Chapter

28

The TTCN to C Compiler (on UNIX)

This chapter describes what the TTCN to C compiler is used for, how to run it and the structure of the generated code.

When the TTCN to C compiler has translated TTCN into C code, the code must be adapted with the system that is to be tested. The adaption process is described in chapter 37, *Adaptation of Generated Code*.

Note: ASN.1 support

The TTCN to C compiler supports only a limited subset of ASN.1. See "Support for External ASN.1 in the TTCN Suite" on page 729 in chapter 14, *The ASN.1 Utilities, in the User's Manual* for further details on the restrictions that apply.

Note: UNIX version

This is the UNIX version of the chapter. The Windows version is chapter 33, *The TTCN to C Compiler (in Windows)*.

Introduction to the TTCN to C Compiler

When developing new systems or implementations of any kind, the developing process is divided into well defined phases. Most commonly, the first phases involve some kind of specification and abstract design of the new system. After a while, the implementation phase is entered and finally, when all parts are joined together, the test phase is activated.

In any case when a system is tested, we want to make sure that its behavior conforms to a set of well defined rules. TTCN was developed for the specification of test sequences. Unfortunately, as very few systems interpret or compile pure TTCN, we need to translate the TTCN notation into a language which can be compiled and executed. In the case of the TTCN suite, the TTCN to C compiler translates TTCN to ANSI-C.

Even after the TTCN code has been translated, there are a couple of things that need to be taken care of. In this case, we must *adapt* the generated code with the system it intends to test. This chapter describes how the TTCN to C compiler is used. The adaption process is described in chapter 37, *Adaptation of Generated Code*.

Getting Started

Unfortunately a test sequence description expressed in TTCN can not easily be executed as it is. This is because the test notation is not executable and only few test environments interpret pure TTCN. A different approach to create an executable test suite (ETS), is to translate the formal test description into a language which can be compiled into an executable format.

The TTCN to C compiler translates TTCN into ANSI-C which can be compiled by an ANSI-C compiler. Figure 213 depicts the first step in the process of creating an ETS using the TTCN to C compiler.

The generated code, called the TTCN runtime behavior, is only one of the two major modules of an ETS.

The second module which is needed includes test support functions which are dependent on the protocol used, the host machine, test equipment, etc. For this reason, it is up to the user to write this second module and in such way adapt the TTCN runtime behavior to the system he/she wants to test.

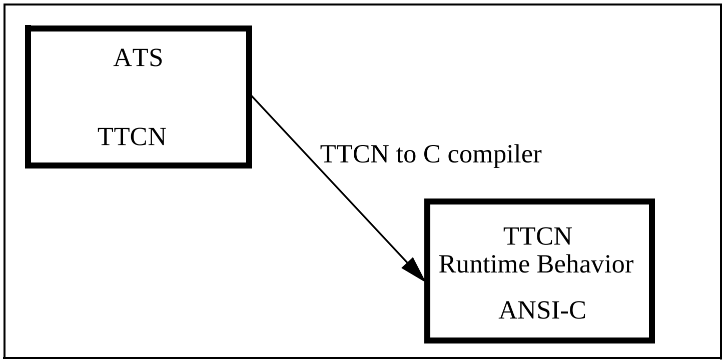


Figure 213: Translating TTCN to ANSI-C

The adaption process is described in chapter 37, *Adaptation of Generated Code*. Figure 214 displays the anatomy of the final result.

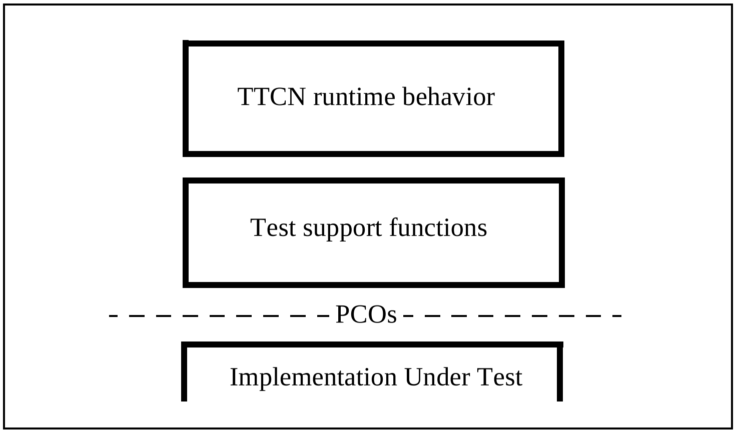


Figure 214: The anatomy of an ETS

Running the TTCN to C Compiler

The TTCN to C compiler can be activated either from the TTCN Browser, both as a quick button and a menu command, or from the Organizer when a TTCN test suite is selected.

Note:

Observe that the TTCN to C compiler is unable to function correctly if the test suite is not fully verified correct. This verification is accomplished by running the Analyzer tool on all the parts of the test suite.

Tools > Make

When you select *Make*, the TTCN Make dialog is opened. The dialog contains two important toggle buttons, *Analyze & Generate* and *Compile & Link*.

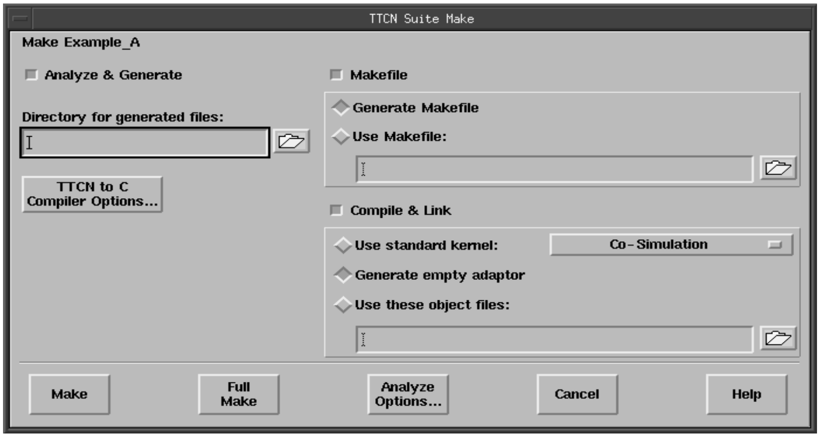


Figure 215: Dialog for starting the TTCN to C compiler

Analyze & Generate

If *Analyze & Generate* is set, the Analyzer will first be called to analyze the test suite before any code is generated. If errors are found, these will be reported and no code will be written to file. The generation phase performed after the analysis phase will only generate code for the parts that need to be re-generated.

Directory for generated files

If code is generated, that code will reside in the directory specified in this field.

Makefile

If *Compile & Link* is set, the TTCN suite will also take care of the making and linking of the generated code. At this point you have the possibility to specify how you want the code to be built. Either you can specify that the default generated makefile is to be used, or your own special makefile. This is done by setting the radio buttons in the field below the *Makefile* toggle button.

Compile & Link

The compiler needs to know what files to link into the executable. Three different possibilities are at hand:

- You have not yet written an adaptor, but you want an empty template to allow compilation to go through successfully.
- You have already written an adaptor which you want to link into the executable.
- You want to link to a simulator kernel to allow you to simulate.

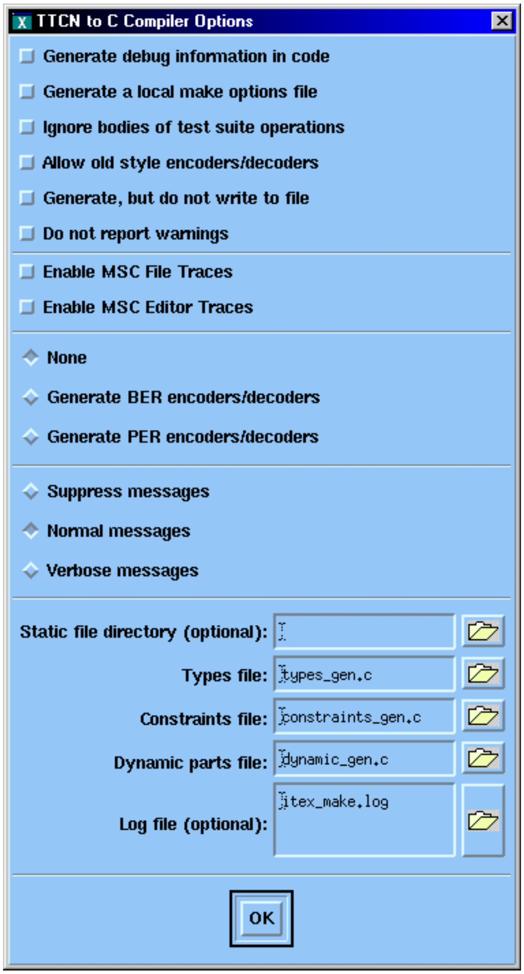
The TTCN to C compiler assumes the first case to be the default one. In this case the *Generate empty adaptor files* radio button is set. If you do not have adaptor files (`adaptor.c` and `adaptor.h`) in the target directory, these will be created, as empty templates, for you. If you already have generated and edited adaptor files these will of course not be overwritten.

If you already have your own compiled adaptor files elsewhere in the file system, these files, as object files, should be listed in the text field below the *Use the object files* dialog button. Observe that you should use absolute path names for the adaptor files to avoid problems for the makefile to find these files! An absolute path name is a path name that identifies a file uniquely from the root of the files system, for example `/home/myhome/myadaptor.o`.

Finally, if you want to generate code for simulation, the *Use standard kernel* radio button should be set with the option *Simulation*. This is the only available kernel at the moment so no other choice is valid here.

TTCN to C Compiler Options

By pressing the *TTCN to C Compiler Options* button in the Make dialog, the dialog depicted in Figure 216 will be displayed.


TTCN to C Compiler Options

- ☐ Generate debug information in code
- ☐ Generate a local make options file
- ☐ Ignore bodies of test suite operations
- ☐ Allow old style encoders/decoders
- ☐ Generate, but do not write to file
- ☐ Do not report warnings
- ☐ Enable MSC File Traces
- ☐ Enable MSC Editor Traces
- ◇ None
- ◇ Generate BER encoders/decoders
- ◇ Generate PER encoders/decoders
- ◇ Suppress messages
- ◇ Normal messages
- ◇ Verbose messages

Static file directory (optional):	
Types file:	types_gen.c
Constraints file:	constraints_gen.c
Dynamic parts file:	dynamic_gen.c
Log file (optional):	itex_make.log

Figure 216: Setting extra TTCN to C compiler options

- If the *Generate debug information in code* button is set, the compiler will generate extra code for tracing and debugging.
- If the *Generate a local make options file* button is set, the compiler will generate a file called `makeopts` in the target directory. This file is used to define settings used by the makefile. You have the flexibility to choose compiler and add extra information for the make process.
- If the *Ignore bodies of test suite operations* button is set, the compiler will ignore to generate code for test suite operations. This is useful if you already have a file with your own test suite operations that only need to be linked to the rest of the code. If the button is not set, the test suite operators will be generated and included in the file `tsop_gen.c`.
- If the *Allow old style encoders/decoders* button is set, the compiler will generate two extra files that will allow you to use an adaptor written for the old code generator to be used with the new one.
- If the *Generate, but do not write to file* button is on, the compiler will not generate code. This is useful for users that only wish to generate code when they know no generation errors will occur.
- If the *Do not report warnings* button is set, the compiler will not inform you about warnings.
- By selecting one of the radio buttons *None, Generate BER encoders/decoders* or *Generate PER encoders/decoders* the compiler will either omit generation of encoders/decoders or generate extra code for BER or PER encoding and decoding support. See chapter 59, *ASN.1 Encoding and De-coding in the SDL Suite, in the User's Manual*.
- By selecting one of the radio buttons *Suppress messages, Normal Messages* or *Verbose messages* you can set the level of verbosity when the compiler generates code. The default value is *Normal messages*.

You can also choose the names of the files you want to use. This is done in the fields in the lower part of the TTCN to C compiler options dialog.

- The optional *Static file directory* field should only be used by users that brutally want to change the behavior of the compiler by using their own static files. For “normal” users it is recommended that this field is not set.

- The *Types file* field determines the name of the file to which type definitions from your test suite are generated. The default value is `types_gen.c`.
- The *Constraints file* field determines the name of the file to which constraint definitions from your test suite are generated. The default value is `constraints_gen.c`.
- The *Dynamic part file* field determines the name of the file to which the dynamic part from your test suite is generated. The default value is `dynamic_gen.c`.
- The *Log file* field determines the name of the file to which messages about the generation are to be written. Default is standard out.

Make

If the *Make* button is pressed, the compiler will only generate code for the parts that really need to be re-generated.

Full Make

If the *Full Make* button is pressed, everything will be regenerated again.

When the generation phase is started, a new log window will be displayed where information about the compiler actions will be displayed.

Running the TTCN to C Compiler from the Command Line

Please see “Running the TTCN to C Compiler from the Command Line” on page 1291 in chapter 33, *The TTCN to C Compiler (in Windows)*.

What Is Generated?

In the case of the code generated from the compiler we also need a set of static functions which handle TTCN basics and other internal events. Even if these functions are vital for the successful compilation and execution of the generated code, the user should not have to worry about this part. These functions are gathered in a small set of static files which are compiled by the generated makefile and linked with the rest of the code.

The Code Files

The generated makefile is the file containing a definition of how the code should be compiled and linked.

The `adaptor.h` and `adaptor.c` files are the files that contain the adaptation code. If code is generated for the first time, these files will be generated by the compiler with empty function templates for the user to implement. On the other hand, if these files are present in the target directory the user does not have to worry about getting them overwritten.

The `*_gen.{c,h}` files contains the code generated for the TTCN test suite.

The `asn1ende.h` file contains the encode and decode functions for the ASN.1 Types. See chapter 59, *ASN.1 Encoding and De-coding in the SDL Suite, in the User's Manual*. (Only generated if ASN.1 encoding/decoding support has been selected)

The Adaptation

We are now ready to deal in greater detail with the adaptation phase which is the final phase to create an executable test suite. The adaptation process is described in "Adaptation of Generated Code" on page 1449 in chapter 37, *Adaptation of Generated Code*.

TTCN Test Logs in MSC Format

Please refer to "TTCN Test Logs in MSC Format" on page 1295 in chapter 33, *The TTCN to C Compiler (in Windows)*.