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);examplefunction
        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.

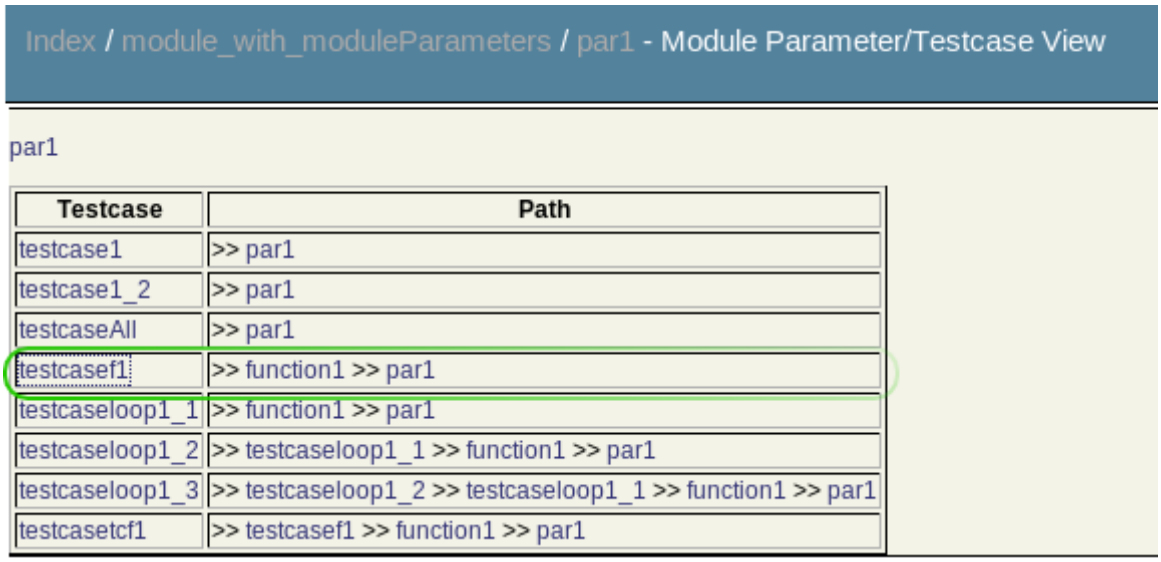
Example

```
// ...
```

```
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

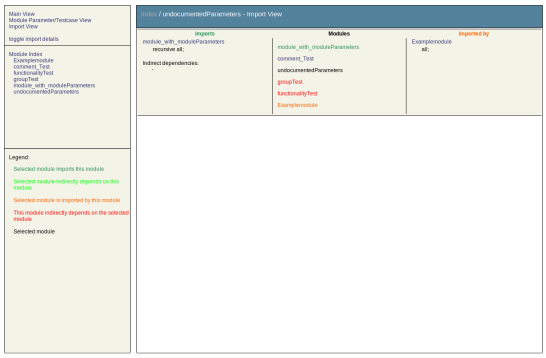
The import view presents an overview of the dependency relations between modules. Put simply, a module A is said to depend on module B if module A imports definitions from module B. That is, to utilize module A, a user will also need to have module B available. On the other hand, when modifying module 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).

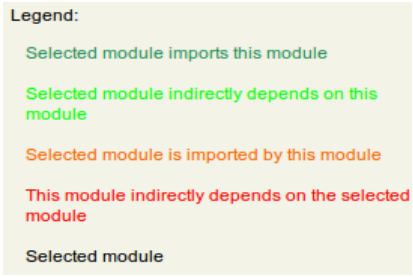
The import view tries to capture these relations and present them in a usable manner, since relations between a large number of modules may be particularly difficult to represent visually. The current approach consists of a tabular presentation format with three columns. The middle column contains a 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



T3D Import View Content Section



T3D Import View Legend

Legend:

Selected module imports this module

Selected module indirectly depends on this module

Selected module is imported by this module

This module indirectly depends on the selected module

Selected module

imports	Modules	imported by
module_with_moduleParameters recursive all; Indirect dependencies: -	module_with_moduleParameters comment_Test undocumentedParameters groupTest functionalityTest Examplemodule	Examplemodule all;

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