

# 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 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 is 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 Description1
 */
function f1(){
    ...
}
/*
 * @desc Description1
 */
function f2(){
    ...
}
```

results in

```
testFile.ttcn: 31: WARNING: Identical @desc tag found: "Description1" (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 **checkUndocumentedParameters** 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 thoroughly.

## Required `@desc` Tags for Functions

If enabled (**checkFunctionDescTagsRequired** configuration flag, disabled by default), this check will throw warnings if there are no `@desc` tags present for a function definition.

### Example

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

will result in

```
testFile.ttcn: 31: WARNING: Code Documentation: Required @desc tag not missing
```

## 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 addition 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).