Chapter

33

The TTCN to C Compiler (in Windows)

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: Windows version

This is the Windows version of the chapter. The UNIX version is chapter 28, *The TTCN to C Compiler (on UNIX)*.

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 "Adaptation of Generated Code" on page 1449 in chapter 37, *Adaptation of Generated Code*.

Getting Started

Unfortunately, a test sequence description expressed in TTCN cannot easily be executed as it is. This, 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 230 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.

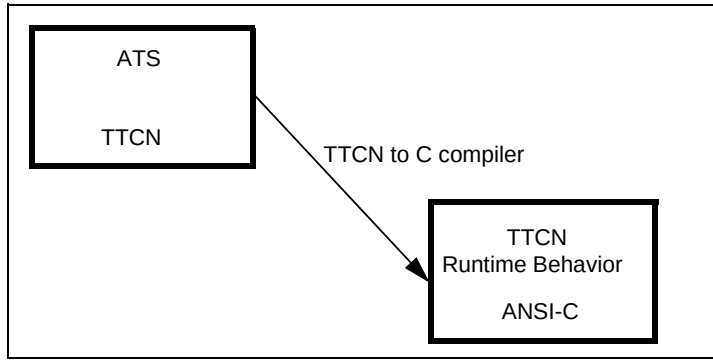

Figure 230: Translating TTCN to ANSI-C

The adaption process is described in "Adaptation of Generated Code" on page 1449 in chapter 37, *Adaptation of Generated Code*. Figure 231 displays the anatomy of the final result.

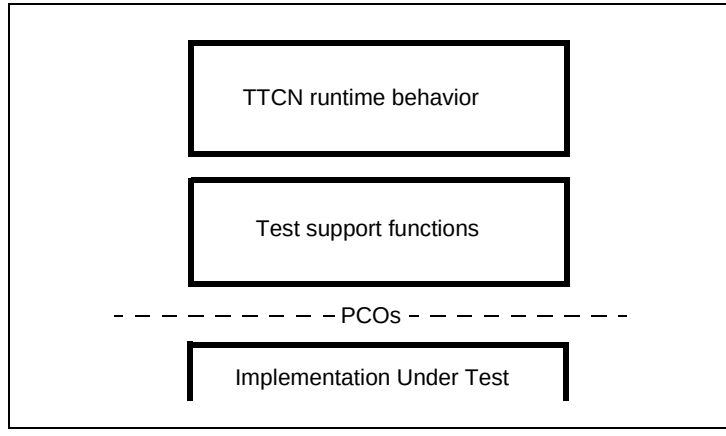

Figure 231: The anatomy of an ETS

Running the TTCN to C Compiler

To generate C code for the currently selected test document:

- Select *Generate Code* from the *Build* menu.
  – The shortcut is `<F8>`.

  This will open a dialog where you may change settings for code generation (see Figure 232).

  You can also open this dialog by the menu choice *Build* > *Options* or by the shortcut `<Alt+F8>`.

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.

Note that every active TTCN document has its own code generation settings, so in practice, two or more TTCN documents can be edited and/or viewed at the same time in the TTCN suite, and yet have individual set-

tings. (Multiple views on the same TTCN document does of course share the same settings.)

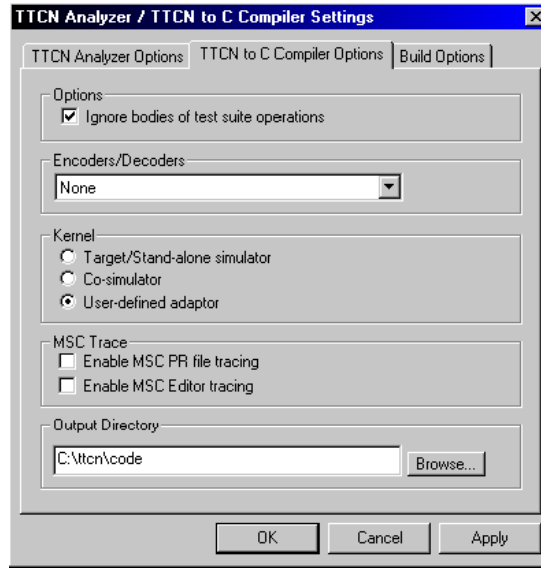

Figure 232: Options dialog for code generation

Options

- If the *Ignore bodies of test suite operations* option is checked, 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 option is not checked, the test suite operators will be generated and included in the file `tsop_gen.c`.

Encoders/Decoders

- By selecting one of the alternatives *None*, *Generate BER encoders/decoders* or *Generate PER encoders/decoders* in the drop down list the compiler will either omit generation of encoders/decoders or generate extra code for BER or PER encoding and decoding support, see chapter 37, *Adaptation of Generated Code*.

Kernel

- To build a TTCN Exerciser, select the option *Target/Stand-alone simulator*. For more information about the TTCN Exerciser, see chapter 35, *The TTCN Exerciser*.
- Select *Co-simulator* if you want to execute a test suite together with a simulated SDL system.
- To build a user-defined adaptor, select *User-defined adaptor*.

MSC Trace

- *Enable MSC/PR file tracing*

  Generates MSC/PR format files that are saved in the current working directory. By default, the MSC file names will be on the form `log_<TestCaseId>_<SequenceNo>.mpr`, where `<TestCaseId>` is substituted by the test case name of the logged test case. `<SequenceNo>` is an integer that is started by 0000 and increased by one if there is already a version n in the working directory.

  The preprocessor constant MSC_FILE_MODE should be defined at compile-time of `mscgen.c` to get this mode. The constant GENMSC should be defined for compiling `globalvar.c` to activate appropriate calls.

  If the file cannot be created, a new attempt will be done at the start of the next test case. No file log will be created.

  The function `MscSetPrefix` can be used to change the path and prefix of generated files at runtime.

- *Enable MSC Editor tracing*

  Creates a new diagram in the MSC Editor when a new test case is started and then appends and displays the events as they are executed. This mode assumes that an MSC Editor license is available and that Telelogic Tau is running at the host where the ETS is running and provides run-time MSC logging.

  The preprocessor constant MSC_MSCE_MODE should be defined at compile-time of `genmsc.c` to get this mode. The constant GENMSC should be defined for compiling `globalvar.c` for activation of the appropriate calls.

If the creation of events in the MSC Editor fails, it will be retried at the next event, possibly creating an inconsistent MSC.

This mode may also require access to the Telelogic Tau Public Interface libraries and include files that can be found in the installation of Telelogic Tau.

For more information, see "TTCN Test Logs in MSC Format" on page 1295.

Output Directory

The final thing you have to do is to set the output directory.

OK

When you have set the options, click *OK* to start the generation of code. When the generation phase is started, information about the code generation will be displayed in the Log Manager. You will receive information about what parts that were generated and also some statistics about the amount of tables traversed during the generation phase.

If code is being generated for a large test suite, the status bar will show the progress.

Running the TTCN to C Compiler from the Command Line

You can start the TTCN to C compiler by executing `ttcn2c` with command line switches. The usage is `ttcn2c [switches] filename`.

Example 205 ----------

To generate C code for `example.itex` in the current directory, with hard values and in silent mode, use the command:

```
ttcn2c -ms example.itex
```

C Code Generator Parameters

Switch	Explanation
`-a <objs>`	Link with `objs` as adaptor
`-A <ASN1files>`	File containing filenames of ASN.1 module files.
`-b <dir>`	Search in `<dir>` for static files
`-B`	Generate BER encoding/decoding interface
`-d`	Debug information in C code
`-D`	Step through ETS execution
`-e <file>`	Use `file` to be included in the make file
`-f`	Forced mode – always overwrite files
`-g`	Ignore bodies of test suite operations See also "Options" on page 1289.
`-h`	Help
`-j <file>`	User file name for types
`-k <file>`	Use file name for constraints
`-l <file>`	Specify log file
`-M file`	Enable MSC/PR file tracing
`-M msce`	Enable MSC Editor tracing
`-o <dir>`	Specify output directory
`-p <file>`	Use file name for behavior
`-P`	Generate PER encoding/decoding interface
`-r <file>`	Use file name to generate a makefile
`-R`	Disable makefile generation
`-s`	Silent mode – do not display messages
`-t`	Generate simulator See also "Kernel" on page 1290.
`-T`	Generate targe/stand-alone kernel
`-u bc`	Generate code for Borland compiler

Switch	Explanation
`-u vc`	Generate code for Microsoft Visual compiler
`-v`	Verbose
`-w`	No warning mode
`-z <lines>`	Aim at splitting generated file at <lines> number of lines

Generating C Code for Modular Test Suites

Generating C code for modular test suites from the command line, may be a bit tricky. Please see the example below for an explanation of how to proceed.

Example 206 ––

You want to generate C code for the modular test suite `mts.itex`, which depends on `module1.itex`, `module2.itex`, `module3.itex` and `module4.itex`. To do this enter:

```
ttcn2c """mts.itex""""""module1.itex""""""module2.itex""""""module3.itex"
""""""module4.itex"""
```

Note that there are six quotation marks between the file names.

This will generate code for the entire modular test suite.

It is also possible to use MP documents in the same manner.

For a description of file naming, see "Files in the TTCN Suite" on page 24 in chapter 2, *Introduction to the TTCN Suite (in Windows), in the TTCN Suite Getting Started*.

––

What Is Generated?

Figure 230 shows how ANSI-C code is generated from a TTCN test suite. The generated code alone is not executable as it needs the test support functions module (see Figure 231).

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. This will not be needed if you are compiling the generated code in a separate development environment.

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 generated code from 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 if ASN.1 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

The TTCN standard conformance log, though it is a relatively complete logging format, has a few drawbacks for some purposes:

- The conformance log is very detailed.
- The conformance log requires knowledge of TTCN and its semantics.
- The conformance log is not always particularly easy to read.

In order to get a more high-level view, the log must either be filtered or another approach needs to be taken – in this case the MSC log format. The current version of the TTCN suite has support for an additional logging mode, to generate MSC/PR standard conformant logs – automatically saved in files and also directly to the MSC Editor.

For limitations in the MSC logging, see "MSC Logging" on page 57 in chapter 2, *Release Notes, in the Release Guide*.

MSC Logging Applications

There are several applications of MSC logging. Here are some:

- Test reviews

  The MSC log format is very readable and can be used for reviewing test progress and for evaluating test results with requirements specifications.

- Test result recreation

  If an SDL model has been used to specify the system behavior, a composed MSC log can be used to re-create the test result by using the SDL Validator's Verify-MSC feature. This will allow a particular test behavior to be handed to system designers who can recreate the result on a workstation level.

- Test re-engineering

  If MSC traces are being used to generate test cases, the tests can be modified at the MSC level. A modified MSC can be used to generate a new tests. For instance – a test written in TTCN generates an MSC. At a review of the MSC logs a new scenario may be identi-

fied. The MSC can be changed to reflect the new scenario, and the MSC can be translated back to TTCN.

MSC Logging Modes

There are three major modes of logging, which can be toggled between the test cases. It is not possible to change the logging mode in a running test case since that would cause an inconsistency in the MSC file. The modes are:

- None

  No MSC logs are generated at all. This mode improves performance somewhat compared to the other modes. For best performance, the MSC generation code can be entirely disabled. For more information see "Compiling an ETS with MSC Generation" on page 1297.

- Composed

  MSC generation of externally visible events of the ETS. This is suitable for test regeneration and for re-viewing test results. The performance is better than Decomposed mode. The generated MSC views the ETS as a set of PCOs that interact with a IUT. The ETS is not visible as one particular instance. Some relative timing information may be lost in this mode.

- Decomposed

  MSC generation of externally visible and internal events of the ETS. This is the most detailed mode and it produces MSC events for almost all actions of the ETS. The purpose of the MSCs that are generated is to visualize the internal events and configuration of the ETS, and for test case debugging purposes.

MSC Instances Generated

Entity	Composed	Decomposed	Comment
SUT	Yes	No	
PCO	Yes	Yes	
MTC	No	Yes	
PTC	No	Yes	Dynamic

Events Logged

Event	Composed	Decomposed	Comment
Final Verdict	Yes	Yes	MSC Text
Preliminary Verdict	No	Yes	MSC Condition
Create	No	Yes	
Done	No	Yes	
Send to PCO	Yes	Yes	
Send to CP	No	Yes	
Receive from PCO	Yes	Yes	
Receive from CP	No	Yes	
Implicit send	No	Yes	
Start Timer	No	Yes	
Cancel Timer	No	Yes	
Timeout Timer	No	Yes	
Implicit Cancel	No	Yes	End of TC
Implicit Receive	No	Yes	End of TC
Message Values	Yes	Yes	

Compiling an ETS with MSC Generation

The ETS Generator can be compiled for a workstation or an embedded environment. Some modifications need to be done for applying it in the embedded environment if it lacks a file system. By default the tool will allow for any of these combinations at code generation time. For more information see "MSC Generation Modifications" on page 1298.

Definitions that may be changed to customize the MSC generation:

Definitions	Explanation
MSC_DEFAULT_SYSTEM_NAME	Name of IUT instance in composed mode.

Definitions	Explanation
MSC_MAX_VALUE_LEN	Maximum length of an encoded message value.
MSC_NO_VALUE_LOGGING	If defined, no message values are displayed – this may make the ETS more efficient and the MSC logs more readable when using complex messages.

MSC Generation Modifications

Modifications to the MSC generation for running on an embedded environment include:

- The MscAppendPREvent function can be modified to transfer the PR events via any type of uni-directional communications device. It is recommended to create another definition similar to the MSC_MSCE_MODE or MSC_FILE_MODE and complement the `mscgen.c` file with these.
- Any place where MSC_FILE_MODE or MSC_MSCE_MODE is used should be having a new definition for the new logging mode.