T3D Documentation

T3D Documentation Outline

T3D is a documentation generation tool for TTCN-3. It utilizes multiple technologies to transform TTCN-3 source code (with or without tagged paragraph comments) into a set of easily navigable offline HTML webpages. In addition to the main view which allows navigation through the constructs in the source code, there are two relational / dependency views that correlate certain particularly important constructs of the TTCN-3 language, namely the Module Parameer / Testcase View and the Import / Module Dependency View, which correlate testcases and module parameters, and modules respectively.

Installation / Setup

The following software must be installed on the system prior to the use of T3D

- A Java Runtime, at least version 1.6 (http://java.sun.com).

Additionally, a standards compliant internet browser with activated JavaScript support is required to be able to optimally use the documentation content generated by the tool.

The T3D command-line tool requires two environment variables to be set no matter what the operating system is.

- `JAVA_HOME` - should point to the Java Runtime
- `T3D_HOME` - should point to the installation directory of T3D

The installation tool for Windows should take care of setting these environment variables correctly including the installation of Java if necessary. For Unix environments, these have to be configured manually (we will refer to MacOS as Unix environment as well). In order to use T3D properly, we suggest to include `T3D_HOME` in the path variable of your Unix system. The Windows Installer automatically takes care of the inclusion in the system path.

Using T3D

T3D is a command line tool which can be called by simply typing "t3d" in Windows or on Unix systems. The Windows command-line can be accessed by starting the program "cmd". For Unix systems, we only support bash environments.

Configuration

T3D uses an XML-based configuration format. Starting with version v1.0.0, the location of the configuration file must be supplied as a command line option during execution (the location was previously based in the default user's data location - depending on the system, a file `t3d.xml` was stored in `%APPDATA%\T3D` (Windows) or in `~/.t3d/` (Unix)). The location of the `%APPDATA%` directory depended on the username, the Windows version and possibly the localization. For example, for a user 'foobar' on a German Windows XP machine, the resulting configuration path would be `C:\Documents and Settings\foobar\Application Data\T3D`. In version v1.0.0 and onwards, the location of the configuration file has to be supplied every time T3D is called. Additionally, starting with version v1.0.1, if there is no existing configuration file in the selected location, the tool will prompt the user to use the appropriate option to generate a new configuration with the default settings. Thus, the tool has to be started with the appropriate parameter to generate a new configuration prior to actual usage. This way, it will be possible to modify the configuration file to accommodate the particular needs of the user prior to generating any output. By location here the path, filename, and extension of the configuration file are meant, which implies that any filename and any extension can be used. It is probably best to retain certain norms in naming the configuration files, at least in preserving the file extensions (`.xml`) to avoid confusion. If the fact that a new XML configuration is automatically generated if existing one is not found is too confusing, there may be a separate option available in the command line to generate new default configuration files in future releases.

Please note that after an update, during the development stage, the configuration files will often be extended. Thus, it may be necessary to generate a new configuration file and transfer the custom settings from the old configuration file. The configuration file has two sections:

A configuration profiles section
A main section

The **ConfigurationProfiles** section contains a list of **DocumentationProfile** elements. Its elements contain the tags **profilename** as well as elements with settings for the documentation generation, such as output path and paths to files needed for the generation among others. In the default `t3d.xml` that is created, there is a profile called *defaultProfile* that initializes all profile configuration values to the default values. This is also the configuration of an implicitly given profile called *all*. The *all* profile is not part of the XML configuration, but it exists and is known to the tool.

Since v0.4, configuration profiles also have a version (**profileVersion** element), that regulates the profile compatibility. If a profile from an older version is used, T3D will throw and error and recommend profile upgrade or the selection of another profile. Also, further robustness checks have been introduced to provide hints if the configuration profile is otherwise incompatible or corrupted. In case of a problem that can be localized, a corresponding error message is provided suggesting the location of the problem.

Also, there are several generic options, that affect the overall behavior of the T3D tool. The known extensions for files that are to be processed can be specified by means of a regular expression in the **resourceExtensionsRegExp** option. By default, *ttcn*,*ttcn3* and *3mp* file extensions are recognized. Since v1.0.1, there is also support for supplying input by means of what will be referred to as *project* files. These files specify a list of input files and directories (including wildcards). The **projectExtension** option regulates what is to be considered a project file. Note that this is not a regular expression option, therefore only a single value is currently supported. There is an option to switch recursive processing on and off (**settingRecursiveProcessing**). There is an option to switch aborting on (parsing) errors on and off (**settingAbortOnError**). Both of these options are turned on by default. Turning them off may result in unreliable output. Files that contain syntax errors will be documented only up to the point of the first syntax error occurrence.

Then there are the options regulating the documentation generation process and the logging output (further details are featured below), and the options defining the paths to dynamically linked files necessary for the documentation generation process (which do not concern the user in the usual usage scenarios). These files are described in more detail below.

Starting with v1.0.1, it is possible to define custom names for the supported documentation tags in the **commentTagsConfiguration** section of the configuration.

Finally, there are options regulating the output. Currently there is only one setting - the output directory (**outputDirectory**), where the output files will be placed. Other options regulating the output have been relegated to the command line interface. Note that this option can also be overridden by a corresponding command line option.

In the *main section*, there is currently one single configuration element **defaultConfigurationProfile**. It points to the implicit *all* profile by default. The **all** profile has all settings set to their default values. The **defaultConfigurationProfile** is the profile that will be used if no specific profile is provided as command-line parameter. If the specified **defaultDocumentationProfile** does not exist in the configuration, T3D will fall back to the implicit *all* profile.

Some of the profile options may be moved to the *main section* in the future, as they are rather generic.

A default newly generated configuration file currently looks like this:

```
<T3DConfig>
  <ConfigurationProfiles>
    <DocumentationProfile>
      <profileName>defaultProfile</profileName>
      <profileVersion>v1.0.1</profileVersion>
      <resourceExtensionsRegExp>ttcn|ttcn3|3mp</resourceExtensionsRegExp>
      <projectExtension>t3p</projectExtension>
      <settingRecursiveProcessing>true</settingRecursiveProcessing>
      <settingAbortOnError>true</settingAbortOnError>
      <loggingConfiguration>
        <showFullPath>false</showFullPath>
        <showFilename>true</showFilename>
        <showMessageClass>true</showMessageClass>
        <showDetails>true</showDetails>
        <logOutputPrefix>   </logOutputPrefix>
      </loggingConfiguration>
      <statShowSummary>true</statShowSummary>
      <statShowLOC>true</statShowLOC>
      <commentTagsConfiguration>
        <descTag>desc</descTag>
        <authorTag>author</authorTag>
        <configTag>config</configTag>
        <exceptionTag>exception</exceptionTag>
        <memberTag>member</memberTag>
        <paramTag>param</paramTag>
        <purposeTag>purpose</purposeTag>
        <remarkTag>remark</remarkTag>
        <returnTag>return</returnTag>
        <seeTag>see</seeTag>
        <sinceTag>since</sinceTag>
        <statusTag>status</statusTag>
        <urlTag>url</urlTag>
        <verdictTag>verdict</verdictTag>
        <versionTag>version</versionTag>
      </commentTagsConfiguration>
      <includeConstructBody>true</includeConstructBody>
      <hideConstructBody>false</hideConstructBody>
      <showOriginalT3DocTags>false</showOriginalT3DocTags>
      <checkUndocumentedParameters>true</checkUndocumentedParameters>
      <checkConsistentTagUsage>true</checkConsistentTagUsage>
      <checkIdenticalDescriptionTags>true</checkIdenticalDescriptionTags>
      <checkCyclicImports>true</checkCyclicImports>
      <cssFile>$T3D_HOME/css/doc.css</cssFile>
      <jsFile>$T3D_HOME/js/doc.js</jsFile>
      <xsltFileHTML>$T3D_HOME/xslt/html.xsl</xsltFileHTML>
      <xsltFileImport>$T3D_HOME/t3d/xslt/html_import.xsl</xsltFileImport>
      <outputDirectory>DOCUMENTATION</outputDirectory>
    </DocumentationProfile>
  </ConfigurationProfiles>
  <defaultConfigurationProfile>all</defaultConfigurationProfile>
</T3DConfig>
```

where {$T3D_HOME} is substituted by the value of the T3D_HOME environment variable as specified in the setup when the configuration file is generated. This means that the same profile may not be directly usable on a different configuration, where the T3D_HOME environment variable has a different value. This will be subject to change in the future in that at least substitution with the environment variable is done at runtime in the default configuration.

The options regulating the documentation generation process and logging output have the following effects:

> **includeConstructBody** (by default *true*) - The documentation will contain the bodies of module definitions. Leaving out the bodies contributes to a more abstract and concise documentation. However the details may often be necessary.
> **hideConstructBody** (by default *true*) - The module definition bodies are hidden by default when an HTML documentation page is loaded. This makes the initial presentation more compact.
> **showOriginalT3DocTags** (by default *true*) - The HTML documentation will use the original T3Doc tags (e.g. "@author John Doe" instead of

"Author(s): John Doe").
checkUndocumentedParameters (by default *true*) - The T3D generation process log output will include warnings on undocumented formal parameters
checkConsistentTagUsage (by default *true*) - The T3D generation process log output will include warnings on incorrect tag usage in consistence with the TTCN-3 Documentation Specification Standard (for example: multiple version or other tags that may only be used once, etc.)
checkIdenticalDescriptionTags (by default *true*) - The T3D generation process log output will include warnings identical description tags (defined by string equivalence currently, more sophisticated identity relation may be employed in future releases)
checkCyclicImports (by default *true*) - T3D will check the project for cyclic imports (which may take up to several minutes on large and complex test suites) and report any occurrences in the documentation generation log output.

Command-Line Usage

T3D is used as follows:

```
t3d [options] ((--html | --xml-only) | --local-dependencies)+
    (filename | path)+
```

This seemingly complicated syntax indicates several peculiarities of the command line usage of the tool:

Apart from any required options (see below), there always have to be at very least two other parameters that needs to be specified. One of these is the input parameter, which can be either a path that contains the TTCN-3 files that should be analyzed, the name of an individual file, or any combination of these (a mixed list), including wildcards.

If a path provided as input (and recursive processing is enabled in the configuration, which is the default setting), T3D will recursively parse and analyze all files in this directory that match the provided file extensions (which are specified in the configuration file) and generate documentation for all of them. For the current path a simple '.' is sufficient. If no files match these extensions, T3D will output a corresponding message and quit.
If an individual file is provided as input, then only that file will be processed and documented.
If the path or filename contains spaces, you need to put the path into quotation marks or use the auto-completion feature of the environment (provided there is one), which should take care of escaping spaces or enclosing the complete path in quotation marks.
If a list of individual files and/or paths are provided as input, these will be combined and processed together.
If wildcards are used, these will be expanded by the command-line environment into a list and subsequently analyzed as such.

The other mandatory parameter is the output format, which may (currently) be one of three possible options:

`--html` is probably the most likely option. It will cause the generation of the complete HTML documentation, including intermediate XML files.
`--xml-only` will result in the generation of the intermediate XML files only, without the HTML files.
`--local-dependencies` will result in the generation of an intermediate XML file containing the local dependencies as described below.
The other intermediate XML files and the HTML documentation will not be generated if only this output option is selected.

The local dependencies intermediate XML file may be generated in addition to either the other intermediate XML files or the complete HTML documentation, including the intermediate XML files by using two output format parameters together (e.g. `--local-dependencies --html`, the order is irrelevant; `--html` supersedes `--xml-only`, meaning it makes no sense to use both).

A summary of the available options is provided below:

```
    --generate-config <FILE NAME>   Generate a new default configuration
                                    file at the specified location
    --config <FILE NAME>            Configuration file location

    --profile <PROFILE NAME>        Configuration profile

    --html                          Generate HTML documentation, including
                                    intermediate XML files
    --xml-only                      Generate intermediate XML files only
    --local-dependencies            Generate local dependencies XML file

    --verbosity <LOG LEVEL>         Verbosity level (currently supports ERROR,
                                    WARNING and INFORMATION values)
    --output-path <PATH>            Destination path for the output (if
                                    applicable, otherwise ignored).
                                    Overrides the profile setting.
    --help                          Show this usage information screen
```

The options not discussed so far are:

`--help` will provide brief usage information. T3D will stop and document no files when the help screen is called.
The `--generate-config` option (new as of v1.0.1) allows the generation of new default configuration files at the location specified. T3D will then quit.
The `--config` option (new as of v1.0.0) is mandatory and has to be specified every time T3D is run (except when `--help` or `--generate-config` are used). It specifies the location of the configuration file. Starting with v1.0.1, if no configuration file is found at the specified location, the user will be prompted to use the appropriate option (`--generate-config`) to produce a new default configuration. To use the default location from previous versions one will have to specify it manually, e.g.

```
t3d --config ~/.t3d/t3d.xml
```

on a UNIX based system or

```
t3d --config %APPDATA%\T3D}\t3d.xml
```

on a Windows based system. The `--generate-config` and the `--config` options are mutually exclusive, with `--generate-config` having precedence, meaning that if both are specified, T3Q will still only generate the new default configuration and quit. Please not that if the location of the configuration file contains spaces, it has to be either enclosed in quotation marks or the auto-completion feature of the environment has to be used to take care of escaping the spaces.

The `--profile` option overrides the **defaultConfigurationProfile** in the XML configuration. This means that you can specify multiple profiles in the XML configuration and run T3D using another existing profile without the need to change the XML configuration. If the profile specified on the command-line does not exist, T3D will automatically fall back to the default profile provided in the main section of XML configuration. In turn, if this default profile does not exist as well, T3D will fall back to the implicit *all* profile. If the configuration profile name contains any empty spaces, they need to be escaped or the profile name needs to be enclosed in quotation marks, depending on the environment.

The `--verbosity` option (newly introduced in v1.0.0) regulates the verbosity level of the output messages based on their type. The possible values are *ERROR*, *WARNING* and *INFORMATION* (in an ascending inclusive order, meaning that when *INFORMATION* is selected, the output will include both *ERROR* and *WARNING* verbosity level messages as well). *INFORMATION* is the default setting. More information about the message types is available in the next section.

The `--output-path` option (newly introduced in v1.0.1) makes it possible to supply the output path at the command line interface (overriding the profile setting). It is added purely for convenience in case the same profile has to be used on different projects with different destination paths.

The native binary executable for Windows supplied with v1.0.1 is discarded again as of v1.0.2, due to the fact that it does not grant the desired advantages. Thus, the basic usage is reverted to the batch scripts. Starting with v1.0.2, the batch scripts include a "hidden" `--echo` option, which simply outputs the deployment specific call to the Java virtual machine, **without** any command line arguments, meaning that it makes no sense to provide any further command line parameters besides the `--echo` option. The sole purpose of this option is to output the full command line necessary for starting the tool, which may be necessary for embedding into third party tools. This is why this option is considered hidden (it also does not show in the standard help screen). It should not affect the general usage of the tool. If, for whatever reason, other command line arguments are supplied together with the `--echo` option, then the `--echo` option needs to be the first command line argument.

Linked Files

A summary of all linked files as specified in the configuration file, their default location and their function is presented below:

cssFile `$T3D_HOME/css/doc.css` - The CSS file defining the layout in the HTML output
jsFile `$T3D_HOME/css/doc.js` - The JavaScript file defining basic functionalities for the navigation and other interactive parts
xsltFileHTML `$T3D_HOME/xslt/html.xsl` - The HTML transformation file used when generating HTML output for the main and the module parameters views
xsltFileImport `$T3D_HOME/xslt/html_import.xsl` - The HTML transformation file used when generating HTML output of the import view

These linked files can be used to customize and manipulate the output of the T3D tool as described in more detail in the subsequent sections. Please note that changing these files requires adequate knowledge of the technologies involved, the lack of which may cause broken functionalities in the generated content or failure in the generation process. Upon request, some customizations and adjustments may be provided with subsequent or customized versions of the tool.

Output Formats

The output of T3D can be split into several categories. There is the *main output*, which is the product of the documentation process, and, in the general perception, comes in the form of multiple HTML files and several XML files, which store extracted data in an intermediate format and serve as a basis for the documentation generation. Then there is also the output of the documentation generation process itself, which will be referred to as the *documentation*

generation log. The details follow below.

Main Output

Before getting into the details of the main output, first the location of the output and its calculation need to be described.

Output Paths

T3D takes the base output path from the **outputDirectory** element in the XML configuration. In this directory, depending on the input a sub-directory will be created for each successful T3D execution. The main output includes the following entities:

> XML files containing the information extracted from the input TTCN-3 file(s)
> An `html` sub-directory containing the generated HTML, CSS, and JavaScript files

The **real** output directory is calculated from the input. If the input is a directory then the name of the last directory in the input path (the directory that is actually to be documented) is selected as destination the name of the sub-directory: For example, given input path `a/b/c` (relative) and base output path `c:\Temp\T3D\`, the generated documentation will be placed in `c:\Temp\T3D\c\`. If the input is a filename, then a sub-directory will be created with the name of that file in the base output directory: For example, given input filename `templates.ttcn`, the output will be generated in `c:\Temp\T3D\templates\`. Any subdirectories leading to the input filename will be disregarded, e.g. generated documentation for inputs such as `Libraries\templates.ttcn` or `c:\TTCN\Libraries\templates.ttcn` will still be placed in the `c:\Temp\T3D\templates\` destination directory. Particular cases, such as documenting the current directory, e.g. "`.`" or "`./`", will result in documentation being generated in a sub-directory in the base output directory with the name of the current directory, that is calling `t3d --config myConfig.xml --profile myProfile .` while in `c:\TTCN\Libraries\` will generate the documentation in `c:\Temp\T3D\Libraries\Libraries\` (assuming the same base output path is used as in the above examples). If multiple inputs are provided (e.g. multiple files, directories, or by means of wildcards), the name of the first entry in the list is taken for the destination sub-path.

XML Output

The XML output consists of (currently) three XML files storing the relevant data extracted from the TTCN-3 sources in a structured intermediate format. The currently generated XML files are:

> `project.xml` - This file contains the most comprehensive information and represents an abstracted version of the source TTCN-3 files, including module structure, the individual elements, their bodies (if configured so), and T3Doc comments for every module, group, and element of the processed TTCN-3 files. It serves as a basis for the main and the module parameter / test case views (for details on the separate views, see Section Views below).
> `imports.xml` - This file contains information about the imports and dependency relations of the processed TTCN-3 modules. It serves as a basis for the import / module dependency view.
> `dependencies.xml` - This file can be thought of as a blend between an abstracted version of the main `project.xml` file and a low-level version of the `imports.xml` file, featuring a compact representation of the low-level dependencies at the module definition (element) level - it contains all the module definitions and all the known and unknown elements referenced directly within each module definition, where unknown elements are currently prefixed with `unresolvedReference---`. There is no separate view associated to this file, since it is basically a compact representation of the main view and its main intent is the use with third-party tools to perform custom tasks, such as slicing or markup of definitions related to a particular module definition (e.g. approved / locked definitions, etc.).

These XML files serve as a basis for the HTML generation. They can also be utilized by custom third-party tools to produce different output depending on the particular needs. More information on this is available in the T3D technical documentation.

HTML

The main output consists of interlinked HTML files. There are three different views serving different purposes.

There is the **Main View**, which serves as a browseable representation of the source code, with a separate page for every every module and every construct. The output currently includes and shows the source code for every module and construct and all references to known constructs are linked to the corresponding page for these construct.

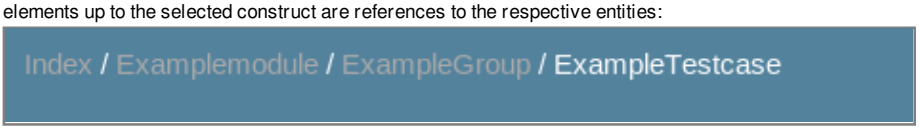
Then there is the **Module Parameter/Testcase View** that shows a summary of the module parameters related to a particular test case and vice versa. Optionally it can also display the path to each reference, which is useful for indirect references (i.e. a module parameter is referenced in a function or an altstep called within another function, which in turn is called within the test case).

Finally, there is the **Import View** which should serve as an overview of the dependency relations between modules. This view is currently based on the import statements only and does not take into account whether imported definitions are actually used in the importing module. Apart from direct dependencies, it also lists distinct indirect dependencies (i.e. when module1 imports from module2 which in turn imports from module3, when module1 is selected, module3 will be listed as an indirect dependency, since for consistent use of module1 both module2 and module3 will be necessary).

Layout

All three views are embedded into a more or less identical layout. The current default layout consists of several sections:

A header section, which currently includes a *path navigation*. The path navigation currently features the view index and the structural path to the selected construct, including the selected / containing module, any (nested) groups the construct may be in and the construct itself. All the elements up to the selected construct are references to the respective entities:



T3D Path Navigation

A left hand side navigation bar consisting of a view selector, presentation options, a module index and a construct index, where:

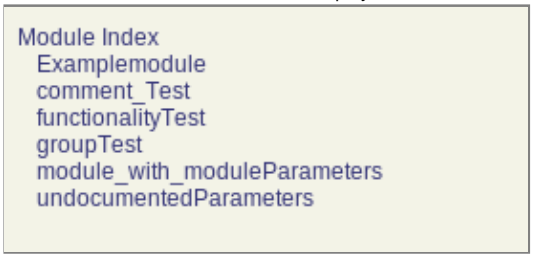
The *view selector* allows the user to switch between different views on certain constructs, e.g. switch between the the main view, the module parameter view, and the import view on the currently selected module, or between the main view and the module parameter view on the currently selected test case. If a certain view is not available for a particular construct, then upon selecting that view, the module index for that view will be displayed.

T3D View Selector

The *presentation options* control whether for example the bodies of the module definitions are displayed or not in the main view, or whether the contents of the import statements are displayed or not in the import view.

T3D Presentation Options

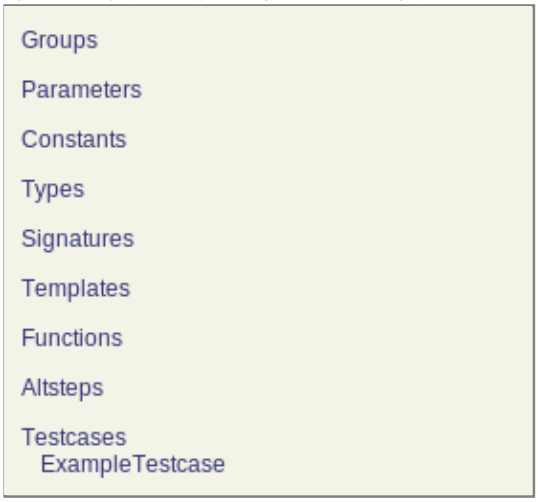
The *module index* allows the user to display the module documentation in all different views, for all the documented modules.



T3D Module Index

The *construct index* lists all the construct categories in the main view, and only test cases, module parameters, and groups in the module parameter view. The contents in each category depend on the currently selected module or group. If no module or group is selected (i.e. the module index of a view is selected), then all the documented constructs in all modules are listed. If a module or a group is selected, then only the constructs defined within that module or group are listed. To make the presentation more compact and manageable, the construct index categories are all collapsed. By clicking on a category, the list of constructs of that category is expanded below the category in a familiar expand-collapse fashion. In the import view, this area is used to display clarifications on the color scheme used to mark the module dependencies.

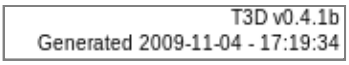
Both the module index and the construct lists under the separate categories, as well as elements in the other views are sorted alphabetically (currently sorting is broken though).



T3D Construct Index

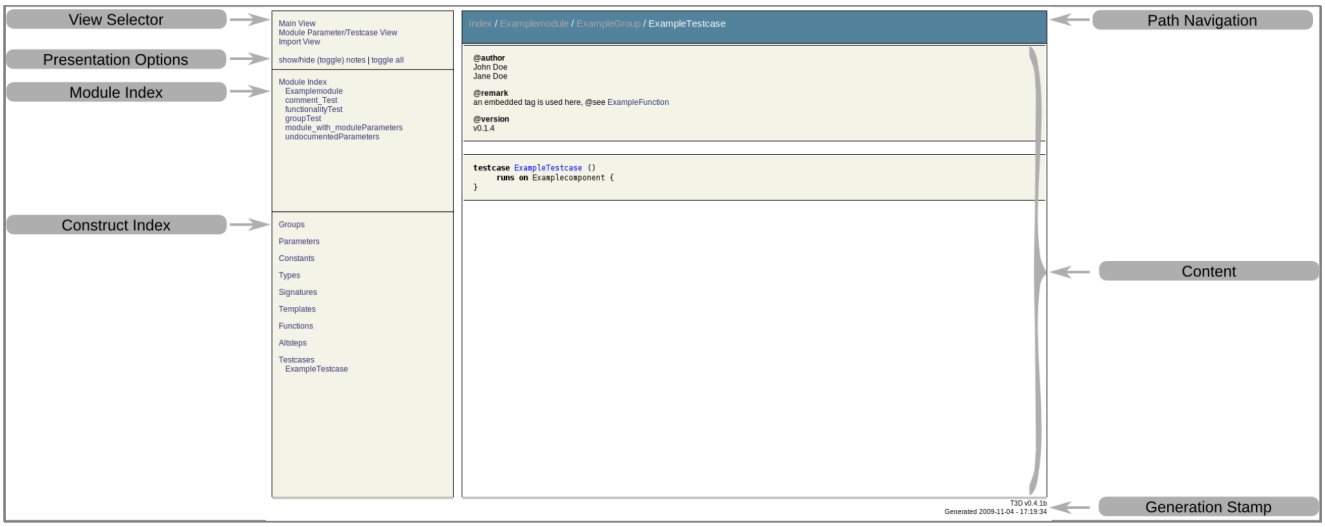
A *content section* on the right hand side, which contains the actual content of the selected view on the selected construct.

A *generation stamp* in the lower-right corner indicating the tool name and version, and the generation date and time for the documentation page.



T3D Generation Stamp

The annotated screenshot below summarizes the complete layout for the main view of an example module.



T3D Annotated Main View

The main output has been tested so far under:

Windows: Firefox 3.x, Opera 9.51, SeaMonkey 1.1, Safari 3.1.2, Internet Explorer 7
Linux: Firefox 3.x, Opera 10.00, Konqueror 4.2.2

CSS and Layout Customization

This default layout can be further customized with the help of a Cascading StyleSheet (CSS) file. The file is named `doc.css` and is placed in a `css` sub-directory under the `html` directory where the generated HTML files are located. The default CSS file is copied from the location specified in the configuration profile, which by default is `$T3D_HOME/css/doc.css`. Once changes are made to the CSS file in the output folder, they can be transfered back to the source CSS file or saved in a separate location and referenced in the documentation profile, so that the updated CSS file will be used the next time this profile is selected for the documentation of TTCN-3 test suites. Be aware that if the same output path is specified for the same input, all files will be overwritten, including any customized CSS files.

The default CSS file shall serve as a reference for customized CSS files. It is still in development and not yet complete. Comments within the file provide hints about the selectors. Further finer details can be deduced by examining the generated HTML code.

The CSS customizations are mostly for the presentation of the generated documentation. To influence the actual generated contents, the Extensible StyleSheet Language Transformations (XSLT) file can be customized. It also needs to be customized shall new CSS selectors be introduced. More on XSLT in section *The Documentation Generation Process* below

JavaScript

The HTML output also makes use of a basic JavaScript (JS) file (by default located in `T3D_HOME/js/doc.js` and copied to a `js` sub-folder in the output directory), which provides some basic functionalities necessary for the enhanced navigation and toggling of details. Same rules as with the CSS file apply, customizations need to be moved upstream if they are to be reused in future documentation generations. Also adaptation of the XSLT file may be necessary if major new JS functionalities are introduced. Additionally, a second JavaScript file called index.js is generated at runtime and then copied to the same folder as `doc.js`. This file is used to generate the *construct index* navigation in the main and module parameters views of the HTML documentation. Since this file is automatically generated with each documentation generation, it makes less sense to manipulate it upstream, instead it can be used for fine-tuning the final output. Due to the use of JavaScript, if support for it is not available or disabled in the user's browser (due to security or other concerns), the navigation will not work properly.

Documentation Generation Log

This log contains information about the generation process itself and is provided initially in the form of console messages which indicate the progress of the documentation generation process and show any errors or problems that are encountered during in the process. This log was also exported as an XML file for convenience and automated post-processing by third party tools. As of v1.0.0 this is no longer the case. The generation of an XML log file has been suspended until such functionality is reintegrated into the new common logging interface. The log file file was called `log.xml` and was located in the target directory where the other XML files from the main output are located as well. The particular format of the log messages will be described in more detail in the relevant sections concerning the causes and contexts of their occurrence. Ultimately, all messages shall have a uniform format, however currently some of

them still have different formats. Further information about the XML file itself will be available in the T3D technical documentation once it is reintegrated.

There are currently three types of log messages:

- Information
- Warning
- Error

Information

Information messages include general information about the documentation process, its progress, as well as brief statistics at the end. Some of the messages may change to status-type messages in the future, once such type of log messages is introduced to T3D. Currently the following types of information are considered information messages:

- Status reporting (parsing a file, processing a file)
- Runtimes of the separate phases of T3D
- The number of generated HTML documentation files
- Brief statistics including
  - A list of all processed files
  - A list of all files processed without errors or warnings
  - A list of all files processed with errors or warnings

Not all of these messages are explicitly marked as *INFORMATION* messages as of yet, but this will be a subject to change in the future, where the log output format will be uniform for all log messages.

Warnings

Warnings are associated with problems that occur during the documentation generation process or violations of certain norms as specified in the Documentation Specification part of the TTCN-3 standard (ES 201 873-10). These warnings resemble the warnings in T3C quality checks and due to this fact, these features may be moved to T3C in the future. It possible to turn each check or group of checks (a large portion of the checks falls under the *consistent tag usage* category) on and off as described in the configuration section.

Empty Tags

In case a tagged paragraph is specified, but the tag body is empty, a corresponding warning will be issued with the location of the occurrence.

Example

```
/*
 * @remark
*/
```

will result in:

```
testFile.ttcn: 17: WARNING: Empty tag found: @remark
```

Note that the warnings currently refer to the line number of the module definition and not of the documentation tag itself. This concerns all warning messages.

Misplaced Tags

The use of certain tags in tagged paragraphs is restricted to certain contexts. If a documentation tag is used in association with a module definition that does not permit the use of such a tag a warning will be issued.

Example

```
/*
 * @config text
*/
function f1(){
     ...
}
```

would produce

```
testFile.ttcn: 31: WARNING: @config tag found (may not be used here)
```

Identical Description Tags

If identical description tags (@desc) are used to describe two or more different element definitions within all the processed modules, a warning will be thrown. The identity checking mechanism so far is rather simple, based on a direct string comparison. An alternative approach may be implemented in the future, using a different notion of identity, shall this become necessary.

Example

```
/*
 * @desc Descrption1
*/
function f1(){
     ...
}
/*
 * @desc Descrption1
*/
function f2(){
     ...
}
```

results in

```
testFile.ttcn: 31: WARNING: Identical @desc tag found: "Descrption1" (testFile.ttcn:37)
```

Too Many Tags

While certain tags may occur multiple times (e.g. @author tags), other can only occur once. If a tag that can occur only once per definition as specified in the standard, occurs multiple times a warning will be thrown.

Example

```
/*
 * @version 0.01
 * @version 0.02
*/
function f1(){
     ...
}
```

will result in

```
testFile.ttcn: 31: WARNING: Multiple @version tags found (may only contain one)
```

Undocumented Parameters

Formal parameters shall be described by means of the corresponding @param tags. If this is not the case and if **showUndocumentedParameters** is enabled in the configuration profile, a warning will be thrown.

Example

```
/*
 * @param a Description of Parameter a
*/
function f1(integer a, boolean b){
     ...
}
```

will result in

```
testFile.ttcn: 31: WARNING: Undocumented parameter found: b
```

Note: There are more subtle details related to the (proper) documentation of formal parameters. In future considerations, these subtleties may be implemented more throroughly.

Documented But Nonexistent Parameters

If on the other hand a tagged description is present for a formal parameter that is not (or no longer) a part of the module definition, a warning will be issued with details of the occurrence.

Example

```
/*
 * @param a Description of a
 * @param b Description of b
*/
function f1(boolean b){
     ...
}
```

results in

```
testFile.ttcn: 31: WARNING: Documented parameter not found: a
```

Cyclic Imports

In adition to the pure documentation-related checks, there is currently also a cyclic imports check, which throws warnings if cyclic imports occur. This is of particular importance for the import view, since cyclic dependencies will be impossible to represent with the current presentation format. Cyclic imports shall also be detected by most compilers since there are a lot more problems associated with them. The warning contains the cyclic import and there is a separate warning for each module that is part of the cycle.

Example

```
module mod1{
    import mod2;
}
module mod2{
    import mod3;
}
module mod3{
    import mod1;
}
```

produces three warnings

```
mod1.ttcn: 1: WARNING: Cyclic imports found: mod1 -> mod2 -> mod3 -> mod1
mod2.ttcn: 1: WARNING: Cyclic imports found: mod2 -> mod3 -> mod1 -> mod2
mod3.ttcn: 1: WARNING: Cyclic imports found: mod3 -> mod1 -> mod2 -> mod3
```

Error

Error messages are produced on unrecoverable errors, such as a parsing error (although there is an option in the configuration that would allow bypassing parsing errors, but it is disabled by default and it is recommended that it stays that way for reliable results).

HTML Views

The generated HTML documentation currently includes three views:

- Main View
- Module Parameter/Testcase View
- Import View

In addition, an XML file with an intermediate representation for low-level dependencies is generated, but no HTML view is available for it due to the fact that it represents more or less an abstracted and summarized version of the main view for special purposes. Some form of HTML presentation may become available in future releases. The low-level dependency representation will be discussed briefly in the respective section and in more detail in the technical documentation.

Main View

The purpose of the main view is to provide a browseable representation of the source code, with a separate page for every every module and every construct defined within the processed files. Additionally, tagged paragraphs used for documentation are extracted and presented in a separate field. Whether the bodies of the separate constructs are included in the documentation and if so whether it is displayed by default is controlled by the configuration

options in the profile. With the help of the presentation options in this view, functionalities to expand and collapse the bodies of constructs can be enabled and used. The source code presentation is formatted using the default settings of the code formatter. The settings may be made available in the configuration profile for better customization in future releases. For convenience, TTCN-3 keywords are highlighted.

Example:

Source code:

```
/*
 * @desc this is an example module
 * @version 0.1
*/
module ExampleModule{
        function examplefunction() runs on exampleComponent{p1.send(integer:1);examplefunction2()}
        function examplefunction2() runs on exampleComponent{}
}
```

HTML:

Description:
this is an example module

Version:
0.1

```
module ExampleModule {
     function examplefunction ()
          runs on exampleComponent {
            p1.send ( integer:1 );
            examplefunction2 ()
     }
     function examplefunction2 ()
          runs on exampleComponent {
     }

}
```

Hiding the construct bodies from the presentation options or excluding them during the documentation generation will result in:

```
module ExampleModule {
     function examplefunction ()
          runs on exampleComponent
     function examplefunction2 ()
          runs on exampleComponent

}
```

In addition, controls for hiding individual construct bodies can be shown via the presentation options, so that only particular construct bodies are shown / hidden:

```
module functions (toggle){
     import from types all;
     import from modulepars all;
     import from configuration all;

      group mtcFunctions(toggle) {
          function f1 ()
               runs on sampleComponent1 (toggle){
                 p1.send ( integer:1 );
                 p2.receive ( charstring:"a" );
          }
          function f2 (
                 integer par1,
                 charstring par2
               )
               runs on sampleComponent1 (toggle)

     }

      group ptcFunctions(toggle)

}
```

Module Parameter / Testcase View

The module parameter / testcase view serves a summary of the module parameters related to a particular test case. This works both ways in that it can serve as a summary of the test cases influenced by a particular module parameter. At the index level it lists all the test cases and all the module parameters with their respective relations in a tabular format. Both the list of test cases and the list of module parameters can be shown or hidden using the toggle controls. Also, by clicking on the respective *Testcases* or *Module Parameters* headlines, the user can quickly jump to the respective section, given that it is present and shown. With the help of the presentation controls, the paths to the actual reference to the module parameter can be shown or hidden for more compact presentation. The paths can be useful when indirect references to the module parameters are used (e.g. references within functions or altsteps referenced in the test case). The construct index in this view shows only the relevant types of constructs - test cases, module parameters and groups. Upon selecting a module, only the test cases and module parameters defined within the selected module will be displayed, both in the module parameter view and in the construct index. Another thing worth mentioning is that upon selecting a test case or a module parameter, the entries in the construct view are also filtered to include only the constructs defined within the module in which the selected construct is defined. Upon selecting a group, aditional filtering is applied, limiting the constructs shown in the construct index to the constructs defined within the selected group.

In the tabular representation, selecting an item in the left column will show the corresponding page for the selected item in the module parameter / testcase view, whereas selecting an item in the right colum (the path to the reference) will show its page in the main view.

Example

```
// ...
modulepar integer par1 := 1;

function function1() runs on sampleComponent1 {
    //...
        p1.send ( par1 );
    p2.receive ( charstring:"a" );
        //...
}
testcase testcasef1() runs on sampleComponent1 system sampleComponent3 {
    // ...
    function1();
    // ...
}
// ...
```

whould be shown as:

(where the relevant relation from the example above is highlighted in green) and

in the module parameter and test case views respectively (with paths enabled from the presentation controls).

Note that these two parts of the module parameter / testcase view may be separated further conceptually in future releases.

Import View

The import view presents an overview of the dependency relations between modules. Put simply, a module A is said to depend on module B if module A imports definitions from module B. That is, to utilize module A, a user will also need to have module B available. On the other hand, when modifying module B, a test developer will have to take into account its uses in module A (and eventually other modules as well) in order to avoid breaking the functionality of module A, by either taking precautions to avoid any (negative) impact on module A, or by adapting module A accordingly to handle the changes in module B. This is referred to as *direct dependency*, i.e. module A directly depends on module B. On the other hand, even though import statements in TTCN-3 are not transitive, indirect dependencies often have to be taken into account as well, due to the fact that changes in one module may have far reaching implications due to a chain reaction effect. An *indirect dependency* is when a module A imports module B, which in turn imports module C. In this context module A is said to depend indirectly on module C (since changes in module C breaking the functionality in module B may have an impact on the functionality of module A).

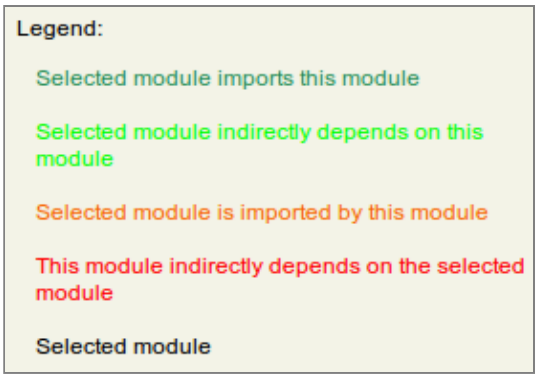
The import view tries to capture these relations and present them in a usable manner, since relations between a large number of modules may be particularly difficult to represent visually. The current approach consists of a tabular presentation format with three columns. The middle column contains a list of all processed modules. Upon selecting any of the modules in the list, the left column is populated by all the modules that are imported by the selected module (i.e. the selected module directly depends on them), followed by all the modules that are imported by the modules imported by the currently selected module (i.e. the currently selected module indirectly depends on them). Note that the list of indirect dependencies contains only such indirect dependencies which are not already listed as direct dependencies. Additionally, in the middle column, all the modules listed as direct and indirect dependencies are also colored accordingly, based on the color scheme as shown in the legend (where otherwise the construct index is located in the other views). The color scheme can be configured to the particular preferences of the user via the CSS file. The right column on the other hand illustrates the opposite relations - it lists all the modules that import the currently selected module, followed by all the modules that import the modules that import the currently selected module. The latter then filtered in a similar fashion to show modules only once. The modules depending on the currently selected module (directly and indirectly) are also colored accordingly in the middle column in a similar fashion to the modules on which the currently selected module depends.

To summarize, this layout can be perceived as a flow of imported definitions from left to right (definitions from modules in the left column are imported in modules in the middle column, whose definitions are in turn imported into modules in the right column), or as a flow dependencies from right to left, where modules in the right column depend on modules in the middle column, which in turn depend on modules in the left column. The import details (the bodies of the import statements) can be shown or hidden with the help of the presentation controls.

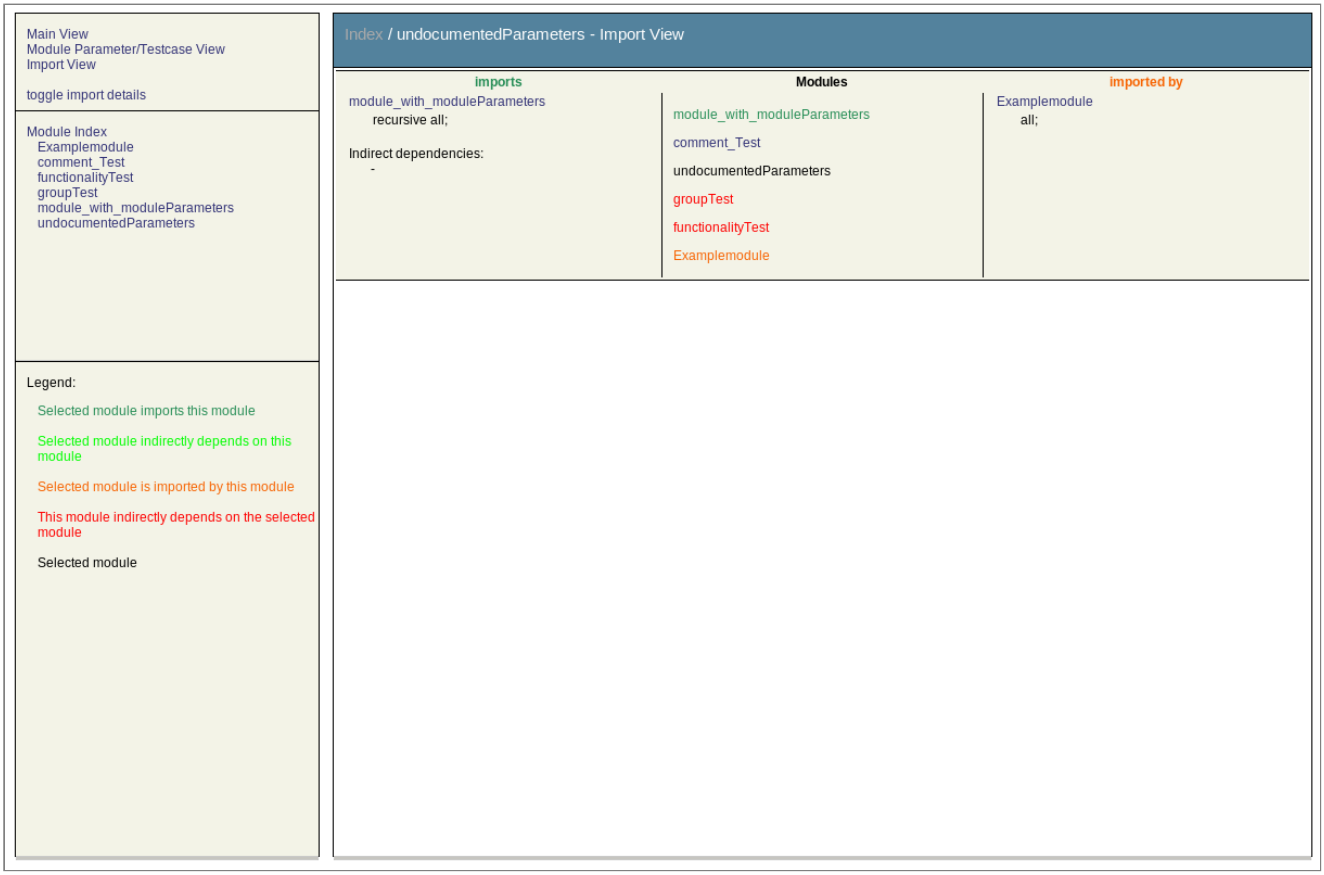
Examples

imports	Modules	imported by
configuration all; types all; functions all; Indirect dependencies: modulepars	modulepars import3 configuration testcases main types templates import2 functions import4 templates-0 import1 ExampleModule	main all;

T3D Import View Content Section



T3D Import View Legend



T3D Import Full Picture

In addition, worth noting is that import statement bodies are grouped by modules. For example, the following TTCN-3 code:

```
import from module1 {group functions};
import from module2 all;
import from module1 {testcase tc_1, tc_2};
```

would be shown as

```
module1
   group functions;
   testcase tc_1, tc_2;
module2
```

```
all;
```

to avoid redundant entries and keep the presentation compact.

The information about the dependency relations is stored in an intermediate XML representation, which can also be utilized by third party tools to create different visualizations of the relations or for other purposes. Also the XSLT file defining the transformations into HTML can be customized as well to accommodate particular needs. More information on this can be found in the T3D technical documentation.

Note also that this view is currently based on the import statements only and does not take into account whether imported definitions are actually used in the importing module.

Print View

The print view is available on all pages as a simplified printer-friendly version of their content. It discards all unnecessary elements, such as navigation entities within the page, which are of no use on printable media. This is achieved through a section in the CSS file and can thus be further customized to accommodate the user's needs.

The print view is not an "official" view (yet), in that it cannot be directly selected from within the layout. However, most modern internet browsers do support print preview functionalities that allow the user to see whether the web page can be printed properly. With or without print preview, browsers will automatically select the printable presentation format when printing a page so that the output will always be the simplified printer-friendly version of the contents (unless configured otherwise in the browser's configuration).

Below are a few screen shots illustrating the print views in the main, module parameters/testcase, and import views.

Examples

T3D v0.4.1b
Generated 2009-11-29 - 19:15:26

Index / Examplemodule

Description:
This is an example of a module in ttcn-3

Author(s):
John Doe

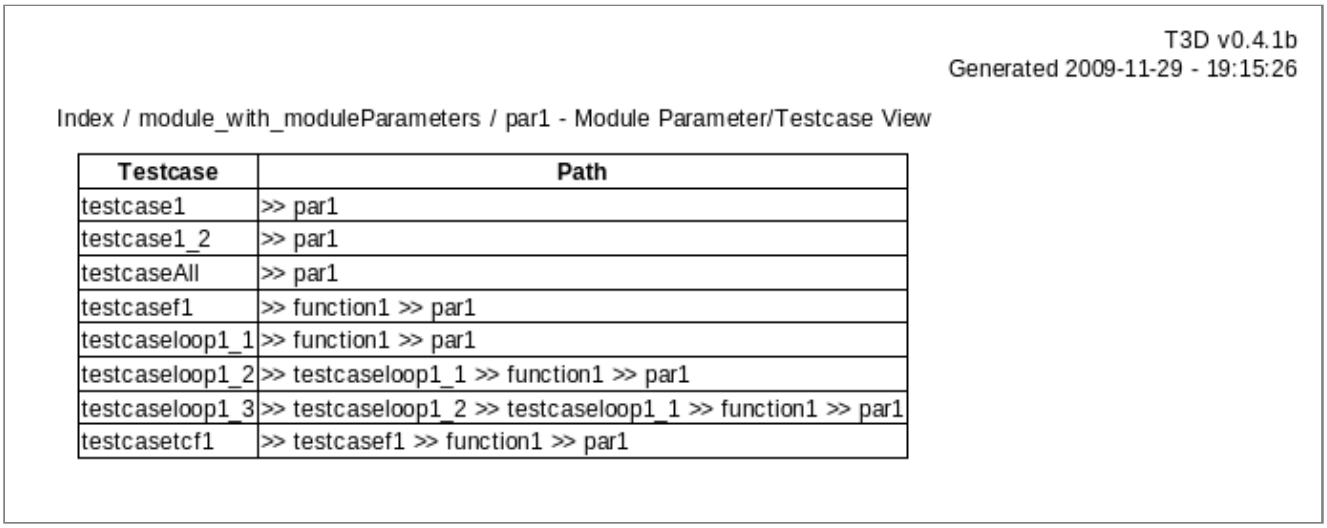
```
module Examplemodule {
     const integer con1 := 1, con2 := 2;
     import from groupTest recursive all;
     import from functionalityTest {
            group types
     }
     import from undocumentedParameters all;

      group ExampleGroup {
          testcase ExampleTestcase ()
               runs on Examplecomponent {
          }

     }
     function ExampleFunction (
            in charstring par1,
            out charstring par2
          )
          return boolean {
     }

}
```

T3D Print View - Main View

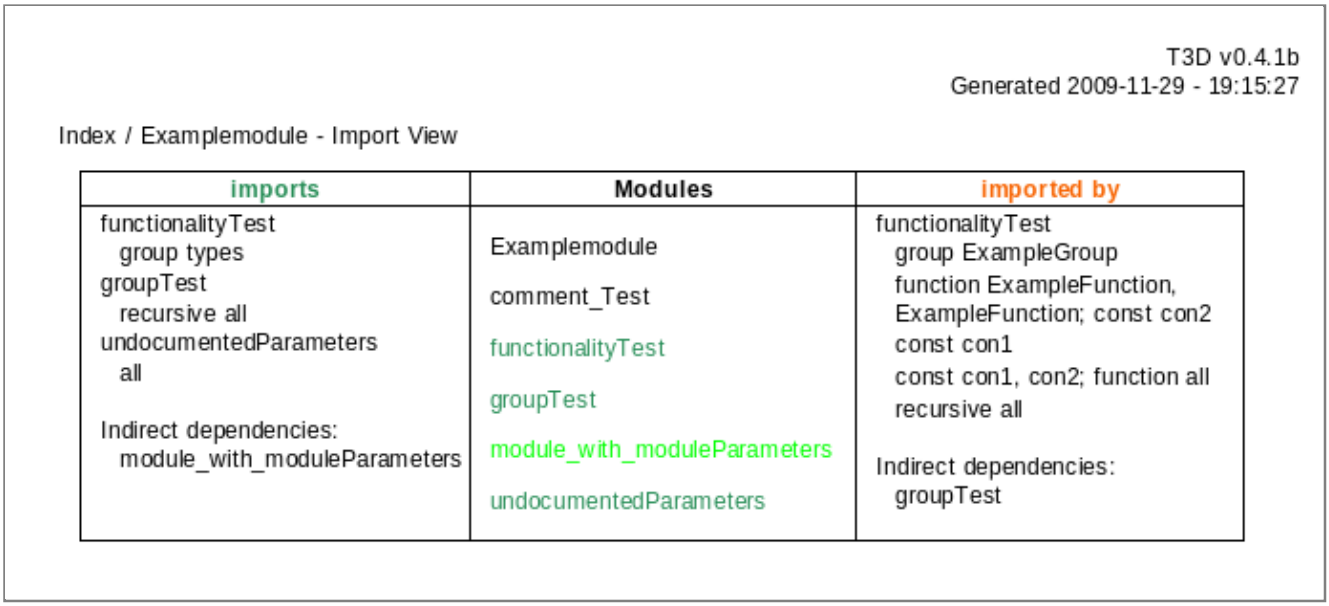


T3D v0.4.1b
Generated 2009-11-29 - 19:15:26

Index / module_with_moduleParameters / par1 - Module Parameter/Testcase View

Testcase	Path
testcase1	>> par1
testcase1_2	>> par1
testcaseAll	>> par1
testcasef1	>> function1 >> par1
testcaseloop1_1	>> function1 >> par1
testcaseloop1_2	>> testcaseloop1_1 >> function1 >> par1
testcaseloop1_3	>> testcaseloop1_2 >> testcaseloop1_1 >> function1 >> par1
testcasetcf1	>> testcasef1 >> function1 >> par1

T3D Print View - Module Parameters / Testcase View



T3D v0.4.1b
Generated 2009-11-29 - 19:15:27

Index / Examplemodule - Import View

imports	Modules	imported by
functionalityTest group types groupTest recursive all undocumentedParameters all Indirect dependencies: module_with_moduleParameters	Examplemodule comment_Test functionalityTest groupTest module_with_moduleParameters undocumentedParameters	functionalityTest group ExampleGroup function ExampleFunction, ExampleFunction; const con2 const con1 const con1, con2; function all recursive all Indirect dependencies: groupTest

T3D Print View - Import View

Low-Level Dependencies

The generated low-level dependencies serve (currently) exclusively for custom processing by third party tools for purposes such as slicing or markup of definitions related to a particular module definition (e.g. approved / locked definitions, etc.). That is the generated content for the low-level dependencies is only in an intermediate XML format and there is no HTML view for it (since it is roughly an abstracted version of the main view). Some form of HTML presentation may become available in future releases.

The low-level dependencies can be thought of as a blend between an abstracted version of the main view and a low-level version of the import view, featuring a compact representation of the low-level dependencies at the module definition (element) level - it contains all the module definitions and all the known elements referenced directly within each module definition.

The structure of the low-level dependencies intermediate representation features a list of all elements, where for each element the following pieces of

information are available:

- A unique ID of the element so that it can be uniquely referenced
- The name of the element as per its identifier
- The top-level type of the element (e.g. type, function, altstep, etc.)
- Its definition start location (the line where the definition of the element starts)
- The name of the containing module where the element is defined
- The name of the file containing the module where the element is defined
- A list of the element IDs for all the elements referenced within the current element

Upon recursive resolution a dependency graph can be created for a set of related elements.

More details about the technical side of the low-level dependencies can be found in the technical documentation.

The Documentation Generation Process

For the benefit of the users, the basic steps needed to produce documentation out of TTCN-3 source files will be briefly described to provide some insight into the documentation generation process.

The documentation generation process involves several steps. These can be summarized as follows:

1. Process the input TTCN-3 files and gather the necessary information, which results in
2. Generate and validate XML files containing all the extracted information, which is the basis for
3. Transform the generated XML files into the desired output format(s)

In more detail, step 1. includes parsing and analysis of the input TTCN-3 files. A user has little influence on this stage. The information collected during the analysis stage of step 1. is then in step 2. exported as XML files in the output path (by default named `project.xml`, `import.xml`, and `dependencies.xml`). The XML files are validated using XML Schema Documents (XSD) (stored by default under `$T3D_HOME/xsd/`). Finally, in step 3. the generated XML files are transformed with the help of the XSLT files into the target documents.

Step 3 is of most interest to the user, since it allows flexible and customizable transformation in just about any format. Currently only HTML format is available, but PDF- and TeX output, for example, are possible as well.

In the HTML generation process, the above mentioned CSS and JS files are involved, as well as the XSLT files for HTML transformations (by default located in `{$T3D_HOME}/xslt/html.xsl` and `{$T3D_HOME}/xslt/html_import.xsl`. The XSLT files for HTML transformations include all the necessary transformation steps to produce the desired HTML output. It may of course be further customized to accommodate user / project specific needs. The default XSLT files serve generally as a reference. Note that modification of these files requires a certain level of knowledge of the underlying technologies and improper modifications may result in documentation generation failures, therefore one should proceed with caution. For further information about technical details related to the customization of XSLT files, please refer to the technical documentation.