Chapter

27

Analyzing TTCN Documents (on UNIX)

This chapter contains a reference manual to the Analyzer in the TTCN suite. A description of the *Find Table* command is also included. It can, for example, be used when you want to find erroneous tables that are displayed in the Analyzer log.

The TTCN Browser and the TTCN Table Editor are described in chapter 25, *The TTCN Browser (on UNIX)* and chapter 26, *The TTCN Table Editor (on UNIX)* respectively.

See chapter 3, *Introduction to the TTCN Suite (on UNIX), in the TTCN Suite Getting Started* for an overview of the TTCN suite toolset.

> **Note: UNIX version**
>
> **This is the UNIX version of this chapter. The Windows version is chapter 32, *Analyzing TTCN Documents (in Windows)*.**

The TTCN Analyzer

The TTCN Analyzer in the TTCN suite does a complete syntax check on both TTCN and ASN.1 (as it is used in TTCN). The Analyzer also performs a number of static semantic checks, mainly the uniqueness and existence of identifiers.

For implementation reasons, the Analyzer does not exactly follow the TTCN standard – some syntax restrictions and relaxations apply. These differences will not affect the vast majority of users, but for reference they are documented together with the standard syntax in chapter 39, *Languages Supported in the TTCN Suite*. This chapter also includes the static semantics supported by the TTCN suite.

During analysis, the TTCN suite compiles an extensive symbol table containing all named objects in the TTCN document and the references between them. This symbol table is used not only during the static semantic checks, but also by tools such as the Selector tool and the Compare tool.

The Analyzer can be called from various tools: from the Organizer, from a Browser, or from a Table Editor. It is possible to analyze an entire TTCN system, a whole TTCN document or just selected portions of a TTCN document (e.g. a single test case or all the PDU definitions).

The Analyzer considers a TTCN system to not contain any Flat View document, so it needs to be explicitly applied in order to analyze the flat view.

Using the Analyzer from the Organizer

Invoking the Analyzer from the Organizer is described in "Analyze TTCN" on page 117 in chapter 2, *The Organizer*.

Using the Analyzer from a Browser

The Analyzer works either on the entire TTCN system or on selected items in the Browser it is applied on. Selections can either be made by using the mouse or by using the Selector tool (see "Using More Complex Selections" on page 1117 in chapter 25, *The TTCN Browser (on UNIX)*.

Selections can be arbitrary – the TTCN suite will analyze each selected item independently.

To analyze an entire TTCN document, do a *Select All* in the TTCN document browser and then apply the Analyzer. Alternatively apply the Analyzer on the TTCN document in the Organizer (see "Analyze TTCN" on page 117 in chapter 2, *The Organizer*).

Tools > Analyze

Conducts a syntax and static semantic check on the TTCN system or on the selected items (tables) in a Browser.

If an item is analyzed as incorrect, this will be recorded as an error message in the Analyzer log. Erroneous items are indicated by a black analysis bar in the Browser. On a color display this bar will be red.

Showing Progress

The Analyzer will, if invoked for a document or a whole system, display the progress made in the relevant Browser status window. This mechanism also provides the means to abort the analysis while it is running. Aborting is accomplished by clicking in the status window.

Tools > Analyze & Stop

Conducts a syntax and static semantic check on the TTCN system or on the selected items (tables) in a Browser but stops on the first found error and shows the table containing that error in a Table Editor.

- To execute this command via the quick-button, `<Ctrl>` must be pressed.

Using the Analyzer from a Table Editor

When used from the Table Editor the Analyzer will be applied to the contents of that particular table.

Tools > Analyze

Conducts a syntax and static semantic check on the table contents.

If a field in a table is analyzed as incorrect this will be recorded as an error message in the Analyzer log. Erroneous fields are indicated by a black status bar and the field is shaded grey (red on a color display).

Tools > Analyze & Stop

Conducts a syntax and static semantic check on the TTCN system but stops on the first found error and replaces the table in the Table Editor with the table containing the found error.

- To execute this command via the quick-button, `<Ctrl>` must be pressed.

Note:

When using the Table Editor, the *On Field* menu choice in the *Help* menu displays the TTCN BNF syntax applicable for the relevant field.

The *Analyzer* Dialog

The Analyzer uses the same dialog when started either from a Browser or a Table Editor:

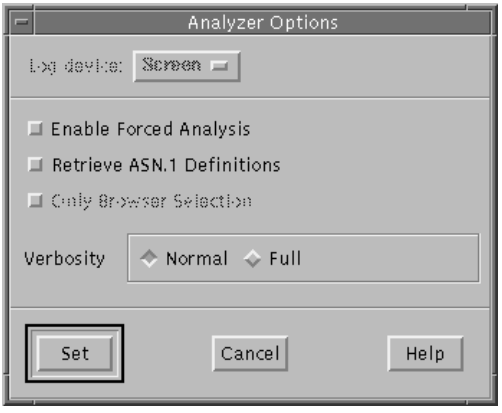

Figure 212: Analyzer dialog

Log Device

This option controls the log device for the Analyzer. The default value is *Screen*, but the log can be directed to a named file by choosing *File* or turned off altogether by choosing *None*.

For a full description of this command see "The TTCN Suite Logs" on page 6 in chapter 1, *User Interface and Basic Operations*.

Enable Forced Analysis

Indicates that all selected items are going to be analyzed and no item will be skipped. An item is normally skipped if it is OK.

Retrieve ASN.1 Definitions

Indicates that the definitions of encountered ASN.1 references should be retrieved. An ASN.1 definition is retrieved if this option is set or if the definition has not been fetched before.

Only Browser Selection

Check this item if the analysis shall be restricted to the Browser selection (this option is not applicable if started from a Table Editor).

Verbosity

Full verbosity gives more detailed information. The following extra information is available with full verbosity:

- Processing Cyclic references

  The items which will be left with incorrect references and incorrect analyze status are listed.

- Detect unused objects

  If applied on a complete test suite without finding any errors, an additional search for unused objects is performed.

Error Messages

Errors detected during analysis are recorded in the log. For the fields in the header of a table the error messages have the general format:

```
================================
Table name and table type
Field type 1
Error message
Field type 2
Error message
:
:
Field type n
Error message
================================
```

Example 200 ----------

An error found in the Default Name (identifier) field of the Default Dynamic Behaviour called UT_DEFAULT:

```
==========================================
Analysis messages in table: UT_DEFAULT of type:
Default Dynamic Behaviour
 Default Name:
The referenced identifier UPPER_PCO is not declared.
UT_DEFAULT (p : UPPER_PCO)
----------------^
==========================================
```

In the body of a table (where the field type name does not uniquely define a field) the row number is also included:

- Row number
- Body type
- Error message

Example 201 ----------

Errors found in a Test Step table header (the Default field) and in the body (rows 1 and 2 of the Behaviour Description):

```
==========================================
Analysis messages in table: ESTABLISH_CONNECTION of
type: Test Step Dynamic Behaviour
 Default:
Mismatched number of parameters: is 1, should be 0.
   Row: (#1)
      Behaviour Description:
The referenced identifier CR is of wrong type.
UPPER_PCO ! CR
------------^
   Row: (#2)
      Behaviour Description:
The referenced identifier CC is of wrong type.
UPPER_PCO ? CC
------------^
==========================================
```

Showing Error Messages

It is possible to display the Analyzer error message (if any) that is associated with the field in a table that has the current input focus. To do this, select *Error Messages* from the Table Editor *Show* menu.

Resolving Forward References

During analysis, the TTCN suite does two main tasks:

- It checks the **syntax** of the abstract TTCN document.
- It checks some of the **static semantics** of the TTCN document.

The syntax of TTCN is defined in ISO/IEC 9646-3 Annex A as a list of BNF productions or rules. See also "The TTCN-MP Syntax Productions in BNF" on page 1530 in chapter 39, *Languages Supported in the TTCN Suite*. The TTCN suite checks all these rules except for a very few minor deviations; for more information, see "The Restrictions in the TTCN Suite" on page 1526 in chapter 39, *Languages Supported in the TTCN Suite*). If a rule is not followed in the TTCN document then this will be reported as an error by the Analyzer.

The static semantic rules of TTCN are also defined in Annex A of ISO/IEC 9646. These are in the form of text statements which **limit** the use of some of the syntax rules, usually in a specific context. "TTCN Static Semantics" on page 1559 and "ASN.1 Static Semantics" on page 1575 in chapter 39, *Languages Supported in the TTCN Suite* describe the static semantics which are controlled by the Analyzer. The static semantics currently supported are mainly of the following types:

- Identifiers (names) are checked for uniqueness with respect to the scoping rules defined in ISO/IEC 9646-3
- The existence of referenced TTCN objects (e.g. a Test Step) from another TTCN object (e.g. Test Case) is verified
- Type control and type restriction control

In order to achieve this, the Analyzer constructs a symbol table of all the named TTCN objects that it finds during analysis, e.g. variable names, type identifiers, tables, etc.

This symbol table is used by some of the other tools, such as the Selector and the Reporter. This explains why it is necessary to ensure that the TTCN document is analyzed **before** applying these tools – the symbol table is updated each time the TTCN document is analyzed.

It is important to note that TTCN allows *forward reference*s. For example a Test Step may attach another Test Step that is declared **later** on in

the TTCN document, that is, the referenced (i.e. attached) Test Step, appears **after** the Test Step that has done the attach.

The Analyzer has a feature called *back-pass* designed to resolve forward references. The back-pass is automatically invoked at certain points in the analysis process.

Forward references usually appear in the context of ASP/PDU definitions, constraints and as the result of attachment statements in behaviour trees. This is the reason that the Analyzer does not traverse the items to be analyzed in exactly the listed order. The Analyzer order deviates from the listed order in a few places:

- Items in the PDU Type Definitions sub-tree are analyzed before items in the ASP Type Definitions sub-tree.
- Items in the PDU Constraint Declarations sub-tree are analyzed before items in the ASP Constraint Declarations sub-tree.
- Items in the Defaults Library sub-tree are analyzed before items in the Test Step Library sub-tree and those are analyzed before items in the Test Cases sub-tree.
- Items in the Declarations Part, Constraints Part and Dynamic Part are analyzed before any items in the Import Part (where applicable) and overviews.

ASN.1 External Type/Value References

References to ASN.1 definitions in external ASN.1 modules can occur in the following four TTCN tables:

- Test Suite Constant Declarations By Reference
- ASN.1 Type Definitions By Reference
- ASN.1 ASP Type Definitions By Reference
- ASN.1 PDU Type Definitions By Reference

In these tables there are three important fields to consider:

- *Module Identifier* (set by user)
  The name of the ASN.1 Module. This should be the real name of the model, i.e. the name stated inside the module itself.

- *Type Reference/Value Reference* (set by user)
  This is the name of the desired type (or value).
- *Type Definition/Value* (set automatically when fetched)
  This is the field where the fetched type definition/value is copied into.

The term *definition* is used below both to denote an ASN.1 type definition and an ASN.1 value.

When an ASN.1 reference is analyzed, the referred definition must be available to the Analyzer. In the pro formas for this kind of reference in the TTCN suite, there is an extra column where both the definition and the parse tree is stored.

Since the ASN.1 module is accessible from the Organizer the Analyzer can fetch the definition. If the Analyzer encounters any reference to a type/value, the definition is fetched, given the type/value name and the module identifier, and copied into the external field before the field is analyzed. The operation of fetching the definition of the reference is controlled by an option in the Analyzer dialog. Important to note is that the fields *Module Identifier* and *Type/Value Reference* are not parsed. This means that if a change is made in either of these two fields, the row will not be analyzed and thus no fetching will be made. The solution to this is to set the *Enable Forced Analysis* in the Analyzer dialog when altering a reference. See "The Analyzer Dialog" on page 1184.

To use external ASN.1 module references in the TTCN suite, information about the referred ASN.1 module must be available. That is why the ASN.1 module must appear in the Organizer and a *dependency* must be defined from the ASN.1 Module to the TTCN document. See "Dependencies" on page 137 in chapter 2, *The Organizer*.

Analysis can be invoked either from the TTCN suite or from the Organizer. The only difference, regarding dependencies, is that when the analysis is made from the TTCN suite the dependencies are fetched explicitly while the Organizer always precedes an analysis with a dependency update.

Analyzing ASN.1 References

This is the general algorithm when encountering an ASN.1 in the analysis phase.

1. First of all, an attempted fetch is only made if no current definition exists or if the Analyzer fetching-option is selected.
2. The actual fetching is made, see "Retrieving External ASN.1 Definitions" on page 1190.
3. If the definition is successfully fetched, the identifiers in the definition is converted to TTCN compatible syntax, see "Syntactic Conversion" on page 1190.
4. Finally a possible update of the definition is made and the analysis is continued.

Retrieving External ASN.1 Definitions

From the ASN.1 module a simple parse tree is built to access its contents. The parse tree is (re-)built when either it has not been built before or the ASN.1 module has been modified since last access. The parse tree for an ASN.1 module "lives" for the duration of the whole analysis phase.

Syntactic Conversion

The received definition must be manipulated in the following sense. All identifiers must be syntactically checked and altered if they include any characters that is disallowed in TTCN. The rule is that all '-' (dash) characters in an ASN.1 name is replaced by a '_' (underscore).

No special care has to be considered to ASN.1 comments since they are ignored when the parse tree is built from the module.

External type references in the type definition (e.g "... Module.Type...") will only be converted like above and left for the Analyzer to deal with (see also "Restrictions" on page 1191).

Restrictions

These features are not supported in TTCN:

- External type reference identifiers on the form *ModuleIdentifier.Type/Valuereference* are not supported.
- Type/value references within the same ASN.1 module are not supported.

Furthermore there are restrictions in the ASN.1 Utilities when it is used to extract the external types and values. These restrictions are introduced in translating ASN.1 to SDL but since the same algorithm is used for extracting types and values, the restrictions are also relevant for the TTCN suite. For more information, see "Translation of ASN.1 to SDL" on page 700 in chapter 14, *The ASN.1 Utilities*.

Error Handling

Any error that occurs while finding, parsing or accessing the ASN.1 module will cause the old definition to remain. Only when a completely successful fetch has been made, the definition will be updated. Possible errors could be:

- The ASN.1 Module not found among the dependencies.
- The ASN.1 Module file could not be found or was not readable.
- The ASN.1 Module file was not syntactically correct.
- The referred type/value could not be found in the module.

See chapter 14, *The ASN.1 Utilities* for a more general view on ASN.1 usage.

TTCN Static Type Restriction Control

The TTCN standard defines a set of semantic rules for type definitions. In particular it defines simple types as types which might contain a true subset of the values which the parent type might contain. The Analyzer now analyzes the TTCN type system and also reports violations of these rules. This analysis is implemented as a post-pass over the related tables, and is only applied when the previous passes have been successful.

Furthermore, restriction control is performed on a number of other constructs in the TTCN language, thus facilitating the task of writing semantically correct TTCN documents. Most of the TTCN types and values are checked and thereby reducing the chance of programming errors

slipping through to other tools which depends upon the correctness of the TTCN document.

Application of Static Type Restriction Control

Static Type Restriction Control is applied to at least the following TTCN tables and fields:

- Simple type definitions
- TTCN structured type / ASP / PDU / CM definitions
- Test Suite Constant Declarations
- Test Suite Variable Declarations
- TTCN Structured Type / ASP / PDU / CM Constraint declarations
- Actual constraint reference parameter lists
- Actual parameter lists

The Static Type Restriction Control also evaluates expressions in advance, where possible, and checks that the result does not violate the type restrictions of the field where the value is found. It operates on both types and values, regarding specific values as special cases of types.

Reducing Level of Restriction Control

Setting the environment variable `ITEX_RESTRICT_LEVEL` to the value `NONE` will disable the strict checking of type restrictions. This might be useful for allowing test suites with deliberate type errors pass through the analysis. A setting of `SOME` will relax the analysis somewhat, and is recommended before trying `NONE`. In order to get the most strict analysis (default), the value `STRICT` might be used.

Generating a Flat View

A Flat View is the part of the modular TTCN suite environment that presents the tree structure of an expanded modularized TTCN source. This complete view includes all explicit defined objects in the TTCN system.

The Flat View interactive interfaces is similar to the Browser. Unlike the Browser, the Flat View is started (generated) from the Organizer.

The structure of the TTCN source in a Flat View is similar to a Test Suite. Neither the Import tables nor Export tables are displayed in a Flat View.

Generating a Flat View in a system, requires that at least the root document of the system to be connected to a file. The connected document must be of one of these types:

- Test Suite
- Modular Test Suite
- TTCN Module

If this condition is not fulfilled, the operation Generate Flat View fails. Also if the current Flat View is modified or is locked (by any user) the operation fails.

If a Flat View is edited by the user, it must be saved in another document file. The user is asked for a new file name. Otherwise, if the existing Flat View file is not modified, it is overwritten by the new generated Flat View if the generation is redone.

For each TTCN system in the organizer only one instance of Flat View can be generated. A Flat View can be regenerated. A Flat View includes all explicit defined objects in the system at the time of the (re)generation.

The first time the Flat View in a system is generated, the name of the Flat View file is the same as the file name of the root document in the system if the name is not already used. The file extension for a Flat View file is `.ifv`.

An already existing Flat View is always connected to a document file. If this existing Flat View is in the current target directory (and neither is modified nor locked) the Flat View file is reused (the Flat View is regenerated). Otherwise a new Flat View file in the current target directory is used.

Algorithm

When the command Generate Flat View is invoked the following cases may occur:

No Flat View is Available

In other words this is the first time a Flat View is generated for the given system. A new Flat View is generated in the target directory. The name of the document file is generated. This name is derived from the file name of the root document in the system. A Flat View icon is added to the selected TTCN system in the Organizer and it is connected to the generated document file.

A Flat View is Available

If the file connected to the existing Flat View either is locked or modified the operation fails. If the existing file is not in the target directory do as the previous case except not to add a Flat View system node.

Otherwise if the existing file is in the current target directory the file is reused (the current document file name must have been derived from the file name of the root document in the system).

If the Flat View is modified, the operation (Generate Flat View) fails and the user is asked to save the Flat View (into another file) before regenerating.

Save a Flat View

A user modified Flat View must be saved as a document of type (non-modular) Test Suite in another file than the Flat View file.

When the user invokes the save operation from the Organizer, a new file name must be given otherwise the save operation fails. If the user invokes the save operation from the TTCN suite a file browser pops up and the user is asked for a file name.

Merging Objects into a Flat View

A Flat View contains a copy of all explicitly defined objects in a system. When a Flat View is generated for a TTCN system, explicitly defined objects in each system node is merged into the Flat View. The merged objects may have the same name.

Objects of the following types are treated special:

Overview Tables

The main table in the Overview Part is the Test Suite Structure table. Depending on the type of the root node in the system the Overview tables in the generated Flat View contain the following information.

Test Suite (Modular or Non-Modular)

This type of system nodes have a Test Suite Structure table. The header of the corresponding table in the Flat View inherits the information in the header of the root table.

The Detailed Comments in the index tables are copied to the index tables in the Overview part of the generated Flat View.

TTCN Module

In this case the TTCN Module Exports table is the corresponding table to the Test Suite Structure table in the Flat View. The TTCN Module Exports contains two extra header fields: Objective and TTCN ModuleRef. These two fields are ignored when the Flat View is generated.

The Detailed Comments in the index tables are copied to the index tables in the Overview part of the generated Flat View.

Package

In this case there is no corresponding table to the Test Suite Structure table in the generated Flat View.

Import and Export Tables

These tables contain no explicit defined object and therefore they are ignored.

External Tables

These tables contain no explicit defined object and therefore they are ignored.

Multiple Tables

The objects in the multiple tables are added to the corresponding multiple table in the generated Flat View. The Detailed Comment in the multiple table is added to the Detailed Comment in the corresponding multiple table.

Single Tables

Add each single table to the appropriate place in the generated Flat View.

Group Nodes

For each group node if it does not exist already in the generated Flat View create a new group node.

Finding Tables

The Find Table tool searches for and displays named tables. The name searched for is given by the currently marked piece of text in any TTCN suite window. This tool is very useful when debugging a TTCN document from an Analyzer log.

Finding Tables by Name

The Find Table tool is available from the *Tools* menu in the *TTCN* sub menu in the Organizer, from the *Tools* menu in a Browser and from the *Tools* menu in a Table Editor. Unlike most other tools it does **not** require a selection: it applies to the context given by where the tool is started (i.e. a TTCN table or a TTCN document), and its action is the same whether it is invoked from the Organizer, from a Browser or from a Table Editor.

Tools > Find Table

Finds a named table or the table in which a named object, such as a test case variable, is declared. The name searched for is given by the currently marked piece of text in a TTCN suite window. If no text is marked Find Table will use the contents of the text clipboard (used for *Cut* and *Copy*).

A typical use of this tool would be to debug a TTCN document when analyzing. Erroneous tables identified in the Analyzer log window can be easily found by marking the names of these tables in the log window and applying the Find Table tool.

The Find Table tool is able to find named objects with the name specified as `<document name>::<object name>` (e.g. `Test_Suite_A::SEND`), where `<document name>` is the name of a document in the TTCN system and `<object name>` is the name of an object available in the context of that document. This may also be specified as `<document name>__<object name>` (i.e. with underscores instead of colons).

> **Note:**
>
> The Find Table tool will in fact use marked text as the look-up string from any X Window, not just TTCN suite windows.

Example 202 ––

Marking the PDU name *CON_req* in, say, the behaviour line *L! CON_req (X:=1)* of a test step, and then choosing the Find Table tool will cause the PDU definition of CON_req to be displayed (assuming it exists).

On the other hand, marking the test case variable *X* in the same line would cause the Test Case Variables table, in which *X* is declared, to be displayed.

––