then you will create a TTCN test suite, edit the tables in the test suite and apply some basic the TTCN Suite tools.

It is assumed that you know how to use UNIX but that you have no or little previous knowledge about the TTCN Suite. However, to understand the full use of the TTCN Suite, you should have knowledge of TTCN. The volume Methodology Guidelines introduces some essential TTCN features, presented with the aid of a simple example.

Setting Up the TTCN Suite Environment

It is assumed that the TTCN Suite has been installed correctly, according to the instructions in the *Installation Guide*. The installation directory is pointed out by the environment variable `$telelogic`. This variable has to be set correctly in your UNIX environment.

1. The TTCN Suite commands are made available if you add the `bin` directory to the environment variable `PATH`:

   ```
   setenv PATH ${PATH}:$telelogic/bin
   ```

2. Also set the environment variable `MANPATH` to include the TTCN Suite and the X11 man directories, to make the manual pages available:

   ```
   setenv MANPATH \
   ${MANPATH}:$telelogic/itex/man:$telelogic/X11/man
   ```

If you type **`man itex`**, the UNIX manual entry for the TTCN Suite will be displayed. It describes the switches that may be used to set the default settings of the dialog options.

To be able to use the on-line help, you need an HTML viewer, either Firefox, Netscape or Internet Explorer. To select the HTML viewer to use, see "Customizing the On-Line Help" on page 298 in chapter 4, *Managing Preferences, in the User´s Manual*.

Starting the TTCN Suite

Once the TTCN Suite software has been installed, you can start it from a UNIX shell tool (e.g. an X-term) by the command `itex`:

- Type **`itex`** to start the TTCN Suite.

After a few seconds, the *Organizer* window is displayed. The Organizer is the main window from which you have access to the tools in the TTCN Suite environment.

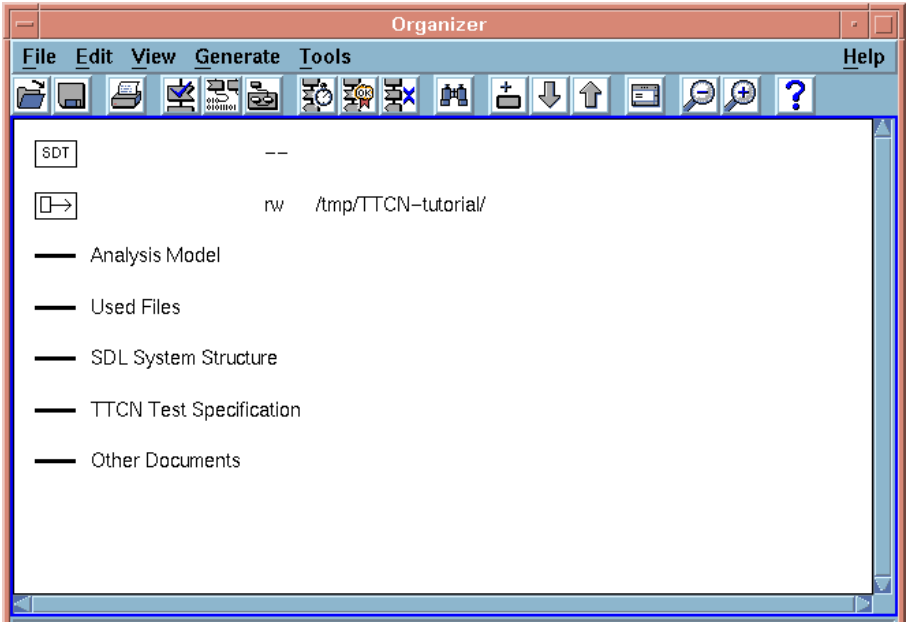


Figure 29: The Organizer main window

A welcome window, where you may read the licensing agreement for TTCN Suite, will also be displayed. The window disappears as soon as you click the *Continue* button or perform any action in the Organizer.

The Organizer consists of a main window and a log window. In the main window, five areas – known as *chapters* – are displayed by default:

- *Analysis Model*
- *Used Files*

- *SDL System Structure*
- *TTCN Test Specifications*
- *Other Documents*

You may freely use these chapters to hold a number of documents and chapters may also be renamed, deleted and created – the actual use is a matter of personal taste.

More information about customizing the chapters can be found in "Customizing the Organizer Chapters" on page 51 in chapter 3, *Tutorial: The Editors and the Analyzer, in the Getting Started*.

Creating a TTCN Test Suite Document

What You Will Learn

- To set the source and target directory
- To create a new test suite

Setting the Source and Target Directory

You will begin by setting the source and target directory in the Organizer. The source directory is where your new documents will be saved by default. The target directory is where generated files are put. Both of these directories must already exist – they cannot be created in the Organizer.

1. Double-click the source directory icon in the Organizer window.

   – You may also select the source directory icon and then select *Edit* from the *Edit* menu in the Organizer or select *Set Directories* from the *File* menu.

   The *Set Directories* dialog is opened, and you may change the source and target directories.

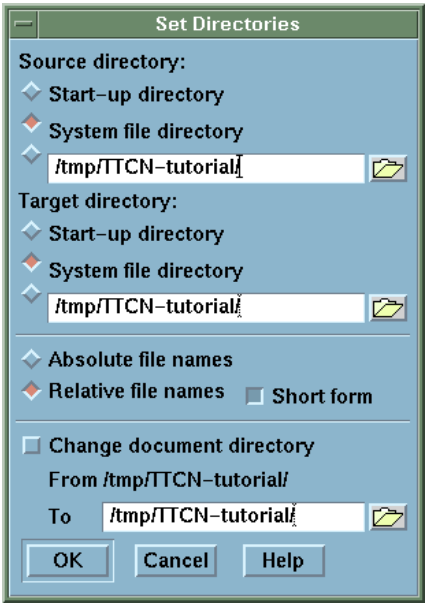


Figure 30: The Set Directories dialog

2. Select the source directory by writing the path in the text field or by browsing in the dialog that is opened when you click the folder button. Make sure that you have write access to the directory.
3. In the same way, you can change the target directory.
4. Click *OK* in the *Set Directories* dialog.

Creating a New Test Suite

When you have set the source and target directories, you should create a new TTCN document of type *test suite* or *modular test suite*:

1. Select the chapter *TTCN Test Specification* in the Organizer.
2. Select *Add New* from the *Edit* menu in the Organizer.

   The *Add New* dialog is opened. In the dialog you should set the document type and name.

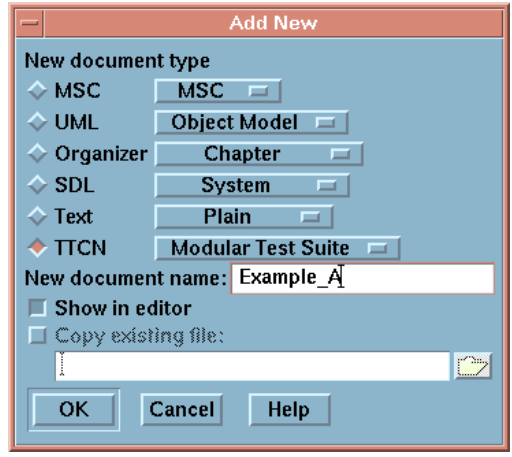


Figure 31: Add New dialog

3. Select *TTCN* and *Test Suite* or *Modular Test Suite* in the *New document type* list.
4. Type **`Example_A`** in the *New document name* field.
5. Make sure that the option *Show in editor* is selected.
6. Click the *OK* button.

   A new TTCN document of type test suite or modular test suite is created. At the same time, a TTCN icon is created in the Organizer. After that, the TTCN Browser window is opened. The Browser will show the new test suite in a collapsed format.

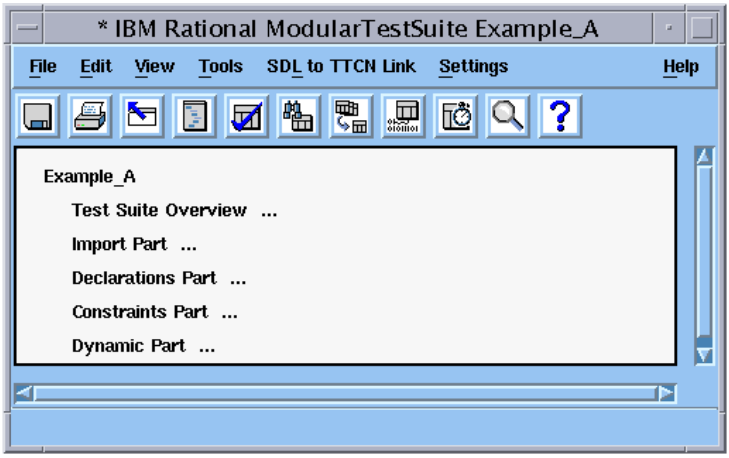


Figure 32: The Browser window with a collapsed test suite

Using the Browser

What You Will Learn

- To expand and collapse the test suite
- To add TTCN tables to the test suite

Expanding and Collapsing the Test Suite

The Browser displays the test suite in a collapsed format, which is indicated by the three dots after the items in the test suite.

To expand the entire test suite:

1. Select the top node in the Browser, that is *Example_A*.
2. Select *Expand Tree* from the *View* menu.
   - It is also possible to open the pop-up menu on *Example_A* and select *Expand Tree* from it.

   All the collapsed items in the Browser are expanded.

Shortcut:

- `<Shift+e>` (*Expand Tree*)

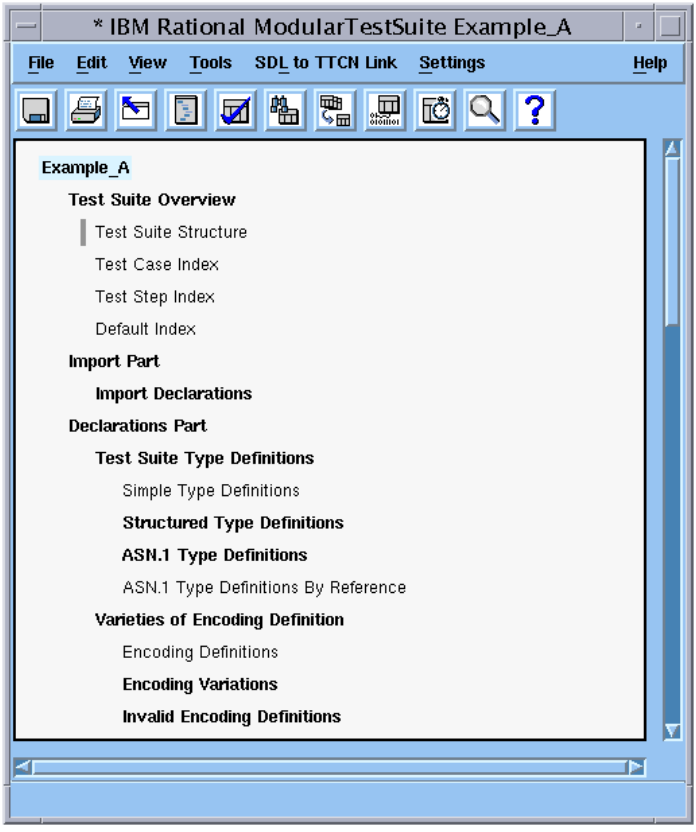


Figure 33: The Browser window with the expanded modular test suite and no selected items

You may also want to deselect the test suite:

3. Deselect the test suite by selecting *Deselect All* from the pop-up menu.

Shortcut:

- `<Ctrl+\>` (*Deselect All*)

To collapse the test suite:

- Open the pop-up menu on *Example_A* and select *Collapse* from it.

Shortcut:

- `c` (*Collapse*)
- `e` (*Expand* – does not have the same effect as *Expand Tree*)
- `<Shift+e>` (*Expand Tree*)

Note that *Expand* and *Collapse* only work on a **single** selection. If more than one item or if no item is selected, the *Collapse*, *Expand* and *Expand Tree* commands are dimmed. In the following, it is assumed that the test suite is completely expanded in the Browser.

You should also note that items in **bold** font represent the static (structural) parts of the test suite that cannot be opened or edited. The items in normal, plain font represent TTCN tables, which of course can be opened and edited.

Building a Test Suite

You are now going to add a simple TTCN table to your test suite.

To add a PCO Type to *Example_A* in the Browser:

1. Select *PCO Type Declarations* in the *Declarations Part*.
2. Select *Add* from the Browser *Edit* menu.

   Note that *Add* only works when a **single** item is selected.

Shortcut:

- `<Ctrl+Shift+Ins>` (*Add*)

A PCO type with the temporary name *NoName* is added in the *PCO Type Declarations* list.

3. Rename the PCO type *NoName* by opening the pop-up menu on *NoName* and selecting *Rename* from it.

Shortcut:

- First click on *NoName*, then press `r` (*Rename*).

It is now possible to type the new name. In this example we will call the PCO type **`LOWER_PCO`**.

4. In a similar manner as described above, you should also add the following items:

   – A TTCN PCO called **`L`**
   (*Declarations Part > PCO Declarations*)

   – A TTCN PDU called **`SEND`**
   (*Declarations Part > PDU Type Definitions > TTCN PDU Type Definitions*)

   – A TTCN PDU called **`RECEIVE`**
   (*Declarations Part > PDU Type Definitions > TTCN PDU Type Definitions*)

   – A TTCN PDU constraint on the SEND PDU called **`S1`**
   (*Constraints Part > PDU Constraint Declarations > TTCN PDU Constraint Declarations*)

   – A TTCN PDU constraint on the RECEIVE PDU called **`R1`**
   (*Constraints Part > PDU Constraint Declarations > TTCN PDU Constraint Declarations*)

   – A test case called **`TEST_CASE_1`**
   (*Dynamic Part > Test Cases*)

   – A test step called **`TEST_STEP_1`**
   Dynamic Part > Test Step Library)

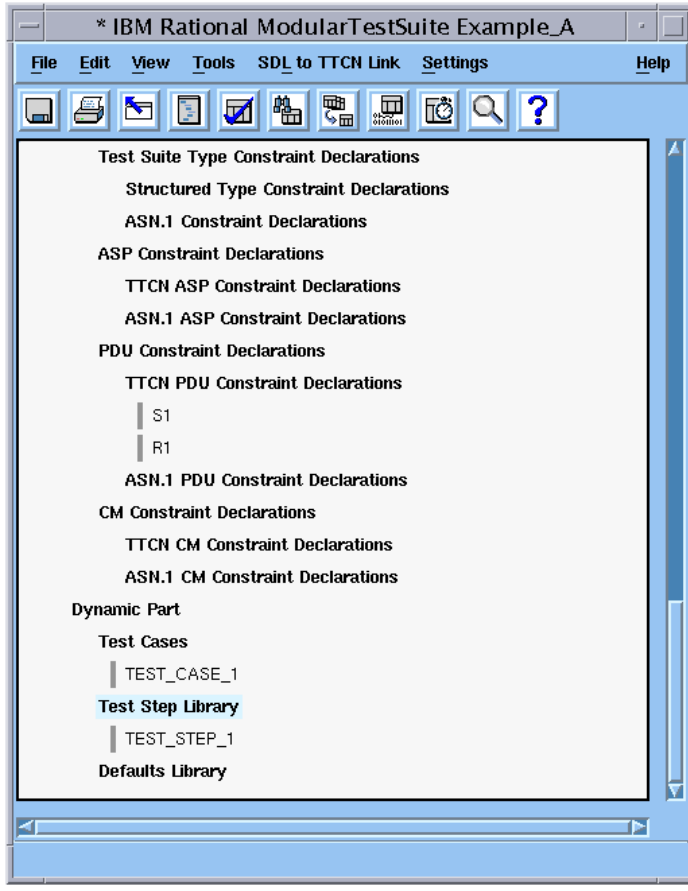


Figure 34: The Browser window showing some of the newly added items

The next step is to edit the tables that you just added.

Using the Table Editor

What You Will Learn

- To open the Table Editor
- To edit the contents of a table

Opening the Table Editor

The Table Editor window is opened when you double-click on a table in the Browser window.

- Double-click the test case *TEST_CASE_1*.

  The test case table is opened in the Table Editor window.

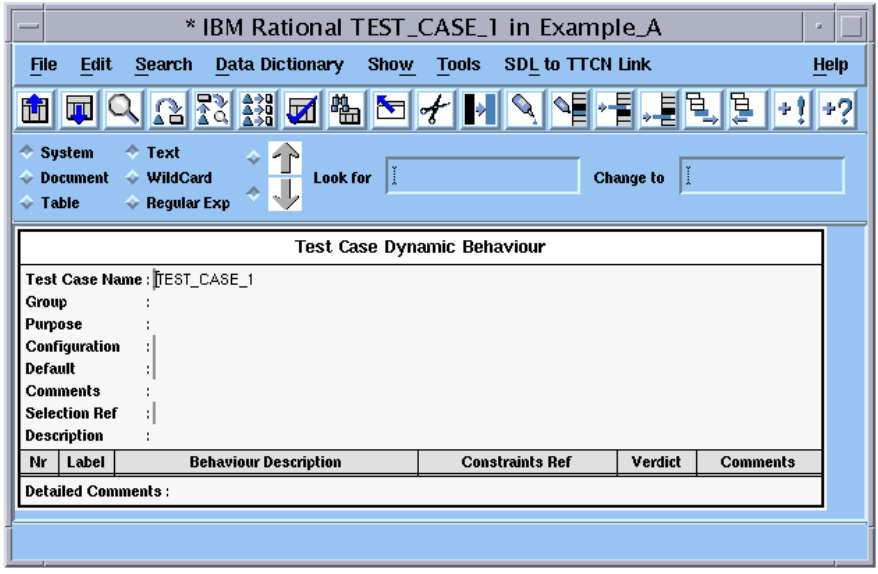


Figure 35: The test case TEST_CASE_1 which has not yet been edited

Editing the Test Case Table

As you can see, the TEST_CASE_1 table, or any other table, consists of fields where text may be inserted and edited. You are now going to edit the contents of TEST_CASE_1:

1. Click in the *Purpose* field and type some text, for example:
   `This is an example test case for the TTCN tutorial`.
2. Type some text in the *Description* field, e.g. **`Example test case`**.
   – Instead of clicking the mouse to set the input focus in the *Description* field, you may also press `<Ctrl+Down arrow>` until the cursor has reached the *Description* field.

   It is also possible to type text in the other fields but you do not have to do that in this tutorial. Note that the *Group* field is not editable. The contents of this field is always kept updated by the TTCN Suite from the Browser structure.
3. Press `<Ins>` or `<Insert>`.

   A new, automatically numbered, empty line is added to the body of the table.
4. Type **`L! SENT`** in the *Behaviour description* field of the new line.

Note:

The misspelling of *SEND* is intentional!

5. Type **`S1`** in the *Constraints Ref* field.
6. Press `<Ins>` or `<Insert>` to add another new line.
7. Type **`+TEST_STEP_1`** in the *Behaviour Description* field of row 2.

   Note that the text is automatically indented.

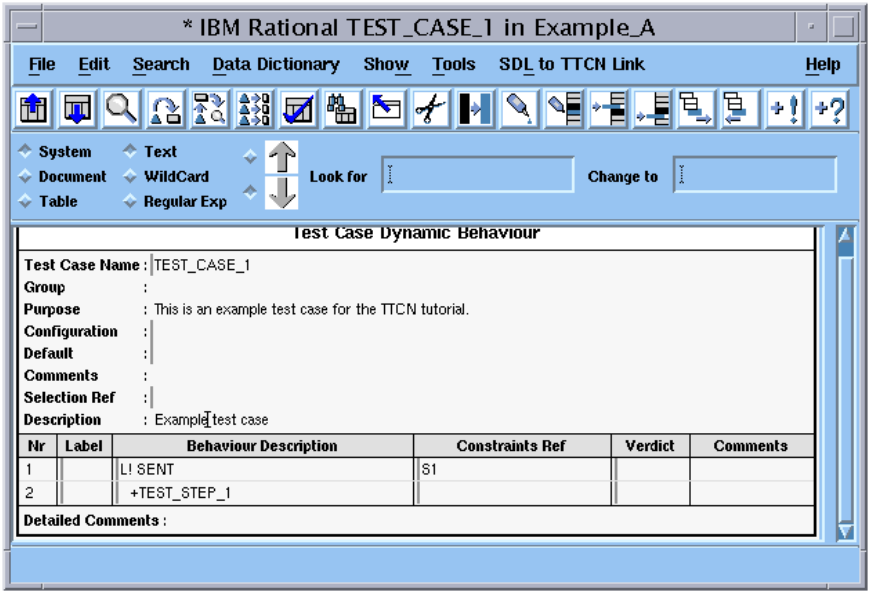


Figure 36: The test case TEST_CASE_1 when it has been edited. The misspelling of "SEND" is intentional.

Copying and Pasting Text and Table Rows

Before you close the editor you may want to try to copy and paste text and rows in the table:

- If you only want to copy text, you simply select and copy it, either by a menu choice or shortcut. To paste the text, you can either use a menu choice or shortcut.
- If you want to copy an entire row in the body of a table, you select it by pressing `<Ctrl>` while at the same clicking in the row. (A row in a dynamic behaviour table is selected if you click in the *Nr* column). When the row has been selected you can press `<Ctrl+c>` to copy it. You paste it with `<Ctrl+Meta+v>` (to paste after selected row) or `<Ctrl+Shift+v>` (to paste last in the body of the table).

Shortcut:

- `<Ctrl+x>` (*Cut* a row)
- `<Ctrl+c>` (*Copy* a row)
- `<Ctrl+v>` (*Paste* a row)
- `<Ctrl+Meta+v>` (*Paste After*)
- `<Ctrl+Shift+v>` (*Paste Last*)
- `<Ctrl+Right arrow>` (moves the input focus right)
- `<Ctrl+Left arrow>` (moves the input focus left)
- `<Ctrl+Down arrow>` (moves the input focus down)
- `<Ctrl+Up arrow>` (moves the input focus up)

Note:

There are two paste buffers: one for plain text and one for table rows.

When you have finished copying and pasting, make sure that the contents of the table are exactly as in Figure 36.

Closing the Table Editor

- When you have edited TEST_CASE_1, close the table from the *File* menu.

Completing the Test Suite

Now you should edit the other tables that you have already added to the test suite. Do this in a similar manner as described above, by using the tables in the following figures as models.

1. Edit the tables in the *Declarations Part* of the test suite, that is the PCO type *LOWER_PCO*, the PCO called *L* and the PDUs called *SEND* and *RECEIVE*:

PCO Type Declarations		
PCO Type	Role	Comments
LOWER_PCO	LT	
Detailed Comments :		

Figure 37: The PCO type LOWER_PCO

PCO Declarations			
PCO Name	PCO Type	Role	Comments
L	LOWER_PCO	LT	
Detailed Comments :			

Figure 38: The PCO L

PDU Type Definition			
PDU Name : SEND			
PCO Type : LOWER_PCO			
Encoding Rule Name :			
Encoding Variation :			
Comments :			
Field Name	**Field Type**	**Field Encoding**	**Comments**
Field_A	INTEGER		
Field_B	BOOLEAN		
Detailed Comments :			

Figure 39: The PDU SEND

PDU Type Definition			
PDU Name : RECEIVE			
PCO Type : LOWER_PCO			
Encoding Rule Name :			
Encoding Variation :			
Comments :			
Field Name	**Field Type**	**Field Encoding**	**Comments**
Field_A	INTEGER		
Field_B	BOOLEAN		
Detailed Comments :			

Figure 40: The PDU RECEIVE

2. Edit the tables in the *Constraints Part* of the test suite, that is the PDU constraints called *S1* and *R1*:

PDU Constraint Declaration

Constraint Name : S1
PDU Type : SEND
Derivation Path :
Encoding Rule Name :
Encoding Variation :
Comments :

Field Name	Field Value	Field Encoding	Comments
Field_A	1		
Field_B	FALSE		

Detailed Comments :

Figure 41: The PDU constraint S1

PDU Constraint Declaration

Constraint Name : R1
PDU Type : RECEIVE
Derivation Path :
Encoding Rule Name :
Encoding Variation :
Comments :

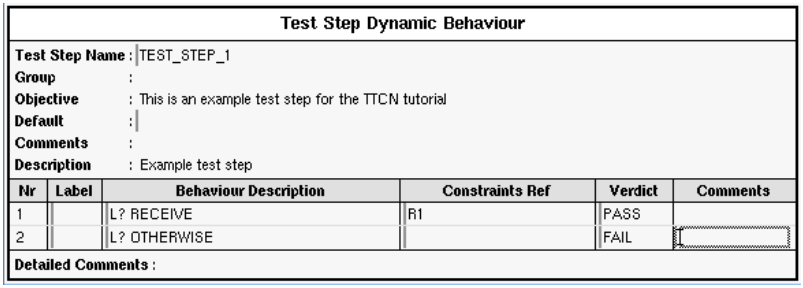
Field Name	Field Value	Field Encoding	Comments
Field_A	2		
Field_B	TRUE		

Detailed Comments :

Figure 42: The PDU constraint R1

3. Edit the test step called *TEST_STEP_1* in the *Dynamic Part* of the test suite:

   §When you add row 2 it is automatically indented, however, in this



Test Step Dynamic Behaviour

Test Step Name : TEST_STEP_1
Group :
Objective : This is an example test step for the TTCN tutorial
Default :
Comments :
Description : Example test step

Nr	Label	Behaviour Description	Constraints Ref	Verdict	Comments
1		L? RECEIVE	R1	PASS	
2		L? OTHERWISE		FAIL	

Detailed Comments :

Figure 43: The test step TEST_STEP_1

   case it should not be. To decrease the indention:

   – Select the entire line by clicking in the *Nr* column. Then select *Decrease Indent* from the *Edit* menu.

     (The shortcuts are <-> and <+> on the right-hand keypad.)

4. Close all Table Editors when you have finished editing.
5. Save the test suite by selecting *Save* in the *File* menu. A *Save As* dialog is displayed.
6. Click OK.

Analyzing the Test Suite

What You Will Learn

- To analyze a test suite
- To find errors

Analyzing the Test Suite

You are now going to analyze the test suite:

1. Select *Analyze* from the Browser *Tools* menu.

   The *Analyzer* dialog is opened. You do not have to change the settings in this tutorial.

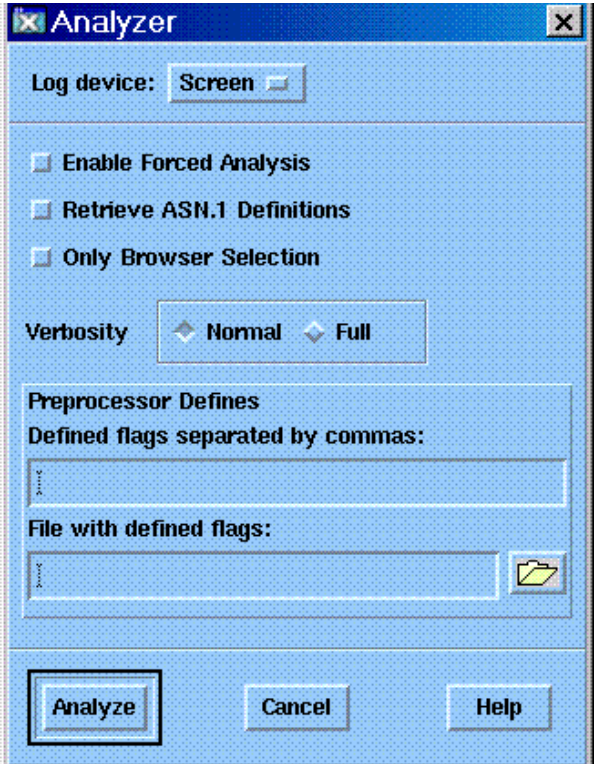


Figure 44: The Analyzer dialog

2. Click *Analyze*.

   The log window is opened. If you have edited the test suite as described above, the log should show a single error message:

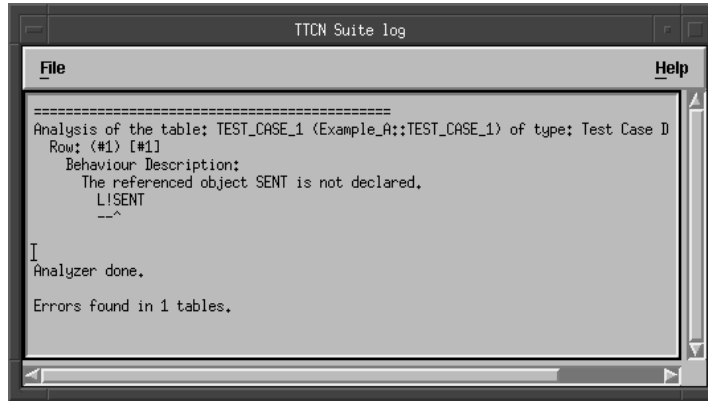


Figure 45: The TTCN Suite log window showing one error

The log text means that there is an error in line 1 of the *Behaviour Description* of the test case dynamic behaviour table called *TEST_CASE_1*.

Finding and Correcting the Error

To find the erroneous table:

1. Select the test case name in the log.

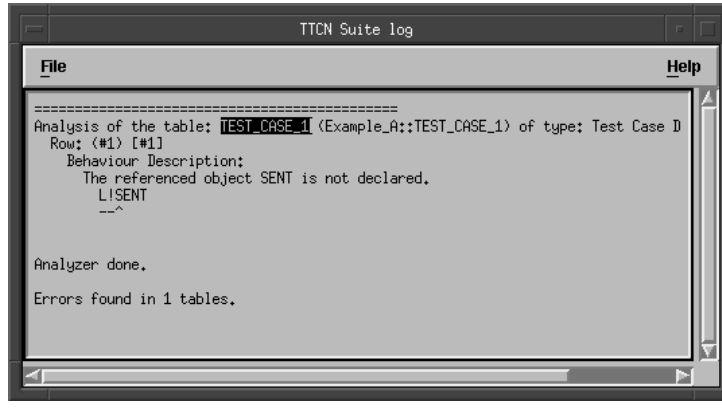


Figure 46: Selecting text in the TTCN Suite log

2. Select *Find Table* from the Browser *Tools* menu.

   The table will be found and displayed. In the first behaviour line you can see that you have written *SENT* instead of *SEND*.

   Note that the error line is marked in red. In the Browser, the erroneous table is also marked with a red bar.

3. Correct the error, i.e. change *SENT* to *SEND*.

4. Analyze the table by selecting *Analyze* from the Table Editor *Tools* menu.

   You will now have a small test suite that is syntactically correct. This means that the red markings will disappear.

   You may now close the table editor and log windows.

Saving a TTCN Test Suite Document

What You Will Learn

- To save a TTCN document
- To change its file name and/or its format

Save As Document

When you want to save a newly created document into a file, you must make a number of choices in the *Save As* operation:

1. Select `Save As` from the `File` menu in the Browser.
2. Select *Binary* (TTCN-GR) or *Text* (TTCN-MP).

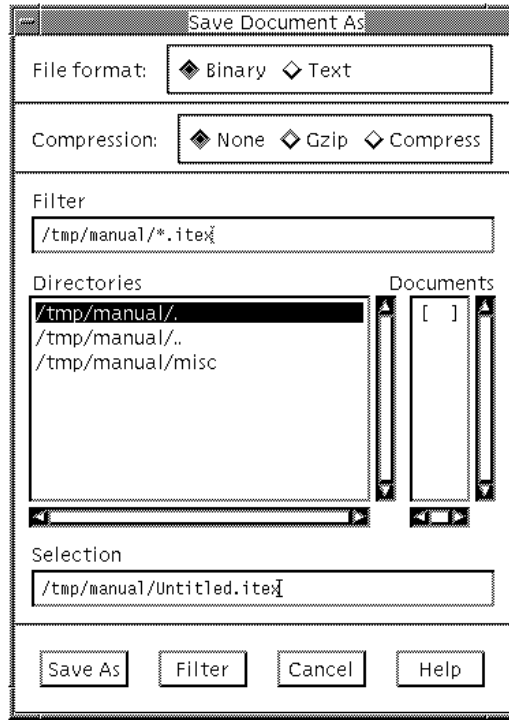


Figure 47

3. If you chose *Binary*, you need to select *None*, *Gzip* or *Compress*.[1]
4. Select a file name.
5. Press *Save As*.

   At the same time, the TTCN icon in the Organizer is associated with the specified file name.

Save Document

To save a document you can use either the *Save Document* item in the *File* menu of the Browser, or use the left most button in the toolbar of the Browser. The *Save Document* operation saves the document on disk with the same file name and format as before. If you try to save a document which has no file name assigned yet, the *Save As* operation will be initiated instead.

1. For maximum compatibility, the None selection should be used. The Gzip selection is preferred over Compress, both for portability and performance.

Creating TTCN Tables in Other Ways

The methods described below make it faster and more efficient to create TTCN tables. They also avoid annoying spelling errors and reduces the amount of text that needs to be typed.

What You Will Learn

- To create a table by copying and pasting
- To create a table by using the *Data Dictionary* menu
- To delete a table

Copying and Pasting

You shall now practice another way of creating a TTCN table, namely by copying and pasting it.

1. Find the PDU called *SEND* in the Browser.
2. Copy the *SEND* PDU by using the popup menu.
3. Find the PDU Constraint called *S1*.
4. Open the pop-up menu on *S1* and select *Paste Before* from it.

   The *SEND* copy is pasted before *S1*.

Editing the New Table

1. Open the new *SEND* table.

   The Table Editor window is opened displaying the table. As you can see, the *PDU Type* field and the *Field* names are filled in. Now you just have to change the name and insert values in *Field_1* and *Field_2*.

2. Type a new name in the *Constraint Name* field, e.g. `S2`.
3. Give *Field_A* the value `3` and *Field_B* the value `TRUE` in the *Field Value* column.
4. Analyze the entire test suite as described in "Analyzing the Test Suite" on page 85.

Using the Data Dictionary

This section contains an example of an efficient way of creating test cases. Start by creating a new test case:

1. Create a new test case.
2. Name the test case, for example **`TEST_CASE_2`**.
3. Open the test case.
4. Select *Add Send Statement* from the Table Editor *Data Dictionary* menu. The *Add Send* dialog is opened.

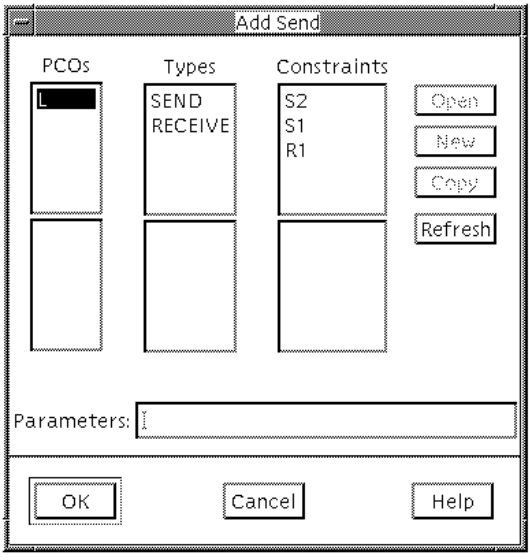


Figure 48: Add Send dialog

5. Click on *S2* in the *Constraints* list.
6. Click *OK*.

   A send statement has now been automatically generated.
7. Select the generated line in the Table Editor by clicking in the *Nr* column.

8. Select *Add Attach Statement* from the Table Editor *Data Dictionary* menu. The *Add Attach* dialog is opened.

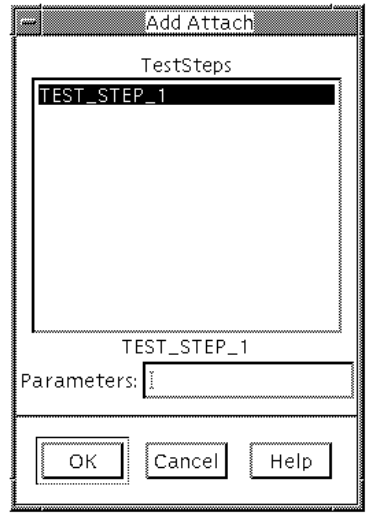


Figure 49: Add Attach dialog

There is only one test step available (i.e. *TEST_STEP_1*) so this will be automatically selected.

9. Click *OK*.

A new line has been added to the table.

Deleting the Tables

For the remainder of the tutorial you will not need the two tables that you have just created. You should therefore delete them.

1. Select *Deselect All* from the pop-up menu in the Browser.
2. Select *TEST_CASE_2*.
3. Select the PDU Constraint *S2*, while at the same time pressing `<Ctrl>`.
4. Select *Delete* from the *Edit* menu. The items are deleted.

Selecting Browser Items

The Selector is a tool that you may use when you want to make selections of Browser items. Before you apply the Selector, the Browser must already contain a selection.Then you can specify, with the Selector, if you want to increase, decrease or replace the existing selection.

Based on the selections in the Browser, you can make sub Browsers. A sub Browser works as the normal TTCN Browser, but it does not contain the entire test suite – only selected parts.

What You Will Learn

- To use the Selector tool
- To create a sub Browser

Selection 1

Suppose that you wish to select all the tables in the test suite that reference the PDU called *SEND*:

1. Deselect the entire test suite.
2. Select the single PDU called *SEND*.
3. Select *Selector* from the Browser *Tools* menu.

   The *Selector* dialog is opened:

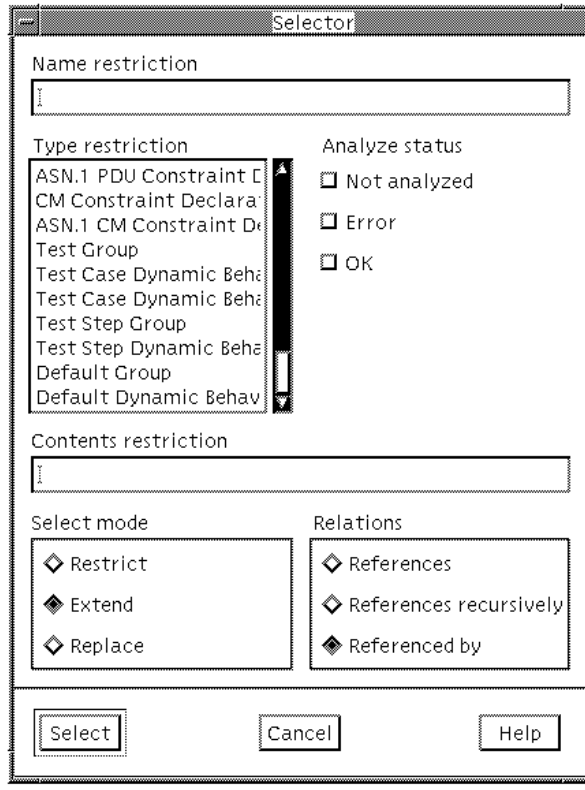


Figure 50: The Selector dialog

4. Select the *Extend* option in *Select Mode*.
5. Choose the *Referenced by* option in the *Relations* box.

   You do not have to change any other options.
6. Click the *Select* button.

   Now take a look at the Browser. You should see that not only is the *SEND* PDU selected but also the constraint *S1* and the test case *TEST_CASE_1*. Only these objects reference the *SEND* PDU. (You may have to scroll the Browser to see them all.)

Selection 2

As another example, suppose that you wish to find all behaviour tables that have the character *1* included in their identifier and have a *PASS* verdict in the table:

1. Select the entire test suite.
2. Select *Selector* from the Browser *Tools* menu.

   The *Selector* dialog is opened.
3. Type `1` in the *Name restriction* field.

   This means that the names of the tables that you want to select should include the character *1*.
4. Select *Test Case Dynamic Behaviour*, *Test Step Dynamic Behaviour* and *Default Dynamic Behaviour* in the *Type Restriction* box.
5. Type `PASS` in the *Content restriction* field.

   This means that there must be at least one occurrence of the word *PASS* in the selected tables.
6. Select *Restric*t in the *Select mode* box.

   In other words, the intention is to select all behaviour tables that have a *1* in their names and that have a *PASS* substring in the contents of the table.

   The *Analyze status* and *Relations* options should not be set.
7. Click the *Select* button.

   Only *TEST_STEP_1* is selected in the Browser window.

Selection 3

Compare the result of the selection above with the following:

1. Select the entire test suite.
2. Apply the Selector with **almost** the same options as described above.
   - The difference is that you should remove *PASS* from the *Content restriction* field, i.e. the field should be empty.

3. Click the *Selector* button.

   This time, both *TEST_CASE_1* and *TEST_STEP_1* are selected, because a *PASS* substring does not appear in the test case.

Creating a Sub Browser

It might be suitable at this point to create a sub Browser of the selection that you have just created (i.e. just the items *TEST_CASE_1* and *TEST_STEP_1*):

- Select *Browser* from the Browser *Tools* menu.

  A sub Browser containing the selected objects is created, together with the necessary minimal static structure to glue them together.

  You will not need this sub Browser in this tutorial so you may close it if you wish.

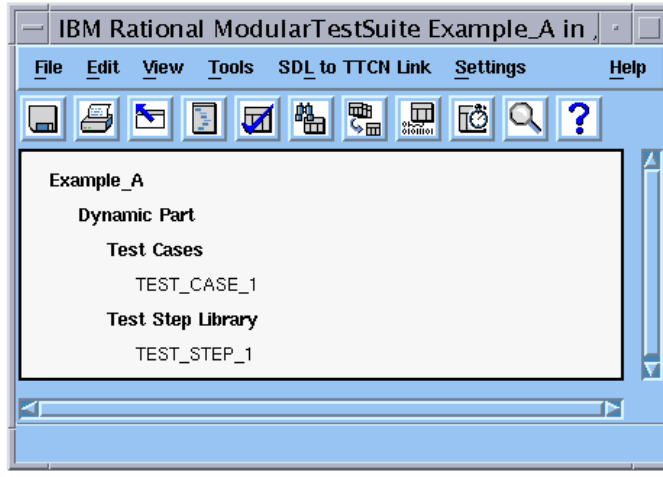


Figure 51: A sub Browser containing TEST_CASE_1 and TEST_STEP_1.

Generating Overview Tables

To complete the test suite, you also have to generate the *Test Suite Overview* tables – that is the *Test Suite Structure*, the *Test Case Index*, the *Test Step Index* and the *Default Index* tables – and the *TTCN Module Overview* – that is the *TTCN Module Structure* and the *Index* tables.

None of these tables are editable. Instead, you have to make the changes in the corresponding tables. The reason for this is that the overview tables should be consistent with the TTCN document.

To generate the overview:

1. Select *Generate Overview* from the Browser *Tools* menu.

   The *Generate Document Overview* dialog is opened.

   The TTCN Suite also generates the page numbers. You can set the start page number in this dialog, but you do not have to do that in this tutorial.

2. Click the *Generate* button in the dialog.

3. Open the *Test Case Index* table.

   You will see that the *Description* field has been generated from the additional *Description* field in the header of the test case. To make changes, you have to edit the tables from which the overview was generated and then generate the overview again.

4. Close the *Test Case Index* table.

Printing the Test Suite

You should by now have a test suite that is syntactically correct and which includes an up to date overview. If you want to, you can print it.

Printing the Entire Test Suite

To print the whole test suite:

1. Select the entire test suite.
2. Select *Print* from the Browser's *File* menu.

   The *Print Document* dialog is opened.
3. Make sure that *Output Device* names a printer that is recognized by your system.

   You do not have to change any other settings in this tutorial.
4. Click the *Print* button if you want to print the entire test suite.

Printing Parts of the Test Suite

It is also possible to print parts of the test suite.

To print, say, just the Constraint tables:

1. Select only *S1* and *R1*.
2. Select *Print* from the Browser *File* menu.

   The *Print Document* dialog is opened.
3. Make sure that *Output Device* names a printer that is recognized by your system.

   You do not have to change any other settings in this tutorial.
4. Click the *Print* button if you want to print the Constraint tables.

Creating Reports

The Reporter tool is used for presenting status information on selected items in the Browser. The information may for example be analysis status and modification date.

To make a list of the full amount of information:

1. Select the entire test suite.
2. Select *Reporter* from the Browser *Tools* menu.

   The *Reporter* dialog is opened.

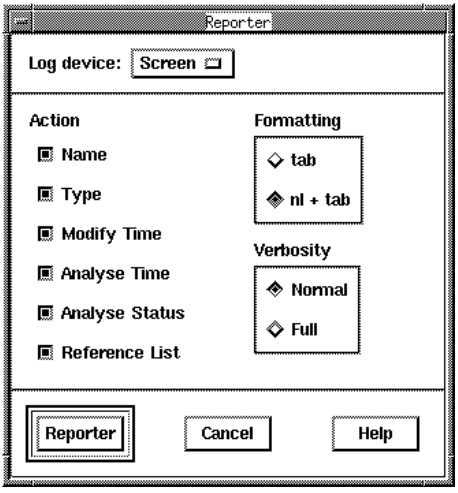


Figure 52: The Reporter dialog

3. Set the options as shown in Figure 52, that is select all options in the list of actions, normal verbosity and the formatting *nl + tab*. The latter means that each list entry will be printed on a *n*ew *l*ine and the components in the list entry will be separated by tabs.
4. Click the *Reporter* button.

   The options set in the dialog will produce the full amount of information available, a portion of which is illustrated below:

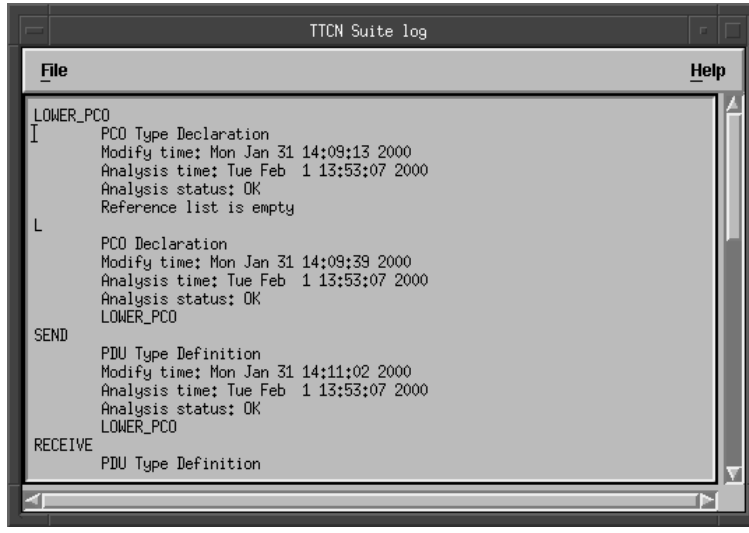


Figure 53: Test suite information

Note that, as with all the TTCN Suite log files, this file can be saved. This may be useful if you wish to print the file or process it with non-TTCN Suite tools.

If you want to, you can apply the Reporter again and experiment with other settings.

You may also find it practical to apply the Find Table tool to text that is marked in the log window:

1. Mark the name of a table in the log window
2. Select *Find Table* from the Browser *Tools* menu.

   The table is found and opened.

Exporting and Importing

In the following examples, you will use the menu choices *Convert to MP* in the Browser and *Convert to GR* in the Organizer for exporting and importing TTCN-MP documents from and to the TTCN Suite. However, you can also use *Save As* in the TTCN Suite for changing the document format. In addition, TTCN-MP documents may be opened directly in the TTCN Suite.

What You Will Learn

- To export a test suite to TTCN-MP format
- To import a test suite from TTCN-MP format

Exporting a Test Suite to MP Format

You shall now convert the test suite to TTCN-MP format:

1. Select the entire test suite in the Browser window.
2. Select *Convert to MP* from the *File* menu.

   The *Convert to MP* dialog is opened, in which you may specify the name of the new document and where it is to be saved.

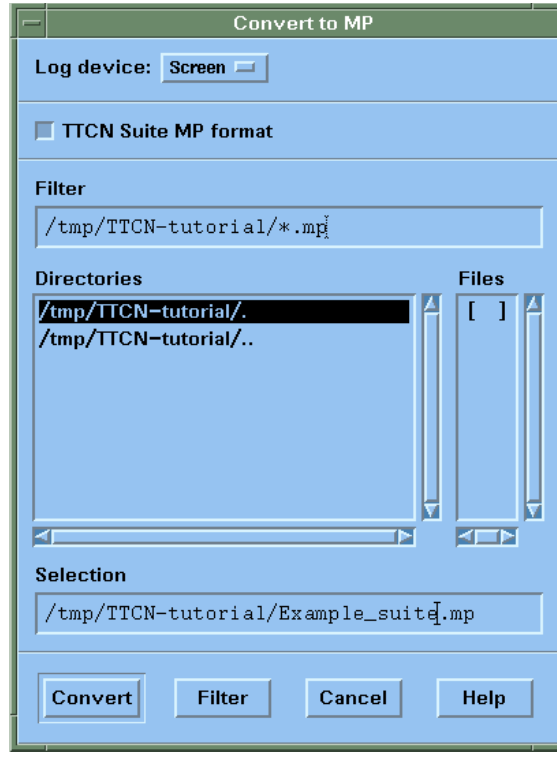


Figure 54: The Convert to MP dialog

3. Select the directory path in the *Directories* box.
4. Type the file name that you wish the exported file to be called. In this example the file name is **`Example_suite.mp`**.

   Note the suffix `.mp` (used by convention). You are now ready to convert the file.
5. Click *Convert* to export the test suite.

   The TTCN-MP file for the test suite can now be found in the file `Example_suite.mp`. You can check it if you like.

Importing a Test Suite from MP Format

Your next task is to import the TTCN-MP file that you have just exported. Do not close the existing open test suite – you need that later.

1. Select *Convert to GR* from the *Generate* menu in the Organizer. The *Convert to GR* dialog is opened:

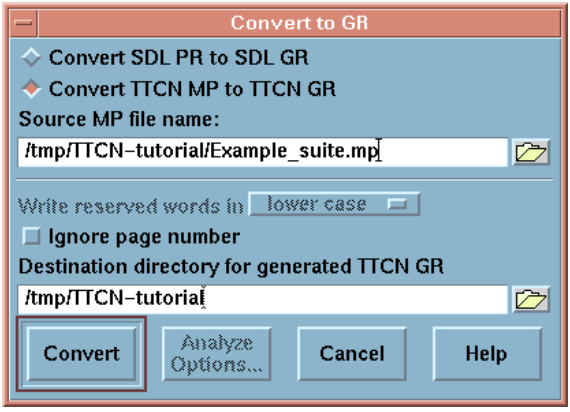


Figure 55: The Convert to GR dialog

2. Select *Convert TTCN MP to TTCN GR.*
3. Click the folder icon of the *Source MP file name* field. The *Select Source MP file* dialog is opened.
4. Click the file that you want to convert, when you have found it in the *Files* list. In this case the file is `Example_suite.mp`.
5. Click *OK*.
6. Click the folder icon of the *Destination directory for generated TTCN GR* field. The *Select Destination Directory* dialog is opened.
7. Select the destination directory, i.e. where you want the converted document to be saved.
8. Click *OK*.
9. Click the *Convert* button in the *Convert to GR* dialog.

The file is converted and is stored in a file with a generated file name in the given directory. The generated file name is the MP file name with the extension `.itex`. This is shown in the Organizer log.

Note:

You can open and work with MP-files just as you do with GR-files. All conversions to and from MP will then be done transparently.

Now perform the following:

1. Select the *TTCN Test Specification* chapter in the Organizer.
2. Select *Add Existing* in the *Edit* menu in the Organizer. The *Add Existing* dialog is opened.
3. Select the file that you just converted (`Example_suite.itex`) and click *OK*.

   Either type the filename and the path or click the folder button to select the file.
4. Make sure that the *Show in editor option* is selected.
5. Click *Add*.

   The test suite is now added in the TTCN Test Specification chapter and it is opened in the Browser.
6. In the Browser, rename this test suite as **`Example_B`**.
7. Expand the entire test suite.
8. Analyze the entire test suite.
9. Do **not** close *Example_B*, you will need it in the following exercise.

Replacing, Comparing and Merging

In this exercise you are going to merge the two test suites, the *Example_A* and the *Example_B* that you have just created. First you should change some names in one test suite though – to make it different – and then compare it with the other.

What You Will Learn

- To start search for a specified pattern and to replace matches
- To compare two TTCN documents
- To merge two TTCN documents

Searching and Replacing

The first thing you should do is to change some text in *Example_B*. The reason for this is that there would be no use comparing the documents – which you are soon going to do – if they both are the same.

You can start the search from both the Browser and the Table Editor, and in this case you will do it from the Browser:

1. In the Browser displaying *Example_B*, click the *Search* button.

   The Table Editor will be opened, and as you can see it is empty.

2. Type `1` in the *Look for* field
3. Click the *Search* button.

   The first table of the test suite containing `1` is opened and `1` is highlighted.

4. Type `2` in the *Change to* field.

5. Click the *Replace* button.

   `1` will be changed to `2` and it is no longer highlighted.

6. Click the *Search* button again.

   This will find another occurrence of `1`.

7. Click the *Proceed* button.

   This is the same as first clicking *Search* and then *Replace*. Another occurrence of `1` will be found.

8. Click the *Replace all* button.

   This will change all occurrences of `1` to `2`. The Table Editor will display the table in which the final replacement was made and the status bar will show the total number of replacements.

Comparing Two Documents

When you have changed some text in *Example_B*, you can compare it to *Example_A* to see if any similarities exist:

1. Ensure that the test suites *Example_A* and *Example_B* are opened.
2. Select the entire test suite *Example_B*
3. Select *Compare* from the Browser *Tools* menu.

   The *Compare Documents* dialog is opened:

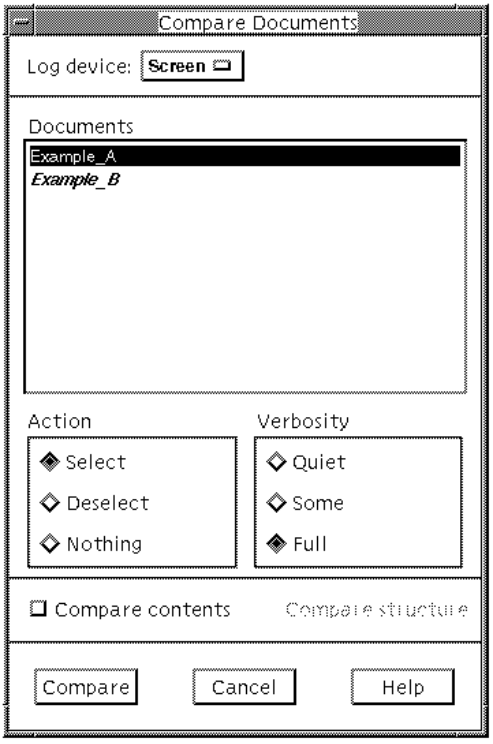


Figure 56: The Compare dialog

4. Make the dialog settings as depicted in Figure 56.

   This will compare the test suite *Example_A* with the selection that you made in *Example_B*, in this case the entire test suite.

   Also make sure that *Example_A* is selected in the *Documents* field in the dialog.

5. Click the *Compare* button.

   The comparison will deselect all items in the Browser for *Example_B* that do **not** match with items in *Example_A*. What remains are the items that are the **same** in both test suites, namely the lower PCO *L* and the PDUs *SEND* and *RECEIVE*. These tables will also be presented in the TTCN Suite log.

6. Delete the selected items from *Example_B*.

   The only items left now are the ones that differ between the two test suites. You are now ready to try a merge.

Merging Two Documents

The Merge tool is used for merging one TTCN document (complete or partial) into another. This only works on condition that the documents do not conflict, that is, if an item in one document has the same name as another item in the other document. This will not be a problem now, as you have already compared the documents and deleted the similarities.

The final exercise in this tutorial is that you should merge what is left of *Example_B* into *Example_A*:

1. Select the entire suite *Example_B*.
2. Select *Merge* from the Browser *Tools* menu.

   The *Merge Documents* dialog is opened:

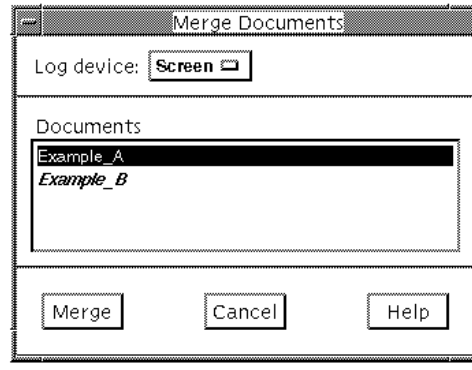


Figure 57: The Merge Documents dialog

3. Select *Example_A* in the *Documents* chapter. That is the suite that you are going to merge *Example_B* into
4. Click the *Merge* button.

   If you look at the *Example_A* Browser, you should see that the suites have been merged into one. This is also depicted in Figure 58.

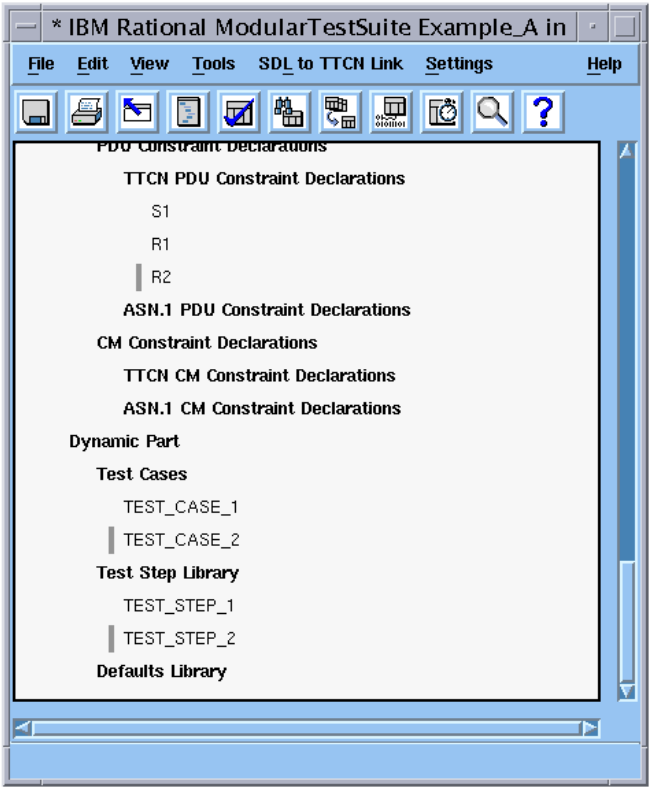


Figure 58: The Example_B has been merged into Example_A

So Far...

You should now have learned how to create a test suite and what the test suite looks like in the Browser. You have also learned to create and edit tables and how to work with the Table Editor. Other things you should have learned is how to analyze the test suite and how to find errors.

Some other TTCN Suite functionality that you have used are selecting, generating the overview, printing, making reports, converting to MP and back to GR, replacing, comparing and merging.

By practicing this tutorial you have probably not achieved any knowledge about TTCN. If you want to know more about the basics of TTCN you may read the Methodology Guidelines. That volume is, however, not a hands-on tutorial on TTCN.

The following tutorial is chapter 7, *Tutorial: SDL and TTCN Integrated Simulator (U)*. Before you start practising that, you should know how to use the SDL Simulator. A tutorial on that can be found in *chapter 4, Tutorial: The SDL Simulator, in the Getting Started*.

Chapter

6

Tutorial: SDL and TTCN Integrated Simulator (W)

This is an introductory tutorial to the SDL and TTCN Integrated Simulator in Windows and an example of a complete test session with Cbasic. The tutorial is divided into sections that describes steps that are performed during a typical test session.

> **Note: Windows version**
>
> **This is the Windows version of the tutorial. The UNIX version is chapter 7, *Tutorial: SDL and TTCN Integrated Simulator (U)*.**

Purpose of This Tutorial

The purpose of this tutorial is to make you familiar with the SDL and TTCN Integrated Simulator. When you have read and practised this tutorial you will know the actions required to execute a simulated test.

It is assumed that you know how to use the SDL Simulator. More information can be found in *chapter 4, Tutorial: The SDL Simulator, in the Getting Started* and in *chapter 49, The SDL Simulator, in the User´s Manual*.

The Steps in a Test Session

When a TTCN test suite and an SDL Suite system already exists, this is what you do in a typical test session:

1. Generate and start the SDL and TTCN Integrated Simulator.
2. Generate and start the SDL Simulator.
3. Switch to simulation mode in the TTCN Suite.
4. Set up the communication between the SDL and TTCN Integrated Simulator and SDL Simulator.
5. Start the simulation.
6. Start the TTCN-SDL simulation.
7. Perform the simulation.

The Test System

This section describes the system that is going to be tested and what roles the SDL and TTCN Integrated Simulator and SDL Simulator play in the test.

An SDL system called `ABP` will be used in this tutorial. The `ABP` system implements a small OSI stack containing a network layer, a data-link layer and a physical layer. The network component on one side is handled by the TTCN Suite and on the other side by an SDL process.

The data-link layer implements a very simple protocol known as the Alternating Bit Protocol (ABP).

The network protocol is very simple protocol, where all messages are echoed back as an acknowledgment that they have been received.

This setup gives us the ability to send messages from the TTCN Suite to the network peer simulated by the SDL Suite, and to receive ac-

knowledgments in return. If the answer is identical to the message, we have managed to communicate successfully.

What we want to test is the ability of the Data-Link layer's protocol to deliver, even when messages are lost on the way. To enable this, we have introduced an extra channel between the SDL and TTCN Integrated Simulator and the SDL simulator, on which commands can be sent to make the physical layer lose messages.

The ABP system is stored in the TTCN Suite installation, in the sub-directory `examples\bitprotocol`. If you have a default installation, the complete path is `C:\IBM\Rational\SDL_TTCN_Suite6.3\examples\bitprotocol`.

Setting Up a Simulation

What You Will Learn

- To set up the SDL and TTCN Integrated Simulator and SDL simulator
- To generate and build simulator executables
- To initialize the communication between the simulators
- To start the simulators

When you are finished with this section, the simulators should be ready to run a TTCN Suite test on a simulated SDL system.

Generating and Starting the SDL and TTCN Integrated Simulator

The first thing you should do is to generate and start the SDL and TTCN Integrated Simulator:

1. Start TTCN Suite.
2. In the Organizer, load the system *ABP* from the file `abp.sdt`.

   You can find this file in `C:\IBM\Rational\SDL_TTCN_Suite6.3\examples\bitprotocol` (where `C:\IBM\Rational\SDL_TTCN_Suite6.3` is the default installation directory).

   You may also find it useful to copy the `bitprotocol` directory and the files included, to another directory. In any case, you have to

make sure the `bitprotocol` directories and files are **not** read-only.

3. Double-click the *abp* icon in the *TTCN Test Specification* chapter.

   This will start the TTCN Suite with the `abp.mp` test suite opened in the Browser.

4. Select the node named *abp* in the Browser.

5. Select *Generate Code* from the *Build* menu.

   The *Analyzer/TTCN to C Compiler Settings* dialog is opened:

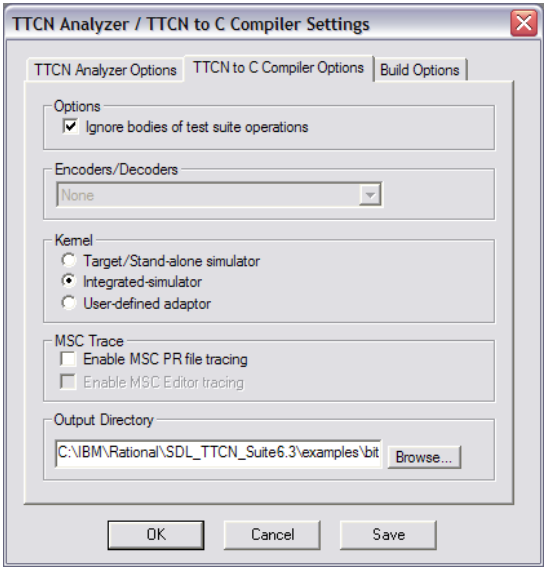


Figure 59: The TTCN to C compiler options

6. In the *TTCN to C Compiler Options*, make sure that *Integrated-simulator* is selected and that *Ignore bodies of test suite operations* is checked.

7. Select an output directory that has sufficient space and where you have write permission.

8. In the *Build Options*, ensure that the *Run Make* option is selected.

9. You may also have to change the *Makefile Type*.
10. Click *OK* to start the generation and compilation of the SDL and TTCN Integrated Simulator executable.

    The TTCN Suite will now generate and build the SDL and TTCN Integrated Simulator executable. In the Log Manager window, you can see the what is happening.

11. Click the *Start Simulator* button in the TTCN Suite tool bar.

    This is a toggle button, which means that if it is depressed, the simulator is in simulation mode. If the button is released, simulation is ended.

    The *Simulator Toolbar* is available whenever this button is depressed.

    – You can also select *Invoke Simulator* from the *Simulate* menu.

    A file selection dialog is opened.

12. Select `abp.exe` in the dialog.

    The TTCN Suite simulation executable has the same name as the test suite that you are simulating.

    This step will put the TTCN Suite in simulation mode, and it is now ready to simulate.

Generating and Starting the SDL Simulator

Now you have to set up the SDL simulator as well.

1. In the Organizer, select the system diagram *ABP* from the *SDL System Structure* chapter.
2. Click the *Simulate* button in the tool bar.

    – You may also want to check the settings in the *Make* dialog (opened from the *Generate* menu) before you generate and compile the simulator. If you select *Make* or *Full Make* in that dialog, you will have to open the Simulator UI and load the simulator after that.

The SDL simulator generates and builds a simulator executable. The simulator window is displayed with the simulator started. You are now ready to simulate.

Getting the Simulators to Communicate

The next step you should perform is to initialize the communication between the simulators:

- Enter the command **`start-itex`** in the SDL simulator window.

  This instructs the SDL simulator to communicate with the SDL and TTCN Integrated Simulator.

Performing the Simulation

What You Will Learn

- To single step a test case
- To run a test case at full speed
- To run a test batch
- To set and delete breakpoints
- To abort a simulation

Single Stepping Test Cases

1. In the TTCN Browser, select the test case *TC_01*.

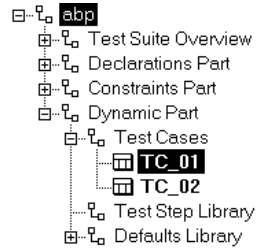


Figure 60: The test case TC_01 is selected in the Browser

This is the test case that you are going to execute.

2. Click the *Step Simulation* button.

   Each time you click this button, the SDL and TTCN Integrated Simulator executes one step of TTCN code.

   A Table Editor is opened. In the first line of TTCN code, there is a colored bar.

> **Note:**
> This bar indicates which line will be executed the **next** time you click the *Step Simulation* button.

3. Enter the command `go-forever` in the SDL simulator window.

The SDL simulator is now executing and it is possible to perform a simulation.

4. Click the *Step Simulation* button repeatedly until you reach the end of the test case.

When you have reached the end of the test case, the bar that indicates the current line will change color. The color indicates the result of the execution of the test case. The colors are as follows:

Color	Verdict
Green	PASS
Yellow	INCONC
Red	FAIL

At this point the SDL Simulator is running at full speed. This is normal and you can safely ignore it. The cause of this is a timer in the simulated system that fires repeatedly while waiting for input from the SDL and TTCN Integrated Simulator.

Running Test Cases at Full Speed

1. In the TTCN Browser, select the test case *TC_01* again.

   This time, you are going to execute the test case at full speed.

2. Click the *Run Simulation* button.

   The Table Editor is opened. The lines of TTCN code in the table will execute without pausing until the end of the test case is reached.

   – It is possible to stop the execution by clicking the *Pause Simulation* button

Running Test Batches

It is also possible to execute more than one test case or test group.

1. In the TTCN Browser, select the test case *TC_01* and *TC_02*.

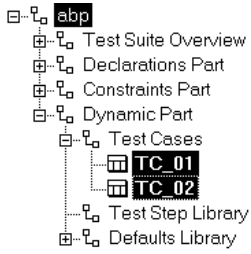


Figure 61: Selecting multiple test cases

2. Press the *Run Simulation* button.

   This will cause the SDL and TTCN Integrated Simulator to execute all the selected test cases in turn without pausing anywhere.

> **Note:**
>
> This requires an SDL Suite system that always ends in a state from where you can execute a new test case. The SDL Suite system may not require any manual resetting between test cases.

Toggling Breakpoints

1. Set the cursor on line 3 in the test case *TC_01*.

2. Press the *Toggle Breakpoint* button.

   This will set a breakpoint on line 3.

   The button will toggle the breakpoint status on the line that the cursor is placed on. If there is a breakpoint set on the line, it will be removed. If there is not a breakpoint there, one will be set.

3. Select test case *TC_01* in the Browser.
4. Press the *Run Simulation* button.

The SDL and TTCN Integrated Simulator will run without pausing between lines of TTCN code. When it reaches the line where the breakpoint was set, it will stop.

5. Set the cursor on line 3 in the test case *TC_01* .
6. Press the *Toggle Breakpoint* button.

   This will remove the breakpoint from line 3.
7. Press the *Run Simulation* button.

   The simulator now continues to run until it reaches the end of the test case.

Ending a Simulation

It is possible to end a simulation in the middle of an execution:

1. Select test case *TC_01* in the browser.
2. Click the *Step Simulation* button twice.

   This will place the current line on line 2 in the test case.

   You now realize that the execution was a mistake and you want to abort it.
3. Click the *Abort Simulation* button.

   This aborts the execution of the test case for the SDL and TTCN Integrated Simulator.
4. In the SDL simulator window, select *Restart* in the *File* menu.

   This restarts the SDL simulator. This step is necessary because otherwise the SDL Simulator and the SDL and TTCN Integrated Simulator would be out of sync.

   To continue co-simulation, you will need to enter the command **`start-itex`** in the SDL simulator, start running the test in the SDL and TTCN Integrated Simulator, and then enter the command **`go-forever`** in the SDL simulator.

Taking a Look at the Log Window

Every execution of the SDL and TTCN Integrated Simulator produces a log. This log is output in the Log Manager. Every line in the test case that is executed will be present in the log.

This makes it possible to examine a trace of the execution after it has finished. This is valuable when running the test cases in a batch.

The log contains the test component, table name, line within the table, behavior, constraint and verdict.

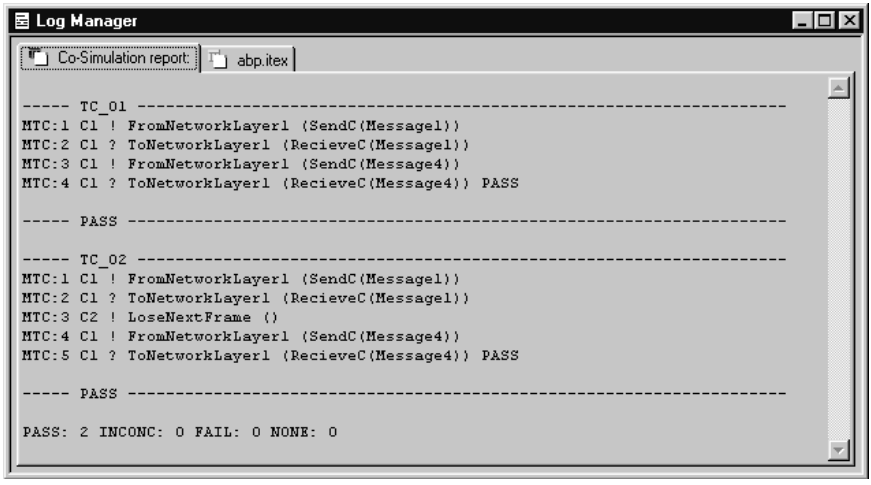


Figure 62: An execution log

Every test case also produces a conformance log. This log contains detailed information about the tests performed. It is very useful when determining why a test case failed or succeeded.

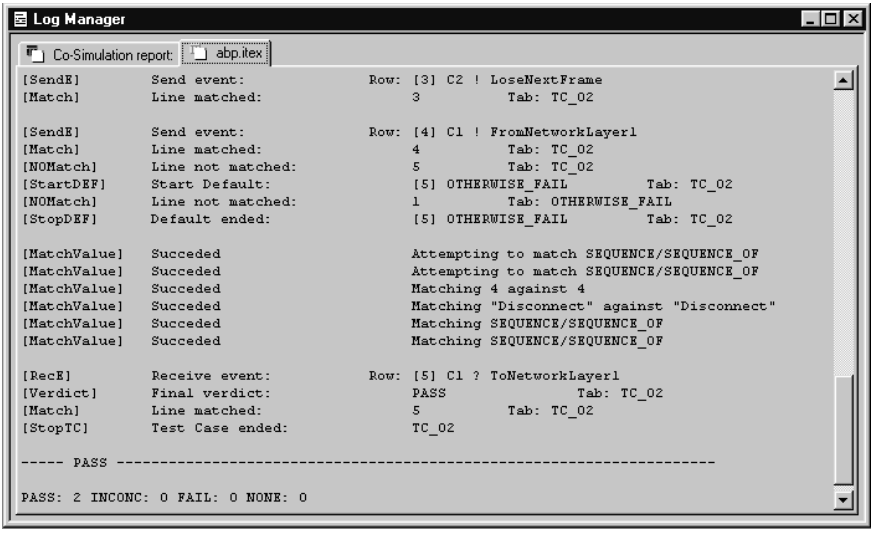


Figure 63: A conformance log

Chapter

7

Tutorial: SDL and TTCN Integrated Simulator (U)

This is an introductory tutorial to the SDL and TTCN Integrated Simulator on UNIX and an example of a complete test session with Cbasic. The tutorial is divided into sections that describes steps that are performed during a typical test session.

> **Note: UNIX version**
>
> **This is the UNIX version of the tutorial. The Windows version is chapter 6, *Tutorial: SDL and TTCN Integrated Simulator (W)*.**

Purpose of This Tutorial

The purpose of this tutorial is to make you familiar with the SDL and TTCN Integrated Simulator. When you have read and practised this tutorial you will know the actions required to execute a simulated test.

In this tutorial it is assumed that you know how to use the SDL Simulator. More information can be found in chapter 4, *Tutorial: The SDL Simulator, in the Getting Started* and in chapter 49, *The SDL Simulator, in the User´s Manual*.

The Steps in a Test Session

When a TTCN test suite and an SDL Suite system already exists, this is what you do in a typical test session:

1. Generate and start the SDL and TTCN Integrated Simulator.
2. Generate and start the SDL simulator.
3. Set up the communication between the SDL and TTCN Integrated Simulator and SDL simulator.
4. Start the simulation.
5. Start the TTCN-SDL simulation.
6. Perform the simulation.

The Test System

This section describes the system that is going to be tested and what roles the SDL and TTCN Integrated Simulator and SDL simulator play in the test.

An SDL system called `ABP` will be used in this tutorial. The ABP system implements a small OSI stack containing a network layer, a data-link layer and a physical layer. The network component on one side is handled by the TTCN Suite and on the other side by an SDL process.

The data-link layer implements a very simple protocol known as the Alternating Bit Protocol (ABP).

The network protocol is very simple protocol, where all messages are echoed back as an acknowledgment that they have been received.

This setup gives us the ability to send messages from the TTCN Suite to the network peer simulated by the SDL Suite, and to receive ac-

knowledgments in return. If the answer is identical to the message, we have managed to communicate successfully.

What we want to test is the ability of the Data-Link layer's protocol to deliver, even when messages are lost on the way. To enable this, we have introduced an extra channel between the SDL and TTCN Integrated Simulator and the SDL simulator, on which commands can be sent to make the physical layer lose messages.

The ABP system is stored in the TTCN Suite installation, in the sub-directory `examples/bitprotocol`. If you have set up the `$telelogic` environment variable, the complete path is `$telelogic/examples/bitprotocol`.

Setting Up a Simulation

What You Will Learn

- To set up the SDL and TTCN Integrated Simulator and SDL simulator
- To generate and build simulator executables
- To initialize the communication between the simulators
- To start the simulators

When you have finished with this section, the simulators should be ready to run a TTCN Suite test on a simulated SDL system.

Generating and Starting the SDL and TTCN Integrated Simulator

The first thing you should do is to generate and start the SDL and TTCN Integrated Simulator:

1. Start TTCN Suite.
2. In the Organizer, load the system *ABP* from the file `abp.sdt`.

   You can find this file in `$telelogic/examples/bitprotocol` (where `$telelogic` points to the installation directory).
3. Double-click the *abp* icon in the *TTCN Test Specification* chapter.

   This will start the TTCN Suite with the `abp.mp` test suite opened.
4. Select the node named *abp* in the TTCN Browser.

5. From the *Tools* menu, select *Make*.

   The *TTCN Suite Make* dialog is opened.

6. Make sure that the *Use standard kernel* option is checked and that the *Simulation* command is selected.

> **Note:**
> The dialog will show up the first time you build your simulator. In later builds it will remember the settings.

7. Select the *Directory for generated files* to point to a location where you have write permission and sufficient space for the generated files.

8. Click the *Full Make* button to start the generation and compilation of the SDL and TTCN Integrated Simulator executable.

   The TTCN Suite will now generate and build the SDL and TTCN Integrated Simulator executable.

9. Click the *Start Simulator* button on the TTCN Suite tool bar.

   This will start the SDL and TTCN Integrated Simulator and open the simulator window. The TTCN Suite is now ready to simulate.

Generating and Starting the SDL Simulator

Now you have to set up the SDL simulator as well:

1. In the Organizer, select the system diagram *ABP* from the *SDL System Structure* chapter.

2. Click the *Simulate* button in the tool bar.

   – You may also want to check the settings in the *Make* dialog (opened from the *Generate* menu) before you generate and compile the simulator. If you select *Make* or *Full Make* in that dialog, you will have to open the Simulator UI and load the simulator after that.

   The SDL simulator generates and builds a simulator executable. The SDL simulator window is displayed with the simulator started. You are now ready to simulate.

Getting the Simulators to Communicate

The next step you should perform is to initialize the communication between the simulators.

- Enter the command **`start-itex`** in the SDL simulator window.

  This instructs the SDL simulator to communicate with the SDL and TTCN Integrated Simulator.

Performing the Simulation

What You Will Learn

- To single step a test case
- To run a test case at full speed
- To run a test batch
- To set and delete breakpoints
- To abort a simulation

Single Stepping Test Cases

1. In the SDL and TTCN Integrated Simulator window, select the test case *TC_01* and then click the right arrow that appears.

   This selects the test case that we will execute by adding it to the *Selected* box:

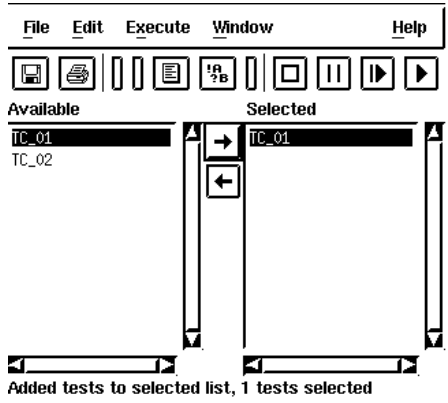


Figure 64: Selecting a test case to execute

2. Click the *Step Simulation* button

   Each time you click this button, the SDL and TTCN Integrated Simulator executes one step of TTCN code.

   A Table Editor is opened. In the first line of TTCN code, there is a colored bar.

Note:

This bar indicates which line will be executed the **next** time you click the *Step Simulation* button.

3. Enter the command `go-forever` in the SDL simulator window.

   The SDL simulator is now executing and it is possible to perform a simulation.

4. Click the *Step Simulation* button repeatedly until you reach the end of the test case.

Running Test Cases at Full Speed

1. In the SDL and TTCN Integrated Simulator window, make sure that the test case *TC_01* is still in the *Selected* box. If it is not, select the test case again and click the right arrow.

Note:

If the test case already is in the *Selected* box, you do not have to select it again. If you select it again, you are actually telling the SDL and TTCN Integrated Simulator that you want to execute the test case twice.

2. Click the *Run Simulation* button.

   The simulator now starts executing lines of TTCN code without pausing. This will continue until the end of the test case is reached or until you click the *Pause Simulation* button.

   As you can see, the colored bar in the Table Editor will change rapidly.

At this point the SDL simulator is running at full speed. This is normal and you can safely ignore the SDL simulator. The cause of this is a timer in the simulated SDL system that fires repeatedly while waiting for input from the SDL and TTCN Integrated Simulator.

Running Test Batches

It is also possible to execute more than one test case or test group.

1. In the SDL and TTCN Integrated Simulator window, deselect the test case *TC_01* in the *Available* box. You do this by clicking on the test case.
2. Select the test case *TC_02* and click the right arrow. The test case is added to the *Selected* box after *TC_01*.

   You can perform this operation with as many test cases as you wish. You can add them in any order. The *Selected* box tells you in what order the test cases will be executed.
3. Press the *Run Simulation* button.

   This will cause the SDL and TTCN Integrated Simulator to execute all the selected test cases in turn without pausing anywhere. Another Table Editor is opened for the second test case.

> **Note:**
>
> This requires an SDL Suite system that always ends in a state from where you can execute a new test case. The SDL Suite system may not require any manual resetting between test cases.

Toggling Breakpoints

1. Make sure that the SDL and TTCN Integrated Simulator window is active and press `<Ctrl+B>`.

   This will open the *Breakpoints* window. In this window you can add and delete breakpoints.
2. Select *Add* from the *Edit* menu.

   This adds an empty breakpoint definition to the breakpoint editor.
3. Click in the edit field at the bottom of the *Breakpoints* window and type in **`TC_01 3`** followed by `<Return>`.

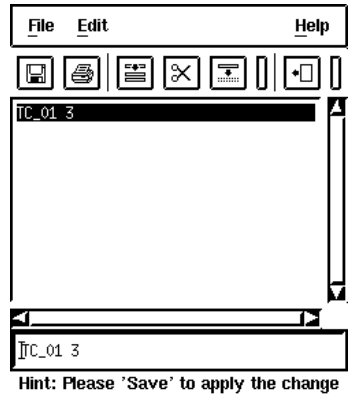


Figure 65: A breakpoint in the Breakpoints window

4. Select *Save* from the *File* menu.
5. Type in a filename with the extension `.ttb` and select *OK*.

   This will set a breakpoint on line 3.
6. In the simulator window, make sure that only the test case *TC_01* is selected.
7. Press the *Run Simulation* button.

   The SDL and TTCN Integrated Simulator will run without pausing between lines of TTCN code. When it reaches the line where the breakpoint was set, it will stop, as you can see in the Table Editor.
8. Select the breakpoint line we just added in the *Breakpoints* window.
9. Select *Delete* from the *Edit* menu.

   This will remove the breakpoint from line 3.
10. Select *Save* from the *File* menu.
11. Select *Close* from the *File* menu.
12. Click the *Run Simulation* button.

    The simulator now continues to run until it reaches the end of the test case.

Ending a Simulation

It is possible to end a simulation in the middle of an execution

1. Click the *Step Simulation* button twice.

   This will place the current line on line 2 in the test case.

   You now realize that the execution was a mistake and you want to abort it.

☐ 2. Click the *Abort Simulation* button.

   This aborts the execution of the test case for the SDL and TTCN Integrated Simulator.

3. In the SDL simulator window, select *Restart* in the *File* menu.

   This restarts the SDL simulator. This step is necessary because otherwise the SDL Simulator and the SDL and TTCN Integrated Simulator would be out of sync.

   To continue co-simulation, you will need to enter the command **`start-itex`** in the SDL simulator, start running the test in the SDL and TTCN Integrated Simulator, and then enter the command **`go-forever`** in the SDL simulator.

The Execution Trace

Every execution of the SDL and TTCN Integrated Simulator produces a trace. This trace is output in the trace window. Every line in the test case that is executed will be present in the trace window.

This makes it possible to examine a trace of the execution after it has finished. This is valuable when running the test cases in a batch.

- To see the execution trace, select *Execution Trace* in the *Window* menu of the SDL and TTCN Integrated Simulator.

The trace contains the test component, table name, line within the table, behavior, constraint and verdict.

```
MTC                TC_01:4    C1 ? ToNetworkLayer1          RecieveC(Message4)  PASS
MTC                TC_01:1    C1 ! FromNetworkLayer1        SendC(Message1)
MTC                TC_01:2    C1 ? ToNetworkLayer1          RecieveC(Message1)
MTC                TC_01:3    C1 ! FromNetworkLayer1        SendC(Message4)
MTC                TC_01:4    C1 ? ToNetworkLayer1          RecieveC(Message4)  PASS
MTC                TC_02:1    C1 ! FromNetworkLayer1        SendC(Message1)
MTC                TC_02:2    C1 ? ToNetworkLayer1          RecieveC(Message1)
MTC                TC_02:3    C2 ! LoseNextFrame
MTC                TC_02:4    C1 ! FromNetworkLayer1        SendC(Message4)
MTC                TC_02:5    C1 ? ToNetworkLayer1          RecieveC(Message4)  PASS
```

Figure 66: An execution trace

The Conformance Log

Every test case produces a conformance log. This log contains detailed information about the tests performed. It is very useful when determining why a test case failed or succeeded.

- To see the conformance log, select *Conformance Log* in the *Window* menu of the SDL and TTCN Integrated Simulator.

```
[StartDEF]     Start Default:              [2] OTHERWISE_FAIL         Tab: TC_0
[NOMatch]      Line not matched:           1          Tab: OTHERWISE_FAIL
[StopDEF]      Default ended:              [2] OTHERWISE_FAIL         Tab: TC_0
[MatchValue]   Succeded                    Attempting to match SEQUENCE/SEQUENC
[MatchValue]   Succeded                    Attempting to match SEQUENCE/SEQUENC
[MatchValue]   Succeded                    Matching 1 against 1
[MatchValue]   Succeded                    Matching "Connect" against "Connect"
[MatchValue]   Succeded                    Matching SEQUENCE/SEQUENCE_OF
[MatchValue]   Succeded                    Matching SEQUENCE/SEQUENCE_OF
[StopTC]       Test Case ended:            TC_01

== Verdict: NONE ============================================================
```

Figure 67: A conformance log

Chapter 7 Tutorial: SDL and TTCN Integrated Simulator

Chapter

8 *Tutorial: The TTCN Link*

This is a tutorial for TTCN Link. It is intended to give a guided tour through some of the facilities provided.

To be able to follow this tutorial, you need some experience of both SDL and TTCN as well as the SDL Suite and TTCN Suite.

Purpose of This Tutorial

The purpose of this tutorial is to make you familiar with the TTCN Link function and understand its usability. You are supposed to read this tutorial sequentially and practice the exercises.

> **Note: Platform differences**
>
> This tutorial is possible to run on both the UNIX and Windows platform, and is described in a way common to both platforms. In case there are differences between the platforms, this is indicated by texts like “on UNIX”, “Windows only”, etc.
>
> When such platform indicators are found, please pay attention only to the instructions for the platform you are running on.

More about TTCN Link

TTCN Link is a CATG (Computer Aided Test Generation) tool that enables efficient development of TTCN test suites based on SDL specifications. By using TTCN Link, you can automatically generate the declarations from the SDL specification and interactively build test cases. This ensures consistency between the test suite and the specification. TTCN Link also provides you with an integrated SDL/TTCN environment with access to the specification directly from the test case.

Using TTCN Link

The inputs needed to use TTCN Link for test case development, are an SDL specification and a test suite structure with test purposes. TTCN Link supports test suite development in three different ways:

- Keeping the test suite consistent with the SDL specification.
- Supporting the test suite developer with information about the SDL specification, making the actual TTCN coding easier.
- Providing access to the SDL specification directly from the test case.

Three distinct phases can be distinguished when TTCN Link is used:

- Preparation of the SDL specification to be used in the test case development.
- Generation of the TTCN declarations.
- Development of the TTCN test cases and constraints.

Taking a Look at the SDL System

In this tutorial, you will use the example SDL system called `inres` (short for Initiator-Responder). This system is an example of a simplified communication protocol intended to give a secure transfer of information over an unsafe communication medium. It provides a one-way, connection-oriented communication service that uses sequence numbers and retransmission to ensure that messages sent to the initiator side, are correctly received at the responder side.

Start this tutorial by taking a look at the `inres` SDL system:

1. Copy the `inres` directory from `$telelogic/sdt/examples/` **(on UNIX)** or `\IBM\Rational\SDL_TTCN_Suite6.3\sdt\examples\` **(in Windows)** to your working directory and make sure that you have write permission.
2. Start TTCN Suite.
3. Open the example SDL system called `inres.sdt` located in the `inres` directory.
4. Select *SDL Overview* in the *Generate* menu in the Organizer, to generate an overview of the SDL system.

   The *Generate SDL Overview* dialog is opened. You do not have to change any settings in this dialog.
5. Click the *Generate* button in the *Generate SDL Overview* dialog.

   The SDL Editor is opened and displays the overview.

   The block *Station_Ini* in *System inres* is the protocol that you are going to create a test for. However, due to the test architecture that is to be used, the tester can not access the lower service access point of the protocol. Since TTCN Link assumes that the channels to/from the environment are accessible for the tester (and will correspond to

PCOs in the test suite), the block *Medium*, that models the underlying service provider, is included in the SDL system.

Generating a TTCN Link Executable

To create a test suite based on an SDL system, it is necessary to begin by creating an executable program that will be used during the test suite development. Such programs, which are created by the SDL to C Compilers, are called *Link Executables*. Sometimes they will also be referred to as *state space generators*, since the purpose of these programs is to generate the combined state space of the SDL system and the TTCN test case.

1. Select *Make* in the *Generate* menu in the Organizer, to create a Link Executable for the SDL system.

   – Since you probably have not saved the system since you generated the overview, you will be prompted to do so before the *SDL Make* dialog can be opened.

2. Make sure that the following options are set in the *SDL Make* dialog:

   – *Code generator: Cbasic*
   – *Prefix: Full*
   – *Separation: No*
   – *Capitalization: Lower case*

3. Make also sure that the following options are turned on:

   – *Analyze & generate code*
   – *Makefile*
   – *Compile and link*

4. Select *Generate Makefile*.

5. Select a *TTCN Link* kernel for a suitable compiler in the *Use standard kernel* option.

6. Click the *Full Make* button.

   A TTCN Link executable will now be generated from the SDL specification. You can open the Organizer log to see the result of the make process.

Now the preparations are completed and you can start developing a new test suite.

Creating a Test Suite

What You Will Learn

- To specify the state space generators to use, in two different ways
- To generate the TTCN declarations

Creating a New Test Suite

Start by creating a new test suite:

- Add a new test suite, for example called `inres`, to the *TTCN Test Specification* chapter. In the *Add New* dialog, make sure that the option *Show in editor* is on.

  The TTCN Suite is started and the Browser will display the new test suite.

The first step when you have created the new test suite, is to use TTCN Link to generate the declarations. Before you can do this, you have to tell the TTCN Suite what state space generator to use.

Specifying the Link Executable

Before you can generate the declarations, you have to tell the TTCN Suite which link executable to use. You can do this in two ways:

One way is to associate the test suite with the SDL system:

1. In the Organizer, select the top icon in *System inres*.
2. Select *Associate* from the *Edit* menu.

   The *Associate* dialog is opened.
3. Select the test suite that you have created.
4. Click the *OK* button.

   The association is illustrated by a new icon placed under the test suite icon.

Another way is to explicitly specify the state space generator to use:

1. In the TTCN Suite, select *Select Link Executable* from the *SDT Link* menu. (**On UNIX**, it is the menu in the Browser.)

   The *Select Link Executable* dialog is opened.

2. Select `inres_stx.exe`.

   All generators will by default be given the name `<sdl system name>_st<x>.exe`, where `<x>` is depending on which compiler was used when the executable was created.

3. Click *OK*.

Note:

In case a link executable has been selected both in the Organizer and in the TTCN Suite, the one selected in the TTCN Suite is the link executable that will be used.

Generating the TTCN Declarations

Since the SDL specification contains all information about the interfaces, this information is available to you when using TTCN Link. When you generate the TTCN declarations, the interface information is extracted from the SDL specification and included in the TTCN test suite as PCO, ASP and data type tables.

To generate the TTCN declarations:

1. In the TTCN Suite, select *Generate Declarations* from the *SDT Link* menu. (**On UNIX**, it is the menu in the Browser.)

2. Expand the declarations part of the test suite and have a look at the generated declarations.

   Objects of the following types have been generated:

   – ASN.1 type definitions
   – PCO type declarations
   – PCO declarations
   – ASN.1 ASP type definitions

   The PCO declarations, and the PCO type declarations, are generated from the channels to/from environment in the SDL system.

The ASP definitions are generated from the SDL signals that can be transported on the channels to/from the environment.

The ASN.1 type definitions are generated from the types of the parameters of the above mentioned signals.

Generated declarations will be analyzed automatically.

3. Expand the dynamic part of the test suite.

   A defaults library is generated. It contains the table *OtherwiseFail* with an *otherwise statement* for each PCO, and in addition a timeout statement that match any timeout event.

Creating a Simple Test Case

It is now time to create the first test case. The one you will develop is the test case intended to test the successful connection establishment of the protocol.

How to create constraints and test cases with TTCN Link differs between Windows and UNIX. **On UNIX**, it is possible to create constraints directly from the *Send* dialog. **In Windows**, you have to create the constraints first, before they will be visible in the *Link* dialog.

What You Will Learn

- To synchronize a test case **(UNIX only)**
- To create constraints
- To add send statements
- To add receive statements

Creating a Test Case and Constraints (in Windows)

Before you start creating the test case, you have to create a constraint for the ASP `ICONreq`:

1. Copy the table *ICONreq* from *ASN.1 ASP Type Definitions*.
2. Paste the table in the constraints part, below *ASN.1 ASP Constraint Declarations*.
3. Open this table and change the name to, for example, `ICONreq_1`.
4. Analyze the table.
5. Close the table.

Creating a Test Case and Adding Statements

You are now going to create a test case and add send and receive statements by using the *Link* dialog. When you use the *Link* dialog, the test case is synchronized with the SDL specification and you cannot edit the behavior lines directly in the table.

1. Create a new test case, for example called `test_case_1`.
2. Open the test case in the Table Editor.

3. Select *Link* from the *SDT Link* menu.

   The *Link* dialog is opened. The first thing you will add is a send statement for the ICONreq service primitive.

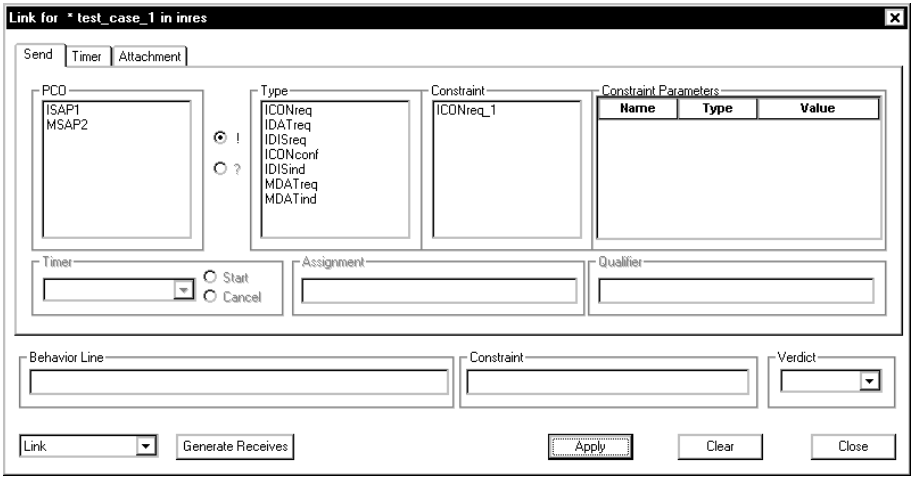


Figure 68: The Link dialog

4. In the dialog, select the following:
   - The PCO ISAP1
   - The type ICONreq
   - The constraint ICONreq_1

   The contents of the *Behavior line* and *Constraint* fields will be updated in accordance with your selection.

5. Click *Apply*.

   The following behaviour line, a send statement, is added to the test case:

Behaviour Description	Constraints Ref
`ISAP1 ! ICONreq`	`ICONreq_1`

6. Click *Generate Receives*.

   This will add a second behavior line, a receive statement, to the test case:

Behaviour Description	Constraints Ref
ISAP1 ! ICONreq	ICONreq_1
MSAP2 ? MDATind	MDATind500

A corresponding constraint will also be created.

If needed, just move the *Link* dialog to see the test case. **Do not close** the dialog.

The table MDATind500 (that was just created) is made visible in the Browser.

7. Analyze the test suite by selecting *Analyze Suite* from the *Build* menu.

Creating Another Constraint

Before you can complete the connection establishment, by adding send and receive statements for the MDATreq ASP, you have to create a new constraint:

1. In the Browser, create a copy of the constraint that was just created, MDATind500.
2. Open the copy of MDATind500 and make the following changes:
   - The name should be MDATreq_1
   - The ASP type should be MDATreq
   - The “id” part of the parameter should be CC instead of CR, which means that the constraint value should be:

```
{ mSDUType1 {id CC,
  num zero,
 data 0}}
```

3. Analyze the table.
4. Close the table.

Adding Statements to Complete the Connection

1. In the *Link* dialog, select *Clear*.

   This will refresh the dialog.

2. Select the following:

   – The PCO MSAP2
   – The type MDATreq
   – The constraint MDATreq_1

3. Click *Apply*.

   This will add a third behaviour line to the test case.

4. Click *Generate Receives*.

   The test case should now contain 4 behavior lines.

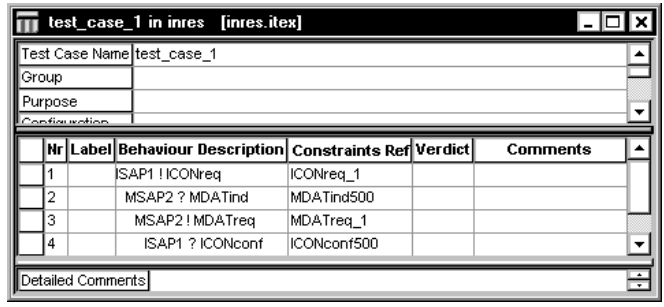


Figure 69: The complete test case

5. Close the *Link* dialog.

Creating a Test Case and Constraints (on UNIX)

1. Create a new test case, for example called `test_case_1`.
2. Open the test case in the Table Editor.

   As you can see, the Table Editor contains the extra menu *SDT Link*.
3. Select *Resynchronize* from the *SDT Link* menu.

   The test case will now be synchronized with the SDL specification. This means that the behaviour lines will be modified from the *SDT Link* menu. However, if you edit a field in the test case manually – except for comment fields – the editor will not be synchronized anymore.

   What happens behind the scene is that the state space generator is started and an initial part of the state space is explored. However, you do not have to think of that now.

Adding Statements

The first thing you will add, is a send statement for the ICONreq service primitive:

1. Select *Send* in the *SDT Link* menu. The *Send* dialog is displayed:

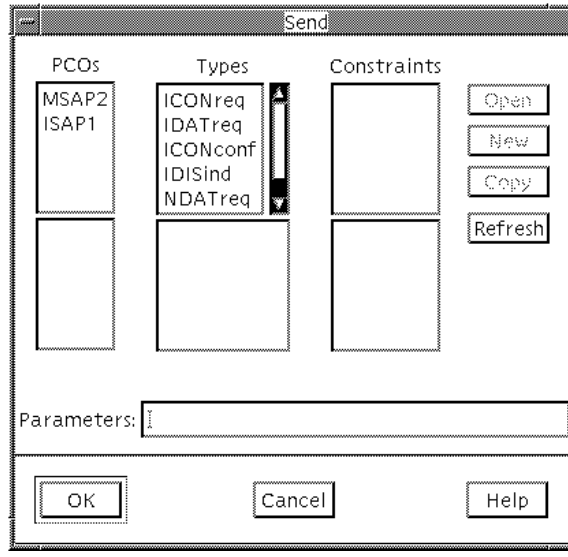


Figure 70: The Send dialog

2. Select *ISAP1* in the *PCOs* list.

   The unselected PCO is moved to the lower part of the list.

3. Select *ICONreq* in the *Types* list.

   The unselected types are moved to the lower part of the list.

4. Click the *New* button to create a constraint for this ASP.

   A Table Editor, displaying the new constraint is opened.

5. Change the name of the constraint to, for example, **`ICONReq_1`**.

6. Analyze the table.

7. Close the table.

8. Click the *Refresh* button in the *Send* dialog to make the new constraint, ICONReq_1, visible in the *Constraints* list.

   Since this is the only constraint of the selected type, it will automatically be selected.

9. Click *OK*.

   The following behaviour line, a send statement, is added to the test case:

Behaviour Description	Constraints Ref
`ISAP1 ! ICONreq`	`ICONreq_1`

10. Select *Receive* from the *SDT Link* menu.

    This adds second behavior line, a receive statement, to the test case:

Behaviour Description	Constraints Ref
`ISAP1 ! ICONreq`	`ICONreq_1`
`MSAP2 ? MDATind`	`MDATind500`

The corresponding constraint is created and analyzed.

Adding Statements to Complete the Connection

To complete the connection establishment, you should now add send and receive statements for the MDATreq ASP:

1. Select row 2 in the *Behaviour Description* column – the field containing `MSAP2 ? MDATind`.
2. Select *Send* from the *SDT Link* menu.

   The *Send* dialog will once again be opened. Now you will create a new constraint by copying an old constraint and modify it.
3. Select *MDATind500* in the *Constraints* list.
4. Click *Copy*.

   The Table Editor will appear, displaying a copy of the MDATind500 constraint.
5. Change the name to `MDATreq_1`.
6. Change the ASP type to `MDATreq`.
7. Change the “id” part of the parameter to `CC` instead of `CR`, which means that the constraint value should be:

```
{ MSDUType1 {id CC,
  num zero,
 data 0}}
```

8. Analyze the table and then close it to get back to the *Send* dialog.
9. Click *Refresh* in the *Send* dialog.
10. Select the new constraint, MDATreq_1.
11. Select the PCO MSAP2.
12. Click the *OK* button.

    The new send statement is inserted in the test case table.
13. Select *Receive* from the *SDT Link* menu once more to complete the connection establishment test case.

Taking a Look at the SDL System (Again)

What You Will Learn

- To display an SDL diagram
- To display an MSC diagram

Displaying an SDL Diagram

It is possible to display the SDL system directly from the test case:

1. Select a behaviour line in the test case.
2. Select *Show SDL* from the *SDT Link* menu.

   The SDL Editor is started, displaying all symbols in the SDL process graphs that where executed in the SDL system as a response to the currently selected row in the test case.

Displaying an MSC Diagram

It is also possible to display an MSC diagram, to illustrate and document the test case:

1. Select a behaviour line in the test case.
2. Select *Show MSC* from the *SDT Link* menu.

   The MSC Editor is started, displaying the execution of the SDL system state up to the currently selected row in the test case. This may be useful as a means to understand how unexpected receive statements are possible.

So Far...

You should now have learned how to create a link executable and how to tell the TTCN Suite which link executable to use. You should also know how to generate TTCN declarations from SDL specifications and how to create a test case that is synchronized with SDL specifications. Finally, you should have tried to generate SDL and MSC diagrams from the test case.

If you know how to use the SDL Explorer and have access to it, you are now ready to read and practise chapter 9, *Tutorial: The Autolink Tool*.

Chapter

9

Tutorial: The Autolink Tool

This tutorial is intended to guide you through some of the features of Autolink. There are some aspects of Autolink which are not covered by this tutorial. For a complete description of all Autolink facilities, please see "Using Autolink" on page 1431 in chapter 35, *TTCN Test Suite Generation, in the User´s Manual*.

To be able to follow the tutorial, some experience of the SDL Explorer is needed. Therefore you should have read and practised *chapter 5, Tutorial: The SDL Explorer, in the Getting Started*. You also need to have basic knowledge of the TTCN Suite.

The example protocol used in this tutorial is the `inres` system, the same as the one used in the TTCN Link tutorial.

Purpose of This Tutorial

The purpose of this tutorial is to make you familiar with Autolink. The tutorial is split into three parts. You are strongly recommended to read the tutorial sequentially as each exercise is founded on the previous one.

In the first exercise, you will start by creating a simple MSC. Based on this MSC, you will generate a TTCN test case and save it in a test suite. You will also learn how to manipulate constraints.

In the second session, you will develop a structured MSC. Once again a TTCN test case is generated, but this time by a direct MSC into TTCN translation. Additionally, so-called translation rules are introduced as a simple example for an Autolink configuration.

In the third part, you will use the Tree Walk algorithm to create a large number of MSC test cases automatically. You will also learn how to save and load generated test cases in Autolink. This facility will allow you to compute test cases separately and distribute test generation over several computers (provided that you have more than one license).

It is assumed that you know how to generate an SDL Explorer. In addition, you should be familiar with the basic functionality of the SDL Explorer, the MSC Editor and the TTCN Suite. You may also find it useful to have knowledge of MSC diagrams.

Note: Platform differences

This tutorial is possible to run on both the UNIX and Windows platform, and is described in a way common to both platforms. In case there are differences between the platforms, this is indicated by texts like “on UNIX”, “Windows only”, etc.

When such platform indicators are found, please pay attention only to the instructions for the platform you are running on.

Basics of Autolink

Autolink assists you in the automatic generation of TTCN test suites based on an SDL specification. In a simple approach, a TTCN test suite is generated in three steps:

1. One or more *system level Message Sequence Charts* have to be defined. System level MSCs contain the externally visible events of a path. These events describe the correct test sequences of a TTCN test case.
2. The MSCs defined in step 1 are used to generate internal test case representations.
3. All internal test case representations are saved as a test suite in a file in TTCN-MP format.

Constraints can be added, modified and deleted between any of the above steps.

Getting Ready to Use Autolink

In order to use Autolink, an SDL Explorer application has to be generated and started. In this exercise, you will use the `inres` system, which is found in `$telelogic/sdt/examples/inres`.

"Inres" is short for Initiator-Responder. The `inres` system is an example of a simplified communication protocol intended to give a secure transfer of information over an unsafe communication medium. It provides a one-way, connection-oriented communication service that uses sequence numbers and retransmission to ensure that messages sent from the initiator side are correctly received at the responder side.

> **Note:**
>
> In order to generate an explorer application that behaves as stated in the exercises, you should copy all files of the `inres` protocol from the release to your working directory.

You also have to create two directories where MSC test steps and test cases will be stored.

1. In your current working directory, create two subdirectories called `TC` (used for test cases) and `TS` (used for test steps).
2. Start TTCN Suite and open the `inres` system in the Organizer.
3. Generate an `inres` explorer and open it in the Explorer UI.

Exercise 1: Basic Concepts

What You Will Learn

- To set up value definitions
- To create an MSC test case definition
- To generate a test case
- To work with constraints

Preparations

First, you should define the location of the test cases and test steps directories which you have created prior to opening the SDL Explorer:

1. Select *Autolink: Test Cases Directory* in the *Options2* menu.

   The *Directory name* dialog is displayed.

2. Double-click the `TC` directory.

3. Click the *OK* button.

4. Select *Autolink: Test Steps Directory* in the *Options2* menu.

   Once again, the *Directory name* dialog is displayed.

5. Choose the `TS` directory by first clicking the *Current* button and then double-clicking the `TS` directory.

6. Click the *OK* button.

> **Note:**
> The directory settings can be saved when you leave the explorer.

You should also set up proper test value definitions for the `inres` system:

1. Open the *TEST VALUES* module by clicking on the box beside.

   A number of new command buttons appear.

2. Click the *Clear Value* button in the *TEST VALUES* module to clear all integer test values.

   A *Select* dialog is displayed.

3. Choose `integer`.
4. Click the *OK* button.

   A *Prompt* dialog is displayed.
5. Click the *OK* button, without typing anything in the text field.
6. Click the *Def Value* button in the *TEST VALUES* group to define a new test value for integers.

   Another *Select* dialog is displayed.
7. Choose `integer`.
8. Click the *OK* button.

   A *Prompt* dialog is displayed.
9. Type `55` as the new test value in the *Value* field.
10. Click the *OK* button.

Creating an MSC Test Case

The first step in the test case generation process is the choice of a path. A path describes a trace through the SDL system. This trace consists of internal and external events. External events are signals sent to or from the environment.

Autolink uses system level MSCs to abstract from a path by only considering the external events. System level MSCs consist of exactly one instance axis for the SDL system and one or more instance axes for the system environment. Figure 71 shows the MSC which you are going to create in this exercise.

1. Click the *Navigator* button in the *EXPLORE* module to start the Navigator.

   The *Navigator* window is displayed.

2. Navigate through the system by double-clicking in turn on the following nodes:

   1, 1, 1, 1, 1, 1, 1, 1 (i.e. 8 times transition #1),
   3,
   1, 1, 1, 1, 1, 1, 1 (i.e. 7 times transition #1),
   10,
   1, 1, 1 (i.e. 3 times transition #1),
   2,
   1, 1, 1 (i.e. 3 times transition #1)

3. Select *MSC*: *Save Test Case* in the *Autolink1* menu to save the current path as a system level MSC.

   A *Prompt* dialog is displayed.

4. Type **`Example1`** in the *Test case name* field.

   The test case name is identical to the name of the generated MSC.

5. Click the *OK* button.

   Autolink saves a file called `Example1.mpr` in the MSC test cases directory.

You have now created your first MSC test case (see Figure 71). You may start the MSC Editor and take a look at it.

Note:

You can create as many MSC test cases as you like, provided that they share the same root node.

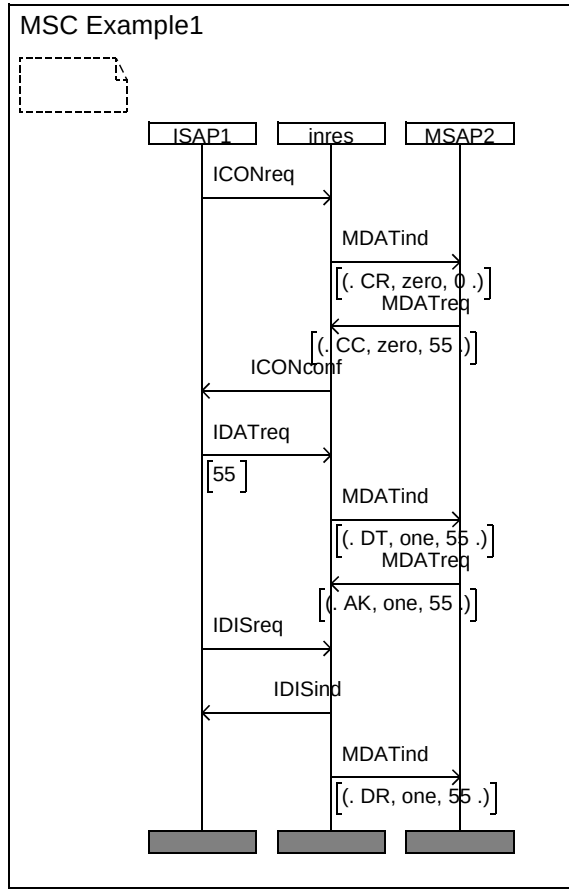


Figure 71: MSC Example1

It is possible to get a list of all MSC test cases (and test steps) which are currently defined:

- Select *MSC: List* in the *Autolink1* menu.

  A message similar to the one below should be given in the text area.

```
MSC test cases:
Example1.mpr

No MSC test steps found in directory
'/home/tutorial/inres/TS/'.
```

Generating the Test Case

In the next step, an internal test case representation has to be created which is based on the MSC test case defined above.

1. Select *Test Case: Generate* in the *Autolink1* menu.

   A *Select* dialog is displayed.

2. Select `Example1.mpr`.

3. Click the *OK* button.

   Autolink performs a bit state exploration, which produces the following output in the text area of the Explorer UI:

```
Autolink state space options are set.

Define-Scheduling All
Define-Transition Symbol-Sequence
Define-Symbol-Time Zero
Define-Priorities 2 1 3 2 2
Define-Channel-Queue ISAP1 On
Define-Channel-Queue MSAP2 On
Define-Max-Input-Port-Length 10

Current state is reset to root.

MSC 'Example1' loaded.

** Test case generation statistics **
No of reports: 2.
  Deadlock         : 1 report
  MSCVerification  : 1 report
Generated states: 2089.
Truncated paths: 0.
Unique system states: 790.
Size of hash table: 8000000 (1000000 bytes)
No of bits set in hash table: 1411
```

```
Collision risk: 0 %
Max depth: 49
Current depth: -1
Min state size: 388
Max state size: 524
Exploration started at: Sat Jan 29 13:39:14 2000
Exploration ended at:   Sat Jan 29 13:39:15 2000
Exploration time: 000 h, 00 m, 01 s

Constraints are checked for naming conflicts...

Constraint 'cExample1' is renamed to
'cExample1_001'.
Constraint 'cExample1' is renamed to
'cExample1_002'.
Constraint 'cExample1' is renamed to and merged with
'cExample1_002'.
Constraint 'cExample1' is renamed to and merged with
'cExample1_001'.
Constraint 'cExample1' is renamed to
'cExample1_003'.
Constraint 'cExample1' is renamed to
'cExample1_004'.
Constraint 'cExample1' is renamed to
'cExample1_005'.
Constraint 'cExample1' is renamed to
'cExample1_006'.
Constraint 'cExample1' is renamed to
'cExample1_007'.
Constraint 'cExample1' is renamed to
'cExample1_008'.
Constraint 'cExample1' is renamed to
'cExample1_009'.
Constraint 'cExample1' is renamed to
'cExample1_010'.

Constraints with identical signal definitions are
merged automatically...

... done.

State space options are restored.
```

You have now generated your first test case. This test case is kept in memory. You will save it in a TTCN test suite soon.

It is possible to get a list of all test cases which have been generated so far.

- Select *Test Case: List* in the *Autolink1* menu.

  The output will simply be:

```
Example1
```

You might also want to take a look at the internal test case representation before you create the TTCN test suite.

1. Select *Test Case: Print* in the *Autolink1* menu.

   A *Select* dialog is displayed.

2. Choose `Example1`.

3. Click the *OK* button.

   The resulting output is:

```
Example1

  0     ISAP1 ! ICONreq - cExample1_010
  1      MSAP2 ? MDATind - cExample1_009
  2       MSAP2 ! MDATreq - cExample1_008
  3        ISAP1 ? ICONconf - cExample1_007
  4         ISAP1 ! IDATreq - cExample1_006
  5          ISAP1 ! IDISreq - cExample1_005
  6           MSAP2 ? MDATind - cExample1_004
  7            MSAP2 ! MDATreq - cExample1_003
  8             ISAP1 ? IDISind - cExample1_001
  9  P           MSAP2 ? MDATind - cExample1_002
  8             MSAP2 ? MDATind - cExample1_002
  9  P           ISAP1 ? IDISind - cExample1_001
```

Modifying the Constraints

During test case generation, a number of constraints have been created. These constraints are also stored in memory and can be manipulated in several ways.

Listing the Constraints

First of all, it is possible to print the list of all constraints:

- Select *Constraint: List* in the *Autolink2* menu.

  This should be the output in the text area:

```
cExample1_001 :
  IDISind {  }
cExample1_002 :
  MDATind { mSDUType1 { id DR, num one, data 55 } }
cExample1_003 :
  MDATreq { mSDUType1 { id AK, num one, data 55 } }
cExample1_004 :
  MDATind { mSDUType1 { id DT, num one, data 55 } }
```

```
cExample1_005 :
  IDISreq {  }
cExample1_006 :
  IDATreq { iSDUType1 55 }
cExample1_007 :
  ICONconf {  }
cExample1_008 :
  MDATreq { mSDUType1 { id CC, num zero, data 55 } }
cExample1_009 :
  MDATind { mSDUType1 { id CR, num zero, data 0 } }
cExample1_010 :
  ICONreq {  }
```

As you can see, 10 constraints have been created during test generation.

Adding a Constraint

Next, you can add a new constraint:

1. Select *Constraint: Define* in the *Autolink2* menu.

   A *Prompt* dialog is displayed.

2. Type **`RequestCon`** in the *Constraint name* field.

3. Click the *OK* button.

   A *Select* dialog is displayed, listing all signals in the system.

4. Select `ICONreq`.

5. Click the *OK* button.

   In the text area of the Explorer user interface, the following message appears:

   ```
   Constraints are checked for naming conflicts...
   ```

   Of course, there is no naming conflict, since there does not exist another constraint with the same name.

Renaming a Constraint

You might want to assign a more reasonable name to constraint `cExample1_007` containing the signal `ICONconf`:

1. Select *Constraint: Rename* in the *Autolink2* menu.

   A *Select* dialog is displayed, listing all constraints currently defined.

2. Choose `cExample1_007`.

3. Click the *OK* button.

   A *Prompt* dialog is displayed.

4. Enter **`ConfirmCon`** in the *New constraint name* field.

5. Click the *OK* button.

Parameterize a Constraint

Constraint `cExample1_006` contains a signal parameter (value `55`). You can define this signal parameter to be a parameter of the constraint.

1. Select *Constraint: Parameterize* in the *Autolink2* menu.

   A *Select* dialog is displayed.

2. Select `cExample1_006`.

3. Click the *OK* button.

   A *Select* dialog is displayed.

4. Select `1` for the first signal parameter.

5. Click the *OK* button.

   A *Prompt* dialog is displayed.

6. Enter **`Data`** in the *Formal parameter* field.

7. Click the *OK* button.

   The *Select* dialog is displayed again.

8. This time, select `0` in order to finish parameterization.

9. Click the *OK* button.

Merging Constraints

The constraints `cExample1_002` and `cExample1_004` only differ in the first signal parameter (which is a struct). Therefore, you might want to merge them.

1. Select *Constraint: Merge* in the *Autolink2* menu.

   A *Select* dialog is displayed, listing all currently defined constraints.

2. Choose `cExample1_002` as the first constraint.

3. Click the *OK* button.

   Another *Select* dialog is displayed.

4. Choose `cExample1_004` as the second constraint.

   A *Prompt* dialog is displayed.

5. Enter **`ProtDataUnit`** in the *Formal parameter name* field.

6. Click the *OK* button.

Listing the Constraints – A Second Time

To see the effect of all your operations, you can list the constraints again:

1. Select *Constraint: List* in the *Autolink2* menu.

   This time, the following output should be displayed in the text area:

```
ConfirmCon :
  ICONconf {  }
RequestCon :
  ICONreq {  }
cExample1_001 :
  IDISind {  }
cExample1_003 :
  MDATreq { mSDUType1 { id AK, num one, data 55 } }
cExample1_004(ProtDataUnit) :
  MDATind { mSDUType1 ProtDataUnit }
cExample1_005 :
  IDISreq {  }
cExample1_006(Data) :
  IDATreq { iSDUType1 Data }
cExample1_008 :
  MDATreq { mSDUType1 { id CC, num zero, data 55 } }
cExample1_009 :
  MDATind { mSDUType1 { id CR, num zero, data 0 } }
cExample1_010 :
  ICONreq {  }
```

In the test case, all references to the constraints have been updated appropriately:

2. Select *Test Case: Print* in the *Autolink1* menu.

   A *Select* dialog is displayed.

3. Select `Example1`.

4. Click the *OK* button.

   The output is:

```
Example1

  0     ISAP1 ! ICONreq - cExample1_010
  1      MSAP2 ? MDATind - cExample1_009
  2       MSAP2 ! MDATreq - cExample1_008
  3        ISAP1 ? ICONconf - ConfirmCon
  4         ISAP1 ! IDATreq - cExample1_006(55)
  5          ISAP1 ! IDISreq - cExample1_005
  6           MSAP2 ? MDATind - cExample1_004
                  ({ id DT, num one, data 55 })
  7            MSAP2 ! MDATreq - cExample1_003
  8             ISAP1 ? IDISind - cExample1_001
  9  P           MSAP2 ? MDATind - cExample1_004
                    ({ id DR, num one, data 55 })
  8             MSAP2 ? MDATind - cExample1_004
                    ({ id DR, num one, data 55 })
  9  P           ISAP1 ? IDISind - cExample1_001
```

Saving the Constraints

Since all constraints are deleted if they are no longer referred to by any generated test case or if you leave the Explorer, they can be saved in a file:

1. Select *Constraint: Save* in the *Autolink2* menu.

   A *Select* dialog is displayed.

2. Click the *OK* button without entering any text before.

   By this you indicate that you want to store all constraints.

   The *File name* dialog is displayed.

3. Type **`Example1.con`** in the *File* field.

4. Click the *OK* button.

The constraint definitions are saved. They can be restored by the *Constraint: Load* command in the *Autolink2* menu but you should not do that at the moment.

Removing a Constraint

A constraint can be removed:

1. Select *Constraint: Clear* in the *Autolink2* menu.

   A *Select* dialog is displayed.

2. Select `cExample1_003`.

3. Click the *OK* button.

   Since this constraint is currently used in a test case, an error message is displayed in the text area:

   ```
   Error clearing constraint 'cExample1_003'.
   The constraint is currently used in a test case.
   ```

4. Once again, select *Constraint: Clear* in the *Autolink2* menu.

   The same *Select* dialog as above is displayed.

5. Select `RequestCon`.

6. Click the *OK* button.

   This time the specified constraint is removed, since it is not referred to in any test case.

Saving the TTCN Test Suite

In a final step, all generated test cases can be saved in a TTCN test suite. In this tutorial there is only one test case, but it is also possible to put several test cases into one test suite.

1. Select *Test Suite: Save* in the *Autolink1* menu.

   A *Prompt* dialog is displayed.

2. Type **`inres`** in the *Test suite name* field.

3. Click the *OK* button.

   A *File name* dialog is displayed.

4. Type **inres.mp** in the *File* field.
5. Click the *OK* button.

   Autolink saves the test suite in file `inres.mp`.

   During saving, Autolink performs a few checks to ensure the consistent use of various test suite elements. You can ignore the message about test steps, since your test case does not have any test steps at all.

You can exit the Explorer now:

1. Select *Exit* in the *File* menu.

   A *Select* dialog is displayed.

2. Select `Yes`.
3. Click the *OK* button.

   The current options are saved. You will need them in the next exercise.

Viewing the TTCN Test Suite

Up to now, you have created a TTCN test suite. This test suite is stored in MP format (*m*achine *p*rocessable) which is somewhat hard to read. Therefore you should add it to the Organizer and open it in the TTCN Suite.

1. In the Organizer, select the *TTCN Test Specification* chapter.
2. Select *Add Existing* in the *Edit* menu.

   A corresponding dialog is displayed.

3. Change the filter from "`*.txt`" to "`*.mp`" and click the *Filter* button.

   Provided that you are in the right directory, the file `inres.mp` is shown in the *Files* box.

4. Select the file `inres.mp` in the *Files* section.
5. Click the *OK* button.

The file is added to the Organizer and the TTCN Suite is opened with the test suite.

Completing the Test Suite

The TTCN test suite you have created so far contains information in three parts: the declarations part, the constraints part and the dynamic part.

In Windows, the test suite overview part will be generated automatically for example when you open one of the overview tables or before you print the test suite. After that, it will be kept updated.

On UNIX, you have to generate and update the overview explicitly:

1. Select *Generate Overview* in the *Tools* menu in the Browser.

   A dialog windows appears.

2. Click the *Generate* button.

   The test case index is generated.

Exercise 2: Advanced Concepts

What You Will Learn

- To create a structured MSC test case
- To define a simple Autolink configuration
- To generate a structured TTCN test case by a direct translation
- To merge new test cases with an existing test suite

Preparations

If you have closed the SDL Suite or the Explorer UI temporarily, perform the following steps:

1. Open the `inres` system in the Organizer.
2. Start the Explorer UI.
3. Open the `inres` explorer again.

If you have **not** quit the `inres` explorer, bring the application to its initial state now:

- Select *Reset* in the *Options1* menu.

Creating a Structured MSC Test Case

For a better understanding, a test case can be structured into different test steps. Typically, a test case consists of three parts:

- The *preamble* comprises a sequence of test events that drives the system from the stable start state to the initial state from which the test body starts.
- The *test body* comprises a sequence of test events that achieve the test purpose.
- The *postamble* comprises a sequence of test events that drives the system from the state reached by the test body to a stable end state.

In this section, you will create a test case with a preamble, a test body and a postamble. This test case will be identical to the one developed in "Exercise 1: Basic Concepts" on page 156 except for its structural information.

Creating the Preamble

You can define structured MSC test cases in a similar way as you did in "Exercise 1: Basic Concepts" on page 156.

1. Click the *Navigator* button in the *EXPLORE* module to start the Navigator.

   The *Navigator* window is displayed.

2. Navigate through the system by double-clicking in turn on the following nodes:

   1, 1, 1, 1, 1, 1, 1, 1 (i.e. 8 times transition #1),
   3,
   1, 1, 1 (i.e. 3 times transition #1)

   Do not close the Navigator yet.

3. Select *MSC: Save Test Step* in the *Autolink1* menu to save the current path as an MSC test step.

   A prompt dialog is displayed.

4. Type **`Preamble`** in the *Test step name* field.

5. Click the *OK* button.

   Autolink saves a file called `Preamble.mpr` in the MSC test steps directory.

The preamble which you have just created is shown in Figure 73 on page 173.

Creating the Test Body

You should also create the body of the MSC test case:

1. Type **`define-root current`** in the input line to set the current root.

   The following message appears in the text area:

   ```
   Root of behaviour tree set to current system state
   ```

2. Double-click the following transitions in the *Navigator* window:

   1, 1, 1, 1 (i.e. 4 times transition #1),
   10,
   1, 1, 1 (i.e. 3 times transition #1).

   Again, do not close the Navigator.

3. Select *MSC: Save Test Case* in the *Autolink1* menu to save the path as an MSC test case.

   Make sure that you save the path as a test case, not as test step!

   A *Prompt* dialog is displayed.

4. Type **`Example2`** in the *Test case name* field.

5. Click the *OK* button.

   Autolink saves a file called `Example2.mpr` in the MSC test cases directory.

Now you have defined the body of the test case. See Figure 72.

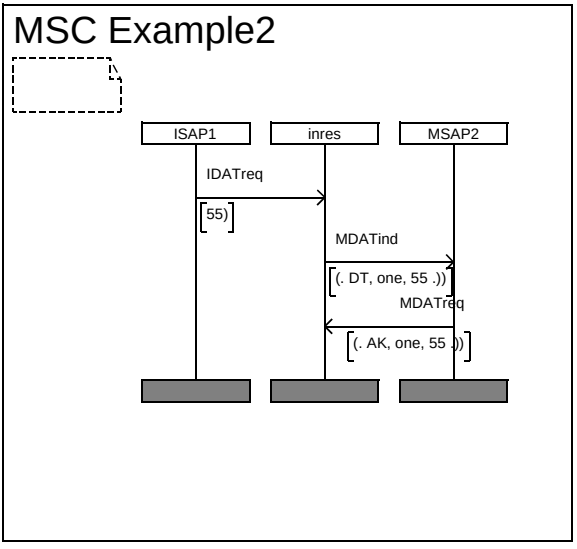


Figure 72: The MSC Example2 – test body

Creating the Postamble

Finally, you create the postamble of the MSC.

1. Again, type **`define-root current`** in the input line to set the root.

   The following message appears in the text area:

   ```
   Root of behaviour tree set to current system state
   ```

2. Navigate through the system by double-clicking in turn on the following nodes:

   2,
   1, 1, 1.

3. Select *MSC: Save Test Step* in the *Autolink1* menu.

   A *Prompt* dialog is displayed.

4. Type **`Postamble`** in the *Test step name* field.

5. Click the *OK* button.

   Autolink saves a file called `Postamble.mpr` in the MSC test steps directory.

You have now created the postamble of the test case, see Figure 73.

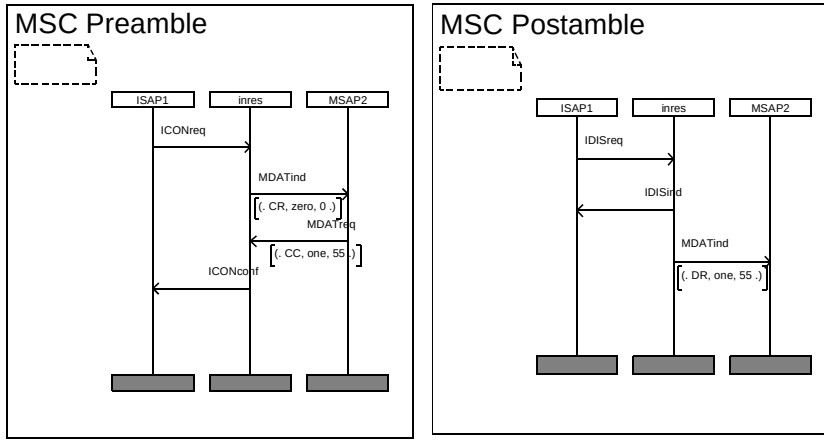


Figure 73: Preamble and Postamble MSCs

Inserting MSC References

Before you can use MSC *Example2* to generate a TTCN test case, you have to insert two global MSC references for the preamble and the postamble. To do this:

1. Select *Toggle MSC Trace* in the *Commands* menu.

   The MSC Editor will start, showing a complete Explorer Trace.

2. Open MSC `Example2.mpr` stored in directory `TC`.

3. Make the MSC look like Figure 74, that is:
   - Insert two global MSC references, one above the first message and one after the last message.
   - Type **`Preamble`** and **`Postamble`** (**without** the file extension `.mpr`) as reference names.

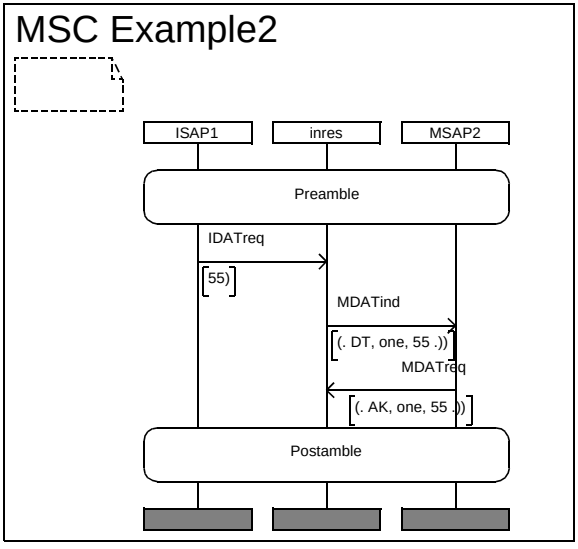


Figure 74: Example2

4. Save the MSC diagram as `Example2.msc`.

   You have now created a complete structured MSC test case with a pre- and a postamble and you are ready to generate a TTCN test case from it.

In the previous steps, you have changed the root of the behavior tree to the beginning of the postamble. Even though it is not necessary for a direct translation, you should better reset it before continuing.

1. Switch back to the *Explorer UI* window.
2. Type **`Define-Root Original`** at the command line.

   The MSC Editor is started, but you do not have to bother about that. Select *Toggle MSC Trace* in the *Commands* menu to quit it.

You can list all MSC test cases and test steps currently stored on disk:

3. Select *MSC: List* in the *Autolink1* menu.

   The following output should appear in the text area:

```
MSC test cases:
Example1.mpr Example2.mpr Example2.msc

MSC test steps:
Postamble.mpr Preamble.mpr
```

It is also possible to remove test cases and test steps from disk. Since you do not need the temporary MSC *Example2.mpr* any more, you can delete it:

1. Select *MSC*: *Clear Test Case* in the *Autolink1* menu.

   A *Select* dialog is displayed.
2. Select `Example2.mpr`.
3. Click the *OK* button.

   Another *Select* dialog is displayed, asking you whether you really want to delete this MSC.
4. Choose `Yes`.
5. Click the *OK* button.

Defining an Autolink Configuration

Autolink allows to define a configuration which guides the naming and parameterization of constraints, the introduction of test suite parameters and constants and the grouping of test cases and test steps into a hierarchy of test groups. An Autolink configuration consists of a set of rules which have to be written by the user.

In this tutorial you will not learn how to write configuration rules. Instead you will use a pre-defined configuration that covers *some* aspects of an Autolink configuration. For a detailed descriptions of configuration rules and corresponding examples see "Translation Rules" on page 1459 in chapter 35, *TTCN Test Suite Generation* and "Test Suite Structure Rules" on page 1465 in chapter 35, *TTCN Test Suite Generation, in the User´s Manual*.

In a first step you have to create a new command file which contains the Autolink configuration:

1. Open a text editor.

   The choice of text editor depends on the operating system of your computer.

   Create a new file called `config.com`.

2. Type in the following text:

```
# -------------------------------
#   Autolink configuration file
#     for the inres protocol
# -------------------------------

Define-Autolink-Configuration

TRANSLATE "ICONreq"
  CONSTRAINT NAME   "Connection_Request"
END

TRANSLATE "ICONconf"
  CONSTRAINT NAME   "Connection_Confirmation"
END

TRANSLATE "IDATreq"
  CONSTRAINT NAME   "Data_Request"
             PARS   $1="Data"
  TESTSUITE  PARS   $1=”DataValue”
END
```

```
TRANSLATE "IDISreq"
  CONSTRAINT NAME   "Disconnection_Request"
END

TRANSLATE "IDISind"
  CONSTRAINT NAME   "Disconnection_Indication"
END

TRANSLATE "MDATind" | "MDATreq"
  IF $1 == "{ id CC, *, * }" THEN
    CONSTRAINT NAME "Medium_Connection_Confirmation"
  END
  IF $1 == "{ id AK, *, * }" THEN
    CONSTRAINT NAME   "Medium_Acknowledge"
  END
  CONSTRAINT NAME   "c_" + $0
END

End
```

3. Save the file in your current working directory.

Next, you have to load this configuration into the Explorer:

1. Select *Configuration: Load* in the *Autolink2* menu.

   A *File name* dialog is displayed.

2. Select `config.com` in the *Files* section.

3. Click the *OK* button.

If an error message is displayed, you have probably made a spelling mistake in the command file. In this case, correct your file and try to re-load the configuration.

Translating the MSC into TTCN

In "Generating the Test Case" on page 160, you have learned how to generate an internal test case representation based on an MSC. For that purpose, a state space exploration has been started.

However, in some cases it is not possible to simulate an MSC, e.g. if the internal processes of the SDL system are not fully specified. In these cases, you have to translate an MSC test case directly into an internal test case representation (without performing a state space exploration).

Even though it should be possible to simulate the MSC `Example2.msc`, you will apply a direct translation in this exercise:

1. Select *Test Case: Translate* in the *Autolink1* menu.

   A *Select* dialog is displayed.

2. Select `Example2.msc`.

3. Click the *OK* button.

   Autolink analyses the preamble, the test body and the postamble and generates a complete, structured test case. The following output should appear in the text area:

```
Current state is reset to root.

MSC 'Example2' loaded.

Constraints are checked for naming conflicts...

Constraint 'c_MDATind' is renamed to
'c_MDATind_001'.
Constraint 'c_MDATind' is renamed to
'c_MDATind_002'.
Constraint 'c_MDATind' is renamed to
'c_MDATind_003'.

Constraints with identical signal definitions are
merged automatically...

... done.
```

Take a look at the internal test case representation to see the difference with respect to test case `Example1`:

1. Select *Test Case: Print* in the *Autolink1* menu.

   A *Select* dialog is displayed.

2. Choose `Example2`.

3. Click the *OK* button.

4. The output is:

```
Example2

  0      IN Preamble
  1       ISAP1 ! ICONreq - Connection_Request
  2        MSAP2 ? MDATind - c_MDATind_003
  3         MSAP2 ! MDATreq -
               Medium_Connection_Confirmation
  4          ISAP1 ? ICONconf -
                Connection_Confirmation
  5           OUT Preamble
  6            ISAP1 ! IDATreq -
                  Data_Request(DataValue)
  7  (P)        MSAP2 ? MDATind - c_MDATind_002
  8              MSAP2 ! MDATreq - Medium_Acknowledge
  9               IN Postamble
 10                ISAP1 ! IDISreq -
                      Disconnection_Request
 11                ISAP1 ? IDISind -
                      Disconnection_Indication
 12   P             MSAP2 ? MDATind - c_MDATind_001
 13                  OUT Postamble
 11                MSAP2 ? MDATind - c_MDATind_001
 12   P             ISAP1 ? IDISind -
                      Disconnection_Indication
 13                  OUT Postamble
```

Also take a look at the constraints to see what the translation rules in your configuration file have affected:

5. Select *Constraint: List* in the *Autolink2* menu.

The following output will appear in the text area:

```
Connection_Confirmation :
  ICONconf {  }
Connection_Request :
  ICONreq {  }
Data_Request(Data) :
  IDATreq { iSDUType1 Data }
Disconnection_Indication :
  IDISind {  }
Disconnection_Request :
  IDISreq {  }
Medium_Acknowledge :
  MDATreq { mSDUType1 { id AK, num one, data 55 } }
Medium_Connection_Confirmation :
  MDATreq { mSDUType1 { id CC, num zero, data 55 } }
c_MDATind_001 :
  MDATind { mSDUType1 { id DR, num one, data 55 } }
c_MDATind_002 :
  MDATind { mSDUType1 { id DT, num one, data 55 } }
c_MDATind_003 :
```

```
MDATind { mSDUType1 { id CR, num zero, data 0 } }
```

Saving the TTCN Test Suite

When a TTCN test suite is created, test steps can be stored in different formats. By default, test steps are put into the test step library. However, you might want to save them locally, i.e. attached to their test cases:

1. Select *Autolink: Test Steps Format* in the *Options2* menu.

   A *Select* dialog is displayed.

2. Select `Local`.

3. Click the *OK* button.

Now, you are ready to save the test suite:

1. Select *Test Suite: Save* in the *Autolink1* menu.

   A *Prompt* dialog is displayed.

2. Type **`inres2_local`** in the *Test suite name* field.

3. Click the *OK* button.

   A *File name* dialog is displayed.

4. Type **`inres2_local.mp`** in the *File* field.

5. Click the *OK* button.

   Autolink saves the test suite in file `inres2_local.mp`.

You may want to save the test suite in different formats. To do this:

1. Select *Autolink: Test Steps Format* in the *Options2* menu.

   A *Select* dialog is displayed.

2. Select `Global` or `Inline`.

3. Click the *OK* button.

4. Save the test suite as above. You can save the current test suite as often as you like.

Merging With the Old Test Suite

It is possible to merge the new TTCN test case (with its constraints) with an existing test suite.

1. Make sure that the `inres` test suite you created in the first part of this tutorial is opened in the TTCN Suite. This test suite contains test case `Example1`.
2. **On UNIX**, select *Autolink Merge* from the *SDT Link* menu in the Browser.

   In Windows, select *Autolink Merge* from the *File* menu.
3. In the dialog that is opened, select `inres2_local.mp`.
4. Click *Merge* **(UNIX)** or *OK* **(Windows)**.

   The test case and the constraints from the `inres2_local.mp` file will now be merged into the old test suite.

There are more possibilities to merge test cases: Autolink also provides commands to combine separately generated test cases within the Explorer. For a detailed description see "Computing Test Cases" on page 1450 in chapter 35, *TTCN Test Suite Generation, in the User´s Manual*.

Exercise 3: Test Generation with Tree Walk

What You Will Learn

- To create a large number of MSC test cases automatically
- To save and load generated test cases in Autolink
- To distribute test case generation

Preparations

If you have closed the SDL Suite or the Explorer UI temporarily, perform the following steps:

1. Open the `inres` system in the Organizer.
2. Start the Explorer UI.
3. Open the `inres` explorer again.

If you have **not** quit the `inres` explorer, bring the application to its initial state now:

4. Select *Reset* in the *Options1* menu.

Creating MSC Test Cases Automatically

In the previous exercises, you had to specify your MSC test cases manually. Creating test cases by hand gives you full control over what is tested.

However, you might not be interested in defining tests for particular aspects of your SDL system. Instead, you only wish to create a large number of test cases randomly which cover most symbols of the SDL system.

In this case, you can use the Tree Walk algorithm provided by Autolink to generate test cases automatically.

In the first step, a number of *TreeWalk* reports have to be computed:

1. Click the *Tree Walk* button in the *EXPLORE* module.

   A *Prompt* dialog is displayed.

2. Enter the value **10**.

   This is the maximum number of minutes given to Autolink for computing reports. In fact, the state space exploration is much faster for simple protocols such as Inres.

   Another *Prompt* dialog is displayed.

3. Enter the value **100**.

   This is the percentage of symbol coverage that you want to reach when executing your test cases.

   A message similar to the one below will appear in the text area:

```
Tree Walk will stop after 10 minutes or after
reaching 100% symbol coverage.
Reports and symbol coverage table cleared.
States: 2109. Reports: 3. Tree walk reports: 0
(+31). Coverage: 100.00%. Time: 000:00:01

Target symbol coverage is reached.
Tree Walk is stopped.
Test report #1 of length 7 added.
Test report #2 of length 8 added.
Test report #3 of length 12 added.
Test report #4 of length 16 added.
Test report #5 of length 16 added.
Test report #6 of length 20 added.
Test report #7 of length 22 added.
Test report #8 of length 30 added.
Test report #9 of length 32 added.
```

The output above states that nine reports have been generated. When traversing their corresponding paths, a total symbol coverage of 100 percent is reached. The Tree Walk command stops immediately when it reaches the targeted symbol coverage.

Since Autolink requires MSC test cases as input, you have to convert the TreeWalk reports into MSCs:

1. Select *MSC: Save Reports* in the *Autolink1* menu.

   A *Select* dialog is displayed.

2. Select `TreeWalk` in order to save all TreeWalk reports.
3. Click the *OK* button.

   A Prompt dialog is displayed.
4. Enter **Tutorial** as test case name prefix.

   Autolink saves all *TreeWalk* reports as system level MSCs in distinct files in the test cases directory:

```
MSC test case is saved in file
'/home/tutorial/inres/TC/Tutorial_Tree_00001.mpr'.
MSC test case is saved in file
'/home/tutorial/inres/TC/Tutorial_Tree_00002.mpr'.
MSC test case is saved in file
...
MSC test case is saved in file
'/home/tutorial/inres/TC/Tutorial_Tree_00009.mpr'.
```

Generating the Test Cases

In the previous exercises you have learned how to generate internal test case representations from MSCs. If you want more than one test case, you can either compute all test cases in the test cases directory at once – by a single command – or compute one test case after another.

For large SDL systems, test generation may take some time. In this case, it is advantageous not to compute all test cases in a single explorer session. Instead, each test case (or a small number of test cases) should be computed separately, preferably in parallel on different machines.

To enable distributed test generation, Autolink allows to store internal test case representations on disk. After all test cases have been computed, their internal representations can be reloaded from disk and saved in a single test suite. While loading the test cases, Autolink checks them for consistency and merges identical constraints.

Generating and Saving a Single Test Case

1. Select *Reset* in the *Options1* menu.

   All options are reset and all generated test cases and reports are cleared.
2. Select *Test case: Generate* in the *Autolink1* menu.
3. A *Select* dialog is displayed.

4. Select **Tutorial_Tree_00007.mpr**.
5. Click the *OK* button.
6. Select *Test case: Save* in the *Autolink1* menu.

   A *Select* dialog is displayed.
7. Select `Tutorial_Tree_00007`.
8. Click the *OK* button.

   A *File name* dialog is displayed.
9. Select the `TC` directory by double-clicking.
10. Type **Tutorial_Tree_00007.gen** in the *File* field.
11. Click the *OK* button.

    The generated test case is stored on disk.

Repeat steps 1 to 11 for other TreeWalk test cases.

Saving the Test Suite

Once you have created all test cases and saved them in files, you can combine them and create a test suite.

1. Select *Reset* in the *Options1* menu.
2. Select *Test case: Load* in the *Autolink1* menu.

   A *File name* dialog is displayed.
3. Select file `Tutorial_Tree_00007.gen` in the *Files* section.
4. Click the *OK* button.

   The generated test case is loaded and identical constraints are merged (if there are identical constraints).
5. Repeat steps 2 to 4 for all other test cases.
6. Select *Test suite: Save* in the *Autolink1* menu.

   A *Prompt* dialog is displayed.
7. Type **`TreeWalk`** in the *Test suite name* field.

8. Click the *OK* button.

   A *File name* dialog is displayed.

9. Type **`TreeWalk.mp`** the *File* field.

10. Click the *OK* button.

    Autolink saves the test suite file `TreeWalk.mp`.

    Now you may exit the Explorer:

11. Select *Exit* in the *File* menu.

In order to take a look at the TTCN test suite, add the test suite to the Organizer just like you did in the first and second example above.

So Far...

You should have learned how to create structured system level MSCs, and how to generate TTCN test cases from those MSCs. Especially you have become acquainted with two ways of producing TTCN test cases: Test generation by state space search and test generation by a direct translation. While doing this, you have discovered many of the commands in the *Autolink* menus of the Explorer. In addition, you should also know how to view the test suite in the TTCN Suite, how to generate the test suite overview and how to merge new test cases and constraints into an existing test suite in the TTCN Suite.