Specador Documentation Generator User Guide

Rev. 18.1.19
21 June 2018

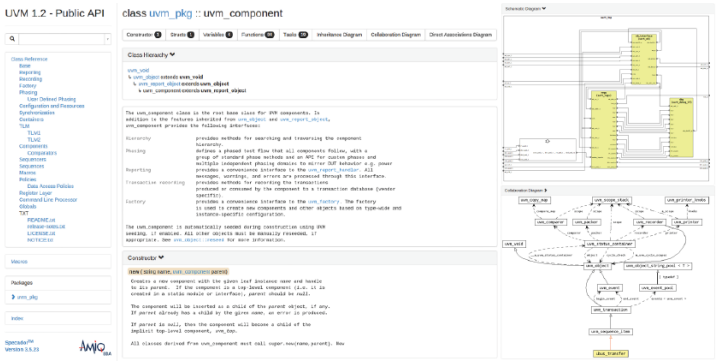
Technical Support: support@amiq.com

Table of Contents

1. Overview 1
2. How to Run 2
3. Compile Arguments 3
    3.1. Auto-config 3
    3.2. Emulating compiler invocations 4
    3.3. Compatibility Modes 5
        3.3.1. Default DVT Compatibility Mode 6
        3.3.2. vcs.vlogan Compatibility Mode 9
        3.3.3. vcs.vhdlan Compatibility Mode 12
        3.3.4. ius.irun Compatibility Mode 13
        3.3.5. questa.vlog Compatibility Mode 17
        3.3.6. questa.vcom Compatibility Mode 18
        3.3.7. gcc Compatibility Mode 19
    3.4. Paths 19
    3.5. Strings 19
    3.6. Comments 20
    3.7. Environment Variables 20
    3.8. Including Other Argument Files 21
    3.9. All Build Directives 22
    3.10. e Language Test Files 37
    3.11. e Language SPECMAN_PATH 37
    3.12. SystemVerilog OVM or UVM Library Compilation 38
    3.13. Xilinx Libraries Compilation 38
    3.14. Intel(Altera) Quartus Libraries Compilation 39
4. Compile Waivers 40
5. XML Preferences File Syntax 43
6. XML Menu File Syntax 46
7. Comments Formatting 48
    7.1. JavaDoc 48
    7.2. NaturalDocs 50
8. Customizing Documentation 52
9. What is New? 54
10. Legal Notices 63
11. Third Party Licenses 64

Chapter 1. Overview

Specador automatically generates accurate HTML documentation from e, SystemVerilog, Verilog and VHDL source code by using dedicated language parsers. It enables the user to easily generate and maintain well organized and consistent documentation based on the comments in the source code.



Chapter 2. How to Run

Specador can be invoked in batch mode by running:

$DVT_HOME/bin/specador.sh ...

Examples:

```
 $> specador.sh -lang vlog -cmd /path/to/simulation.f -title "MY CHIP"

 $> specador.sh -lang vhdl -cmd /path/to/file_list.f -preferences /path/to/dvt_export_html.xml

 $> specador.sh -lang e -ignore_compile_errors -cmd irun.args -title "USB 3.0 uVC"
```

Arguments

-lang e|vlog|vhdl
  The source code language: e Language, SystemVerilog or VHDL.
-cmd <file>
  The compilation command file.
[-ignore_compile_errors]
   Ignore compile errors and continue.
[-preferences <XML file>]
  Use preferences specified in the XML file.
[-gen_preferences_xml]
 Generate an XML file with all supported preferences.
[-menu <XML file>]
  Use menu specified in the XML file. It has precedence over the html menu specified by preferences.
[-title <title>]
  Use specified title. It has precedence over the title specified by preferences.

NOTE: When generating HTML documentation in GUI mode, a *dvt_export_html.xml* settings file is saved in the project's *.dvt* directory.

NOTE: In order to generate diagrams, make sure the Graphviz **dot** executable is installed & accessible.

Chapter 3. Compile Arguments

The compiler reads arguments from a file passed using the -cmd flag. The file may contain:

- comments
- directives in two forms:
  - +directive+arg1+arg2+
  - -directive arg1 arg2
- top files. Anything that is not a directive or comment is regarded as path to a **top file**.

Note: Unknown directives are ignored. In general, tool-specific directives start with **+dvt_**. Unknown directives that start with this prefix are flagged with a warning.

Note: Several AMIQ tools take compile arguments in the same format, for example the DVT Eclipse IDE, hence the references to DVT in the scope of this chapter.

The tool parses each top file, following includes/imports as specified by the language. Some directives (**parsing directives**) allow you to control how files are compiled based on their extension, for example using System Verilog 1800-2012 for *.sv, Verilog 2001 for *.v, VHDL 1076-2008 for *.vhdl and e Language 1647-2011 parser for *.e. The parsing directives are either generic or mode specific.

In general, the directives are similar with the arguments (or flags) that you would pass to any compiler/simulator.

An argument file may include other argument files and so on. The internal builder follows the included files as it encounters them (as if part of a continuous stream). Note that the way a file is included (for example with '-f' or '-F') has an influence on how paths inside the included files are interpreted. For more details see Including Other Argument Files.

To simplify the flow integration, the tool supports several Compatibility Modes. This capability allows you to reuse existing arguments or argument files that you already use for a particular simulator invocation.

3.1 Auto-config

Particularly for small projects, in order to simplify project configuration, instead of explicitly specifying lists of files, incdirs etc., you can use one or more +dvt_init_auto directives in default.build. For example:

```
// Identify and compile sources from the project directory
+dvt_init_auto

// Identify and compile sources from some other path
+dvt_init_auto
+dvt_compilation_root+/some/other/path
```

DVT scans the specified directories and automatically detects how to compile the source code files. For each +dvt_init_auto directive, a corresponding default.build.auto.# file is created. The .build.auto.# files contain compilation directives like incdirs, top files, libraries, UVM libraries, Xilinx libraries, etc. resulting from the auto-configuration algorithm. After scanning, DVT compiles the code using the directives in generated auto files.

DVT automatically detects and analyzes existing Intel(Altera) Quartus or Xilinx ISE/Vivado projects in the compilation root directory of a +dvt_init_auto directive. For more details see FPGA Support.

NOTE: Auto files are overwritten on every full build.

You can specify additional directives in +dvt_init_auto sections. They are copied as is in the corresponding .build.auto.# file. For example:

```
+dvt_init_auto
+define+RTL
+define+ADDR_SIZE=32
+incdir+$UVM_RGM_HOME/src
+dvt_skip_compile+*/some/path*
```

You can specify a compatibility mode and use simulator specific directives:

```
+dvt_init_auto+ius.irun
-v93
```

The available compatibility modes are: dvt, ius.irun, vcs.vlogan and vcs.vhdlan. If a compatibility mode is not specified, it defaults to dvt. See Compatibility Modes for a detailed description.

TIP: When working with large filesystem hierarchies or slow network drives, the scan phase might time out, by default after 40 seconds. To increase the timeout for a particular +dvt_init_auto section, use +dvt_autoconfig_timeout build config directive. For example to set timeout to 2 minutes:

```
+dvt_init_auto
+dvt_autoconfig_timeout+120
[...]
```

3.2 Emulating compiler invocations

Sometimes a design is compiled across multiple invocations. For example, environment variables or preprocessing directives may change their values between invocations, or sources may be compiled into different libraries.

The +dvt_init+<mode> directive is equivalent with a new compiler invocation, where <mode> represents the compiler compatibilty mode. The directive resets the DVT builder to the mode specific default state and clears all the previous directives (preprocessing defines, system variables, libraries, etc.).

You may specify any number of +dvt_init directives inside a build file.

The compatibility mode is enforced until the next +dvt_init directive.

Even if there is no +dvt_init directive specified, there is always an initial reset equivalent to **+dvt_init+dvt**.

3.3 Compatibility Modes

A compatibility mode defines how DVT decides what top files to parse and with what language syntax to parse them.

File Extension to Language Syntax Mapping	File extensions can be mapped either to a specific language syntax or **skipped (that is they will not be parsed)**. Each mode has a default file extension to syntax mapping. You control the extensions mapping by using various directives, like for example +verilog2001ext in vcs.vlogan Compatibility Mode. The +dvt_ext_unmap_all directive clears the syntax mapping, including skipped. This means that all top files will be parsed using the **Language Syntax for Unmapped Extensions**. Note that the syntax for unmapped extensions can be **Skip**, for example in the Default DVT Compatibility Mode, as a result nothing will be compiled.
Language Syntax for Unmapped Extensions	An unmapped top file will be parsed using this syntax or skipped, depending on the compatibility mode. Can be controlled by various directives, like for example +dvt_ext_unmapped_syntax+<syntax> in the Default DVT Compatibility Mode or +v2k in vcs.vlogan Compatibility Mode.
Language Syntax for Included Files	Where relevant, for example for Verilog/ SystemVerilog, the included files are parsed either with the same syntax as the including file, or as specified by the extension mapping. See also each compatibility mode documentation.

Note: A compatibility mode might also introduce specific predefined API, like for example Verilog preprocessing macros.

List of Compatibility Modes

The following compatibility modes are supported:

- **dvt - This is the default mode.**

- vcs.vlogan
- vcs.vhdlan
- ius.irun
- questa.vlog
- questa.vcom

How to Specify the Compatibility Mode

The **+dvt_init+<mode>** directive changes the compatibility mode, where **<mode>** can be any of the above modes.

A **+dvt_init+<mode>** directive:

1. Resets the dvt builder to the mode specific default state.
2. Clears all the previous directives (preprocessing defines, system variables, libraries, etc.).

You can see it as the equivalent of a new tool invocation.

You may specify any number of +dvt_init directives inside a build file.

The compatibility mode is enforced until the next +dvt_init directive.

3.3.1 Default DVT Compatibility Mode

The **+dvt_init+dvt** directive resets the builder to the dvt default state.

File Extension to Language Syntax Mapping

Syntax	Extensions
Verilog 2001	.v, .vh
System Verilog 1800-2012	.vp, .vs, .vsh, .v95, .v95p, .sv, .svh, .svp, .svi, .sva
VHDL 1076-2008	.vhd, .vhdl
e Language 1647-2011	.e
PSS DSL	.pss
C/C++	.c, .h, .cpp, .cc, .cxx
Shared objects (C/C++ libraries)	.so, .a, .o

Language Syntax for Unmapped Extensions: Skip unmapped extensions.

Language Syntax for Included Files: Included files are parsed as specified by the extension mapping.

Mode Specific Directives

Directive	Description
+dvt_ext_map +<syntax> +<ext>	Files with <ext> extension are parsed using the specified <syntax>. See the list below for more details regarding <syntax>.
+dvt_ext_unmap +<ext>	Files with <ext> extension are parsed using the **Language Syntax for Unmapped Extensions**.
+dvt_ext_unmap_all +<syntax>	Set the syntax for **Language Syntax for Unmapped Extensions**. See the list below for more details regarding <syntax>.
+dvt_ext_unmap_all	All files are parsed using the **Language Syntax for Unmapped Extensions**.
+dvt_ext_include_by_ext +by_ext +dvt_ext_include_by_parent +by_parent	by_ext: The included files are parsed using the syntax as specified by directives, that is using by ext syntax (if explicit) or the syntax for unmapped extensions by_parent: Included files are parsed using the syntax that was used for parsing the including file

Specifying a <syntax>

To specify the <syntax> for the directives above, one should use any of the following strings, case-insensitive:

1364-1995, Verilog_95

1364-2001-noconfig, Verilog_2001_noconfig

1364-2001, Verilog_2001

1364-2005, Verilog_2005

VAMS-2.3, Verilog_AMS_23

1800-2005, SystemVerilog_2005

1800-2009, SystemVerilog_2009

1800-2012, SystemVerilog_2012, SystemVerilog

1647-2011, e_2011, e

1076-1987, VHDL_87

1076-1993, VHDL_93

1076.1-1999, VHDL_AMS_99

1076-2000, VHDL_2000

1076-2002, VHDL_2002

1076.1-2007, VHDL_AMS_2007

1076-2008, VHDL_2008, VHDL

PSS

SKIP

The dot (.) for specifying <ext> is optional. For example *+dvt_ext_map +verilog_1364_1995+.svh* and *+dvt_ext_map+verilog_1364_1995+svh* are equivalent.

You can specify more extensions at once, for example *+dvt_ext_map +verilog_1364_1995+.svh+svp*.

When several directives change the syntax of a specific <ext>, the last one wins.

Examples

- I want to parse **.c** and **.v** files as SystemVerilog:

+dvt_init+dvt // By default .c are skipped and .v are parsed with Verilog 2001 syntax
+dvt_ext_map+SystemVerilog_2012+.v+.c // Now .c and .v are parsed with SystemVerilog 2012

Note Every time you re-map an already mapped extension, DVT will warn you. For the example above, you get the following warnings:

.v was previously mapped to Verilog_2001
.c was previously mapped to Skip

- I want the **.vp** files to be parsed with the **Language Syntax for Unmapped Extensions**:

+dvt_init+dvt // By default .vp is parsed with SystemVerilog 2012.
+dvt_ext_unmap+.vp // Because by default the unmapped extensions are skipped, .vp files will be skipped

- I want to change the **Language Syntax for Unmapped Extensions**:

+dvt_init+dvt // By default the unmapped extensions are skipped
+dvt_ext_unmapped_syntax+Verilog_95 // Now unmapped extensions, for example .foo, will be parsed as V

- I want to configure everything from scratch:

+dvt_init+dvt
+dvt_ext_unmap_all // All files are skipped
+dvt_ext_unmapped_syntax+Verilog_95 // Compile all files with unmapped extensions as Verilog 95
+dvt_ext_map+skip+.cpp // Skip .cpp files
+dvt_ext_map+SystemVerilog_95+.sv // Compile .sv as SystemVerilog
+dvt_ext_map+VHDL_1076_1_1999+.vhd // Compile .vhd as VHDL-AMS

- I want to parse everything as SystemVerilog

+dvt_init+dvt
+dvt_ext_unmap_all // All files are skipped
+dvt_ext_unmapped_syntax+SystemVerilog // All files are parsed as SystemVerilog

3.3.2 vcs.vlogan Compatibility Mode

The **+dvt_init+vcs.vlogan** directive resets the builder to the vcs.vlogan default state.

File Extension to Language Syntax Mapping

Syntax	Extensions
C/C++	.c, .cpp, .cc, .cxx
Shared objects (C/C++ libraries)	.so, .a, .o

Language Syntax for Unmapped Extensions: Verilog 1995

Language Syntax for Included Files: Included files are parsed using the syntax that was used for parsing the including file.

Mode Specific Directives

Directive	Description
+verilog1995ext+<ext>	All files with <ext> extension are parsed using the Verilog 1995 syntax.
+verilog2001ext+<ext>	All files with <ext> extension are parsed using the Verilog 2001 syntax.
+systemverilogext+<ext>	All files with <ext> extension are parsed using the SystemVerilog syntax.
-sverilog	Sets the syntax for unmapped extensions to SystemVerilog. This directive has precedence over +v2k.
-upf <upf_file>	Specify a Unified Power Format file to be analyzed.
+v2k	Sets the syntax for unmapped extensions to Verilog 2001.
-extinclude	The included files are parsed using the syntax as specified by directives, that is using by ext syntax (if explicit) or the syntax for unmapped extensions. It overrides the default behavior.
-ntb_opts [uvm / uvm-1.0 / uvm-1.1 / uvm-1.2]	Compiles or references the UVM library. See more details below.
-ntb_opts [rvm / vmm]	If $VCS_HOME is defined, equivalent with:

	+incdir+$VCS_HOME/etc/[rvm / vmm] $VCS_HOME/etc/[rvm / vmm]/vmm.sv If $VCS_HOME is not defined, equivalent with: +incdir+$DVT_VMM_HOME/sv/ $DVT_VMM_HOME/sv/vmm.sv

The dot (.) for specifying <ext> is optional. For example *+verilog1995ext+.svh* and *+verilog1995ext+svh* are equivalent.

You can specify more extensions at once, for example *+verilog1995ext+.svh+svp*.

When several directives change the syntax of a specific <ext>, the last one wins.

ntb_opts [uvm / uvm-1.0 / uvm-1.1 / uvm-1.2]

Typical use-case #1 - single vcs command

Command:

vcs -ntb_opts uvm top1.sv top2.sv [other compilation directives]

Build config:

// The UVM library is compiled from $VCS_UVM_HOME or $VCS_HOME/etc/uvm or $DVT_UVM_HOM
// (in this order of precedence, depending on which environment variables are defined).
+dvt_init+vcs.vlogan -ntb_opts uvm top_file1.sv top_file2.sv [other compilation directives]

Typical use-case #2 - multiple vlogan commands

Commands:

vlogan -ntb_opts uvm

vlogan -ntb_opts uvm top1.sv top2.sv [other compilation directives]

vlogan -ntb_opts uvm top2.sv top2.sv [other compilation directives]

Build config:

// The UVM library is compiled from $VCS_UVM_HOME or $VCS_HOME/etc/uvm or $DVT_UVM_HOM
// (in this order of precedence, depending on which environment variables are defined).
+dvt_init+vcs.vlogan -ntb_opts uvm

// The UVM library is not recompiled. It is "referenced" by adding the relevant incdirs.
+dvt_init+vcs.vlogan -ntb_opts uvm top1.sv top2.sv [other compilation directives]

// The UVM library is not recompiled. It is "referenced" by adding the relevant incdirs.
+dvt_init+vcs.vlogan -ntb_opts uvm top1.sv top2.sv [other compilation directives]

Details

The effect of this directive in a particular invocation depends on the following factors:

- if it's the **first** invocation which specifies -ntb_opts uvm
- if the invocation specifies source files to be compiled (**top files**) in addition to the -ntb_opts uvm flag
- if **$VCS_UVM_HOME / $VCS_HOME** environment variables are set

#	First	Top files	$VCS_UVM_HOME	$VCS_HOME	Equivalent incdir(s)	Equivalent top file
1	Y	---	Y	---	$VCS_UVM_HOME/ src	$VCS_UVM_HOME/ uvm_pkg.sv
2	Y	---	N	Y	$VCS_HOME/ etc/uvm[-version]/src	$VCS_HOME/ etc/uvm[-version]/src/ uvm_pkg.sv
3	Y	---	N	N	$DVT_UVM_HOME/ src	$DVT_UVM_HOME/ src/ uvm_pkg.sv
4	N	N	Y	---	$VCS_UVM_HOME/ src	$VCS_UVM_HOME/ uvm_pkg.sv
5	N	N	N	Y	$VCS_HOME/ etc/uvm[-version]/src	$VCS_HOME/ etc/uvm[-version]/src/ uvm_pkg.sv
6	N	N	N	N	$DVT_UVM_HOME/ src	$DVT_UVM_HOME/ src/ uvm_pkg.sv
7	N	Y	Y	Y	$VCS_HOME/ etc/uvm/ vcs/wrap $VCS_UVM_HOME	
8	N	Y	Y	N	$DVT_HOME/ predefined_projects/ libs/uvm/ vcs/wrap $VCS_UVM_HOME	
9	N	Y	N	Y	$VCS_HOME/ etc/uvm/ vcs/wrap	

					$VCS_HOME/etc/uvm[-version]/src	
10	N	Y	N	N	$DVT_HOME/predefined_projects/libs/uvm/vcs/wrap $DVT_UVM_HOME/src	

Note: In the invocations which "reference" the UVM library (rows 7-10 in the table above) -ntb_opts uvm is equivalent with:

+incdir+<VCS>/wrap
+incdir+<UVM>/src

and any **`include uvm_pkg.sv** will solve to **<VCS>/wrap/uvm_pkg.sv** which contains the following code:

```
`ifndef WRAP_UVM_PKG_SV
`define WRAP_UVM_PKG_SV

`include "uvm_macros.svh"

import uvm_pkg::*;

`endif
```

Note: The second, third, ... **ntb_opts uvm** invocations without top files (rows 4-6 in the table above) will take into account the **dvt_disable_uvm_reuse** directive.

Predefined API

VCS	Defined as preprocessing macro without value.

3.3.3 vcs.vhdlan Compatibility Mode

The **+dvt_init+vcs.vhdlan** directive resets the builder to the vcs.vhdlan default state.

File Extension to Language Syntax Mapping

Syntax	**Extensions**
C/C++	.c, .h, .cpp, .cc, .cxx
Shared objects (C/C++ libraries)	.so, .a, .o

Language Syntax for Unmapped Extensions: VHDL 1076-1993

Mode Specific Directives

Directive	Description
-w <lib> -work <lib>	Compile intro library <lib>
-vhdl87	Sets the syntax for unmapped extensions to VHDL 1076-1987
-upf <upf_file>	Specify a Unified Power Format file to be analyzed.

3.3.4 ius.irun Compatibility Mode

The **+dvt_init+ius.irun** directive resets the builder to the ius.irun default state.

File Extension to Language Syntax Mapping

Syntax	Extensions
Verilog 1995	.v95, .v95p
Verilog AMS 2.3	.vams
Verilog 2001	.v, .vp, .vs
System Verilog 1800-2012	.sv, .svp, .svi, .svh, .vlib, .vcfg
VHDL 1987	.vhd, .vhdl, .vhdp, .vhdlp, .vhcfg
VHDL AMS 1999	.vha, .vhams, .vhms
VHDL 2008	.pslvhdl
e Language 1647-2011	.e
C/c++	.c, .h, .cc, .cpp, .cxx, .pslsc
Shared objects (C/C++ libraries)	.o, .a, .so, .sl
Skipped Files	.pslvlog, .s, .scs, .sp

Language Syntax for Unmapped Extensions: Skip

Language Syntax for Included Files: Included files are parsed using the syntax that was used for parsing the including file.

Mode Specific Directives

Note: in IUS compatibility mode all directives are case-insensitive except for -f / -F

Note: in IUS compatibility mode, top and test files specified using relative paths are solved, in order, as relative to the compilation root, then $SPECMAN_PATH entries

Directive	**Description**
-default_ext <syntax>	Set the **Language Syntax for Unmapped Extensions**. See the list below for more details regarding the <syntax> argument.
-<syntax>_ext [+]<ext>[,<ext>]	Files with <ext> extension will be parsed using the specified <syntax>. If the optional + is specified, the mapping will be added to the default **File Extension to Language Syntax Mapping**. Otherwise, the default mapping of the specified <syntax> is overridden. If you specify the override directive multiple times for the same <syntax>, the default **File Extension to Language Syntax Mapping** will be overridden only the first time. You can specify more extensions at once, comma-separated, for example - *vlog_ext .svh,.svp*. The dot (.) for specifying <ext> is mandatory. The following directives are supported: -a_ext, -amsvhdl_ext, -amsvlog_ext, -as_ext, -c_ext, -cpp_ext, -dynlib_ext, -e_ext, -o_ext, -spice_ext, -sysv_ext, -vhcfg_ext, -vhdl_ext. See the list below for more details regarding <syntax>.
-asext <ext>[,<ext>]	Equivalent to -as_ext +<ext>[,<ext>]
-ccext <ext>	Equivalent to -c_ext +<ext>[,<ext>]
-cxxext <ext>	Equivalent to -cpp_ext +<ext>[,<ext>]
-objext <ext>	Equivalent to -o_ext +<ext>[,<ext>]
-vhdlext <ext>	Equivalent to -vhdl_ext +<ext>[,<ext>]
-vlogext <ext>	Equivalent to -vlog_ext +<ext>[,<ext>]
-sv	All files that would be parsed according to the **File Extension to Language Syntax Mapping** or **Language Syntax for Unmapped Extensions** with a Verilog syntax flavor will be parsed with SystemVerilog 2012 instead. Has precedence over -v1995.
-v1995 -v95	All files that would be parsed according to the **File Extension to Language Syntax Mapping** or **Language Syntax for Unmapped Extensions** with Verilog 2001 will be parsed instead with a reduced keywordset variant of Verilog 2001. The reduced keywordset does not contain the keywords **automatic**, **localparam**, **generate**, **endgenerate**, and **genvar**.
-v200x	All files that would be parsed according to the **File Extension to Language Syntax Mapping** or **Language Syntax for Unmapped Extensions** with a VHDL syntax flavor (but not VHDL AMS) will be parsed with VHDL 2000 instead. Has precedence over -v93.
-v93	All files that would be parsed according to the **File Extension to Language Syntax Mapping** or **Language Syntax for Unmapped Extensions** with a VHDL syntax flavor (but not VHDL AMS) will be parsed with VHDL 93 instead.

-ams	All files that would be parsed with a Verilog / VHDL syntax flavor will be parsed with Verilog AMS 2.3 / VHDL AMS 1999 instead. Has precedence over other syntax specifications.
-amscompilefile "file:<file_path>[...]"	Equivalent with specifying <file_path> as a top file.
-snpath <path>	Equivalent to +dvt_setenv+SPECMAN_PATH=$SPECMAN_PATH:<path>
-sndefine <arg>	Equivalent to +define+<arg>
-ovm -uvm	Load the OVM / UVM library from the IUS installation location. For ovm: **`ncroot`/tools/methodology/OVM/CDNS-2.1.2** or **`ncroot`/tools/ovm** if the first does not exist. For uvm: **`ncroot`/tools/methodology/UVM/CDNS-1.1d/sv** or **`ncroot`/tools/uvm/uvm_lib/uvm_sv** if the first does not exist. **NOTE**: If the path to OVM/UVM cannot be located within the IUS installation, the tool tries to load the library from $OVM_HOME or $DVT_OVM_HOME (resp. $UVM_HOME or $DVT_UVM_HOME).
-ovmhome <path> -uvmhome <path>	If <path> is: **the word "default"**: equivalent with -ovm / -uvm **an existing absolute path or relative path**: load the OVM / UVM library from the specified <path> **an existing subpath of `ncroot`/tools/methodology/OVM/**: load the OVM library from `ncroot`/tools/methodology/OVM/<path> **an existing subpath of `ncroot`/tools/methodology/UVM/**: load the UVM library from `ncroot`/tools/methodology/UVM/<path>/sv Has precedence over -ovm / -uvm.
-makelib **<lib_name>** ... -endlib -makelib /path/to/ **<lib_name>** ... -endlib -makelib /some/path:	Compiles files specified inside a - **makelib** ... - **endlib section** into the **<lib_name>** library. Files in makelib sections are compiled before files in the enclosing invocation. Directives in the makelib section only apply to the makelib section files. Directives in the enclosing invocation apply to all files in the invocation. The - **work** directive is ignored within a makelib section.

<lib_name> ... -endlib	
-lps_1801 <upf_file>	Specify a Unified Power Format file to be analyzed.
-lps_cpf <cpf_file>	Specify a Common Power Format file to be analyzed.

How to specify <syntax> for - *default_ext <syntax>* **and** - *<syntax>_ext* **directives**

Language Syntax	**- default_ext <syntax>**	**- <syntax>_ext**
Verilog 2001	-default_ext verilog	-vlog_ext
Verilog 1995	-default_ext verilog95	N/A
SystemVerilog 2012	-default_ext systemverilog, -default_ext vcnf	-sysv_ext
VHDL 1987	-default_ext vhdl, -default_ext vhcfg	-vhdl_ext
e Language	-default_ext e	-e_ext
VHDL AMS 1999	-default_ext vhdl-ams	-amsvhdl_ext
SKIP	-default_ext verilog-ams, -default_ext psl_vlog, -default_ext psl_vhdl, -default_ext psl_sc, -default_ext c, -default_ext cpp, -default_ext assembly, -default_ext o, -default_ext a, -default_ext so, -default_ext scs	-a_ext, -amsvhdl_ext, -amsvlog_ext, -as_ext, -c_ext, -cpp_ext, -dynlib_ext, -e_ext, -o_ext, -spice_ext, -sysv_ext, -vhcfg_ext, -vhdl_ext

Predefined API

INCA	Defined as preprocessing macro without value.

Examples

- I want to parse **.sv**, **.c** and **.v** files as SystemVerilog:

+dvt_init+ius.irun // By default .c are skipped and .v are parsed with Verilog 2001 syntax
-sysv_ext +.v,.c // Now .c and .v are parsed with SystemVerilog 2012; however, the default extensions mapp

Note Every time you re-map an already mapped extension, DVT will warn you. For the example above, you get the following warnings:

.v was previously mapped to Verilog_2001
.c was previously mapped to Skip

- I want the **.vp** files to be parsed with the **Language Syntax for Unmapped Extensions**:

```
+dvt_init+ius.irun // By default .vp, .v, .vs are parsed with Verilog 2001
-vlog_ext .v, .vs // We override the mapping for Verilog 2001 with only the other two extensions.
// Now .vp is not mapped to any Language Syntax.
// Because by default the unmapped extensions are skipped, .vp files will be skipped
```

- I want to change the **Language Syntax for Unmapped Extensions**:

```
+dvt_init+ius.irun // By default the unmapped extensions are skipped
-default_ext verilog95 // Now unmapped extensions, for example .foo, will be parsed as Verilog 95
```

- I want to parse all Verilog source files and all files with unmapped extensions as SystemVerilog, and all VHDL files as VHDL 2000:

```
+dvt_init+ius.irun
-default_ext systemverilog // All files with unmapped extensions are parsed as SystemVerilog
-sv // All Verilog source files are parsed with SystemVerilog
-v200x // All VHDL source files are parsed with VHDL 2000
```

3.3.5 questa.vlog Compatibility Mode

The **+dvt_init+questa.vlog** directive resets the builder to the questa.vlog default state.

File Extension to Language Syntax Mapping

Syntax	Extensions
System Verilog 1800-2012	.sv, .svp, .svh
C/C++	.c, .cpp, .cc, .cxx

Language Syntax for Unmapped Extensions: Verilog 1995

Language Syntax for Included Files: Included files are parsed using the syntax that was used for parsing the including file.

Mode Specific Directives

Directive	Description
-sv	Parse files with unmapped extensions as SystemVerilog
-sv05compat	Use SystemVerilog 2005 syntax flavor
-sv09compat	Use SystemVerilog 2009 syntax flavor
-sv12compat	Use SystemVerilog 2012 syntax flavor
-vlog95compat	Use Verilog 1995 syntax flavor
-vlog01compat	USe Verilog 2001 syntax flavor

-svfilesuffix=<ext1>,<ext2>... [,<ext2"...]	All files with <ext1>, <ext2>, ... extensions are parsed as SystemVerilog
-uvm	DVT compiles the UVM library, in order of precedence, from: $UVM_HOME, $MTI_HOME/verilog_src/uvm-1.1d, $DVT_UVM_HOME If $MTI_HOME is not defined, it is inferred from the location of the **vlog** executable If -L $MTI_HOME/<uvm_lib> is specified anywhere within the current invocation, then UVM is compiled from $MTI_HOME/verilog_src/<uvm_lib> Whenever compiling UVM from $MTI_HOME, the $MTI_HOME/verilog_src/questa_uvm_pkg-1.2/ is also compiled
-ovm	DVT tries to load the OVM library, in order of precedence from: $OVM_HOME, $DVT_OVM_HOME
-pa_upf <upf_file>	Specify a Unified Power Format file to be analyzed.

Predefined API

QUESTA	Defined as preprocessing macro without value.
MODEL_TECH	Defined as preprocessing macro without value.

3.3.6 questa.vcom Compatibility Mode

The **+dvt_init+questa.vcom** directive resets the builder to the questa.vcom default state.

Language Syntax for Unmapped Extensions: VHDL 1076-2002

Mode Specific Directives

Directive	**Description**
-87	Enable support for VHDL 87
-93	Enable support for VHDL 93
-2002	Enable support for VHDL 2002
-2008	Enable support for VHDL 2008
-pa_upf <upf_file>	Specify a Unified Power Format file to be analyzed.

Predefined API

QUESTA	Defined as preprocessing macro without value.
MODEL_TECH	Defined as preprocessing macro without value.

3.3.7 gcc Compatibility Mode

The **+dvt_init+gcc** directive resets the builder to the gcc default state.

File Extension to Language Syntax Mapping

Syntax	Extensions
C/C++	.c, .i, .ii, .h, .cc, .cp, .cxx, .cpp, .CPP, .c++, .C, .hh, .H, .hp, .hxx, .hpp, .HPP, .h++, .tcc

Language Syntax for Unmapped Extensions: Skip unmapped extensions.

When you compile C/C++ code using gcc rather than the simulator, for each gcc invocation it is recommended to use a +dvt_init+gcc directive followed by the gcc command line arguments.

3.4 Paths

You can specify absolute or relative paths.

Relative paths are resolved relative to the *compilation root*. By default the compilation root is the project directory. However, when using the -F directive to include another *argument file* the compilation root might change (for more details see Including Other Argument Files).

You can use environment variables when specifying paths.

You can use ant-like path patterns [http://ant.apache.org/manual/dirtasks.html#patterns] when specifying *top files* and *incdirs*:

- **?''' matches a single character, for example '''top?.sv** matches **top1.sv** and **topA.sv** but not **top_1.sv**
- ***''' matches a sequence of characters from a file or directory name, for example '''top*/*.sv** matches all files with **.sv** extension from all the directories starting with **top**, **+incdir+/path/to/incdir_*** will set as incdirs all directories from **/path/to/** having the **incdir_** prefix
- **** matches a sequence of directories from a path (recursively), for example ****/*.sv** matches all files with **.sv** extension from the whole project directory tree

3.5 Strings

You may use single quoted (') or double quoted (") strings to group a piece of text so that it will be considered a single directive argument. Example:

+define+FOO='my macro'

To use a quotation mark " in a string, you should escape it using backslash \ like this:

+define+MOO="Hello \"Moo\""

You can also use \" as string delimiter, and any " inside the string defined like this are considered as part of the string, like this:

+define+MOO=\"Hello "Moo"\"

NOTE: When in vcs compatibility mode, the escaping only works in files included with -file (see Including Other Argument Files).

3.6 Comments

The following comment styles are allowed:

single line comment

// single line comment

-- single line comment

/* multi
line
comment */

3.7 Environment Variables

To **define** an environment variable you can use the following syntax:

+dvt_setenv+<NAME>[=VALUE]

Examples:

+dvt_setenv+UVM_HOME=/uvm/uvm-1.1

+dvt_setenv+SPECMAN_PATH=$SPECMAN_PATH:/path/to/my/vips

To **use** the value of an environment variable called ENV_VAR you can use any of the following notations:

- UNIX: $ENV_VAR or ${ENV_VAR}
- Windows: %ENV_VAR%
- Makefile: $(ENV_VAR)
- TCL: $::env(ENV_VAR) or $env(ENV_VAR)

Environment variables, either from the parent shell or explicitly defined, can be used to specify:

- top files
- directives that take a path as an argument, for example +incdir+
- value for defines

When an environment variable is not defined, but it is used, the behavior is as follows:

- for the value of a define directive, the define value is left as-is

- for top files and directives that take a path as an argument, an error is signaled and such top files and directives are ignored
- when used inside the value of an environment variable, it is expanded to the empty string

Environment variables are visible to the DVT parsers, for example when used in the e language **import** statements.

Environment variables are propagated to the external tools launched from DVT, for example by using Run Configurations or External Builders.

When an environment variable is used inside a build file, only the +dvt_setenv+ directives above the usage line are taken into account.

Implementation note

The Dot '.' segment in a path denotes the current directory, which is solved when the path is used, for example to specify a top file or incdir. It is not solved when defining an environment variable. See the example below:

```
-------- In default.build --------
+dvt_setenv+MYVAR=.   // $MYVAR has the value '.' until used to specify a path
$MYVAR/top1.e         // Equivalent with $DVT_PROJECT_LOC/top1.e
-F other/file.f

-------- In other/file.f --------
                      // $MYVAR is used here to specify a path, '.' shall be solved to $DVT_PROJECT_LOC/other/
$MYVAR/top2.e         // Equivalent with $DVT_PROJECT_LOC/other/top2.e
```

Predefined environment variables

Variable	Value
DVT_PROJECT_LOC	Absolute path of the project directory.

3.8 Including Other Argument Files

You can include other argument files with -f, -F or -file:

-f /path/to/argument/file.f

The -f and -file directives are equivalent, except for the vcs compatibility mode. In the vcs compatibility mode Strings [/wiki/Strings] are handled differently when using -f or -file.

The behavior of -F is not consistent across simulators, so DVT treats it according to the compatibility mode:

dvt, vcs — when analyzing the contents of the included argument file, *all* relative paths are resolved as relative to the parent directory of the *argument file*; in other words, this is equivalent to changing the *compilation root* to be the parent directory of the *argument file*

ius — relative paths are solved like in **dvt** and **vcs** modes, except:

- for -v, -y and +incdir directives, if solving relative paths to the *argument file* parent fails, they are solved relative to the *compilation root*
- for -f directives, relative paths are always solved as relative to the *compilation root*

questa the directive is disregarded

3.9 All Build Directives

Directive	Note	Description
-ams		In dvt and vcs.vlogan compatibility modes: enables **Verilog AMS 2.3** extended syntax for Verilog/ SystemVerilog files. In questa.vlog compatibility mode: enables **wreal** extended syntax for Verilog/SystemVerilog files. In ius.irun compatibility mode: all files that would be parsed with a Verilog / VHDL syntax flavor will be parsed with Verilog AMS 2.3 / VHDL AMS 1999 instead. Has precedence over other syntax specifications.
-amscompilefile "file:<file_path>[...]"	ius.irun Compatibility Mode-Specific	Equivalent with specifying <file_path> as a top file.
-asext <ext>[,<ext>]	ius.irun Compatibility Mode-Specific	Equivalent to -as_ext +<ext>[,<ext>]
-CFLAGS -ccflags -ccargs -I -D -L -l -imacros -include	GLOBAL SystemVerilog Only	Gcc arguments used by DVT to configure the CDT builder.
-ccext <ext>	ius.irun Compatibility Mode-Specific	Equivalent to -c_ext +<ext>[,<ext>]
-cxxext <ext>	ius.irun Compatibility Mode-Specific	Equivalent to -cpp_ext +<ext>[,<ext>]
-cuname <compilation_unit_name>		Compile under <compilation_unit_name> package; the directive is enforced until: * another -cuname directive is encountered

		* +dvt_init directive is encountered * end of default.build is encountered
-default_ext <syntax>	ius.irun Compatibility Mode-Specific	Set the **Language Syntax for Unmapped Extensions**. See ius.irun Compatibility Mode for more details regarding the <syntax> argument.
+define +<DEFINE_NAME>=<replacement> -define <DEFINE_NAME>=<replacement>		Define a preprocessing define; the replacement is optional; you may quote the replacement with ' or "; if defined, environment variables are expanded.
+dvt_active_test +<path>	e Language Only	The definition of structs/units/types declared in several Test Files is considered to be the one in the Active Test File.
+dvt_auto_link +<true/false>	GLOBAL	Enable automatic linking of resources located outside the project location. Default: true.
+dvt_auto_snps_vip_macros	GLOBAL	Generate and load in each invocation .dvt/auto_snps_vip_macros.svh file. It contains dummy definitions for macros which are commonly used but not defined or encrypted in Synopsis VIPs.
+dvt_auto_snps_vip_waivers	GLOBAL	Generate and load .dvt/auto_snps_vip_waivers.xml file. It contains waivers for problems commonly encountered when working with encrypted Synopsis VIPs.
+dvt_auto_link_file +<path/to/file>	GLOBAL	Auto-link the specified file.
+dvt_auto_link_root +<alias>=<root_path>	GLOBAL	When Auto-Linking <root_path>/subpath/to/file link it as DVT Auto-Linked/<alias>/subpath/to/file This directive helps to reduce the depth of the virtual filesystem hierarchy under DVT Auto-Linked, because the <root_path> sequence of virtual folders is compacted to <alias> virtual folder. The <root_paths> and <aliases> specified like this must be unique, and therefore only the first occurrence is considered. This is a global directive. Aliases may not be names of directories located directly under the filesystem root (like for example **etc** or **bin**).
+dvt_autoconfig_qsystem_project_revision +<revision_name>	System Verilog and VHDL only	Use <revision_name>.

+dvt_autoconfig_disable_quartus	SystemVerilog and VHDL only	Ignore Quartus project configuration files and fallback to default auto-config.
+dvt_autoconfig_disable_xilinx	SystemVerilog and VHDL only	Disables auto-config from Xilinx ISE/Vivado project. Fallback to default auto-config.
+dvt_autoconfig_ise +<xise_file_name>	SystemVerilog and VHDL only	Auto-config from ISE project using the <xise_file_name> file.
+dvt_autoconfig_quartus_script_location +<script_file_path>	SystemVerilog and VHDL only	For debugging purposes. Use <script_file_path> to analyze Quartus project configuration files.
+dvt_autoconfig_timeout +<timeout_seconds>		Interrupt build if project autoconfiguration takes more than the specified threshold (in seconds). Set 0 to disable timeout. Default: 40 seconds.
+dvt_autoconfig_vivado_fileset +<fileset_name>	SystemVerilog and VHDL only	Auto-config from Vivado project using the <fileset_name> fileset.
+dvt_build_log_file_location +<path_to_existing_directory>	GLOBAL	Specify the location of the internal builder log file. Default: ./ (Project location).
+dvt_build_log_to_console +<true/false>	GLOBAL	Enable/disable internal builder logging to console. Default: true.
+dvt_build_log_to_file +<true/false>	GLOBAL	Enable/disable internal builder logging to file. Default: true.
+dvt_cpf +<cpf_file>	SystemVerilog and VHDL Only	Specify a CPF file for compilation.
+dvt_compilation_root +</path/to/compilation/root>		Specify the compilation root. Relative paths specified in default.build will be resolved as relative to this location, except for the special cases that rise when Including Other Argument Files.
+dvt_db_location +<path>	GLOBAL	Save the project database files under <path>/dvt_db/<project_name> directory. The <path>/dvt_db/<project_name> directory will be created if needed and may be overwritten at each full/incremental project build. In certain situations (for example if write access for <path>/dvt_db/<project_name> is denied or the directory is in use by another DVT instance) DVT falls back to the default location: <dvt_workspace>/.metadata/.plugins/org.eclipse.core.resources/.projects/<project_name>.
+dvt_disable_parallel_lex_parse	GLOBAL SystemVerilog Only	Disable lexing-parsing parallelization.
+dvt_disable_preprocessor_optimize +<true/false>	GLOBAL SystemVerilog Only	Disable DVT preprocessing optimizations. Default: false.

+dvt_disable_rtl_checks	GLOBAL SystemVerilog and VHDL Only	Disable the RTL specific semantic checks: SENSITIVITY_MISSING/SENSITIVITY_UNUSED and SIGNAL_NEVER_READ/ SIGNAL_NEVER_WRITTEN/ SIGNAL_NEVER_USED. By default the checks are enabled.
+dvt_disable_checks +<check_id1> +<...>	DEPRECATED GLOBAL SystemVerilog and VHDL Only	Disable a set of semantic checks by ID. For example: +dvt_disable_checks +UNDECLARED_IDENTIFIER Possible check IDs: UNDECLARED_IDENTIFIER (SystemVerilog and VHDL), CONTINUOUS_ASSIGNS (SystemVerilog), INSTANCES_AND_PORT_CONNECTIONS (SystemVerilog and VHDL), REDUNDANT_OTHERS_CHOICE (VHDL), MISSING_OTHERS_CHOICE (VHDL). By default no checks are disabled.
+dvt_disable_uvm_reuse	GLOBAL SystemVerilog Only	Compile the UVM package in every invocation which specifies -uvm / -ntb_opts uvm. By default UVM is compiled only in the first invocation, and subsequent -uvm / -ntb_opts uvm only provide the UVM incdir. Default value: false
+dvt_disable_naming_convention_checks	GLOBAL e Language and VHDL Only	Disable naming convention checks. Default value: false
+dvt_e_enable_non_standard_checks +<true/false>	GLOBAL e Language Only	Enable/disable non-standard syntax and semantic checks. Default: false.
+dvt_e_sn_which_emulation	GLOBAL e Language Only	Use the following set of search paths to locate VIPs\n instead of sn_which.sh: / <IUS Install Location>/specman/linux/ <IUS Install Location>/specman/src/ <IUS Install Location>/specman/docs/ <IUS Install Location>/specman/tcl/specman/ <IUS Install Location>/specman/linux/

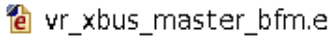
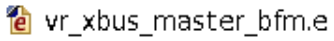
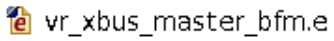
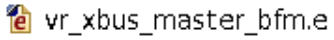
		<IUS Install Location>/specman/src/ <IUS Install Location>/specman/docs/ <IUS Install Location>/specman/tcl/specman/ <IUS Install Location>/specman/erm_lib/ <IUS Install Location>/specman/sn_lib/ <IUS Install Location>/specman/packages/ <IUS Install Location>/specman/uvm/uvm_lib/ <IUS Install Location>/specman/ovm/ovm_lib/ <IUS Install Location>/specman/erm_lib/ <IUS Install Location>/specman/sn_lib/ <IUS Install Location>/specman/packages/ <IUS Install Location>/specman/uvm/uvm_lib/ <IUS Install Location>/specman/ovm/ovm_lib/
+dvt_e_macro_strict_exp_checking +<true/false>	GLOBAL e Language Only	If true, the <exp> match expression will match only valid expressions. If false, <exp> is equivalent with the <any> match expression that matches any non-empty sequence of characters. Default: false.
+dvt_e_macro_exp_back_tracking +<true/false>	GLOBAL e Language Only	If true, the parser will reject a user defined expression match if the result of the macro reparse is not a valid expression. Default: true.
+dvt_e_sn_extract_defines +<true/false>	GLOBAL e Language Only	Automatically extract and define the Specman version defines. Default: true.
+dvt_e_sn_which_add +<vip1>+<vip2> +... +dvt_e_sn_which_clear	GLOBAL e Language Only	By default, the following VIPs are located using sn_which.sh, and their locations are added to the $SPECMAN_PATH: evc_util, vr_ad, ovm_e, uvm_e. When an irun installation newer than 13.10 is detected, only evc_util, vr_ad, and uvm_e are added. Use +dvt_sn_which_add+<vip1>+<vip2>+... to add to this list and +dvt_sn_which_clear to clear it.
+dvt_enable_non_top_instances_check +<true/false>	GLOBAL e Language Only	Enables analysis of all modules/instances when using -top/+nctop+ directive. The modules/instances that are not children of the top s are not resolved (semantic

	SystemVerilog Language Only	errors/warnings are not reported) unless this variable is set to true. Default: false.
+dvt_enable_unknown_directive_warnings +<true/false>	GLOBAL	Trigger warnings for unknown build directives. Default: false.
+dvt_ext_map +<syntax> +<ext>		Files with <ext> extension are parsed using the specified <syntax>. See Default DVT Compatibility Mode for more details regarding <syntax>.
+dvt_ext_unmap +<ext>		Files with <ext> extension are parsed using the **Language Syntax for Unmapped Extensions**.
+dvt_ext_unmap_all		All files are parsed using the **Language Syntax for Unmapped Extensions**.
+dvt_ext_unmapped_syntax +<syntax>		Set the **Language Syntax for Unmapped Extensions**. See Default DVT Compatibility Mode for more details regarding <syntax>.
+dvt_extract_comment_above +<true/false>	GLOBAL SystemVerilog and VHDL Only	Extract comments above elements. Default: true.
+dvt_extract_comment_above_max_empty_lines +<number_of_lines>	GLOBAL SystemVerilog and VHDL Only	Extract comment if located at no more than specified number of empty lines above element declaration. Default: 1.
+dvt_extract_comment_bol +<true/false>	GLOBAL SystemVerilog and VHDL Only	Extract /** begin comment delimiter comments. Default: true.
+dvt_extract_comment_header +<true/false>	GLOBAL SystemVerilog and VHDL Only	Extract file header comments and associate them with the first element in file (module, entity etc.). Default: false.
+dvt_extract_comment_inline +<true/false>	GLOBAL SystemVerilog and VHDL Only	Exctract comments inline with elements. Default: true.
+dvt_extract_comment_ml +<true/false>	GLOBAL SystemVerilog and VHDL Only	Extract /* multi line comments. Default: true.
+dvt_extract_comment_sl +<true/false>	GLOBAL SystemVerilog and VHDL Only	Extract // single line comments. Default: true.

+dvt_file_compile_timeout +<timeout>	GLOBAL	During full compilation, skip parsing a file if it takes more than the specified threshold (in seconds). Set 0 to disable timeout. Default: 40 seconds.
+dvt_file_substitute +<file_path>=<substitute_file_path>	GLOBAL SystemVerilog and VHDL Only	During compilation, the <file_path> file will be substituted with the <substitute_file_path> file.
+dvt_full_compile_checks +<scope>	GLOBAL SystemVerilog and VHDL Only	In order to speed-up full compilation, you may chose to fully check only a relevant subset of your source code. This directive controls the scope of the full build checks: +dvt_full_compile_checks+FULL - all of the code is checked +dvt_full_compile_checks+LIBS+lib1+lib2 - only the specified libraries are checked, some basic checks are still performed for the rest of the code +dvt_full_compile_checks+NOT_LIBS+lib1+lib2 - all of the code is checked except for the specified libraries, where only some basic checks are performed +dvt_full_compile_checks+PKGS +lib1::pkg1+lib2::pkg2 - only the specified packages are checked, some basic checks are still performed for the rest of the code +dvt_full_compile_checks+NOT_PKGS +lib1::pkg1+lib2::pkg2 - all of the code is checked except for the specified packages, where only some basic checks are performed +dvt_full_compile_checks+OFF - only some basic checks are performed Default: FULL
+dvt_gcc+</path/to/gcc_executable>	GLOBAL	Specify location of GNU C compiler executable.
+dvt_gcc_link_system_headers	GLOBAL	Enable automatic linking of C system headers. Default: false.
+dvt_gcc_timeout +<timeout>	GLOBAL	Timeout in seconds when running GCC. Set 0 to disable timeout. Default: 40 seconds.
+dvt_hdtv	GLOBAL	Hide duplicates from Types, Checks and Coverage Views.

	SystemVerilog Only	
+dvt_incremental_compile_checks +<scope>	GLOBAL SystemVerilog and VHDL Only	In order to speed-up incremental compilation, you may chose to turn off advanced checking. +dvt_incremental_compile_checks+OFF - only some basic checks are performed all over the code +dvt_incremental_compile_checks+ON - checks are performed in all affected areas of your code that are also checked at full build (see +dvt_full_compile_checks) Note: if +dvt_full_compile_checks is set to OFF this flag has no effect. Default: ON
+dvt_incremental_compile_max_lines +<max_lines_number>	GLOBAL SystemVerilog and VHDL Only	Files with more than max lines will not be incrementally compiled. Set 0 for infinite limit. Default: 7000.
+dvt_incremental_compile_timeout +<timeout>	GLOBAL SystemVerilog and VHDL Only	During incremental compilation, skip the file if parsing or semantic checking takes more than the specified threshold (in seconds). Set 0 to disable timeout. Default: 4 seconds.
+dvt_init[+<compat_mode>]		Equivalent of a new invocation, resets all directives except for the GLOBAL ones. See Compatibility Modes for a detailed description.
+dvt_init_auto[+<compat_mode>]		Automatically identify and compile all the source files in the compilation root. The compilation root defaults to the project directory and can be changed using +dvt_compilation_root+ directive. Available compatibility modes are: dvt, ius.irun, vcs.vlogan and vcs.vhdlan. If a compatibility mode is not specified, it defaults to dvt. See Auto-config for a detailed description.
+dvt_init_xilinx[+<lib1> +<lib2+...>]		Compile the specified libraries from the $DVT_XILINX_HOME installation. Similar with +dvt_init, it is equivalent with a new invocation. The available libraries are UNISIM, UNIMACRO, UNIFAST, XILINXCORELIB, CPLD, SIMPRIM, SECUREIP_VER, UNISIMS_VER, UNIFAST_VER, UNIMACRO_VER, SIMPRIMS_VER, XILINXCORELIB_VER, UNI9000_VER, CPLD_VER, RETARGET. See Xilinx Libraries Compilation for a detailed description.
+dvt_init_uvvm		Compile the UVVM sources. Similar with +dvt_init, it is equivalent with a new invocation.

+dvt_init_uuvm_vvc		Compile the UVVM_VVC sources. It should be used for every VVC. Similar with +dvt_init, it is equivalent with a new invocation.
+dvt_init_osvvm		Compile the OSVVM sources. Similar with +dvt_init, it is equivalent with a new invocation.
+dvt_max_nof_threads +<num_threads>	GLOBAL	Configure the maximum number of threads to use during different phases of intensive computation (e.g. semantic checking, etc.). Default: 8
+dvt_path_pattern_timeout +<timeout>		Timeout in seconds when scanning path patterns (like for example /**/*.v). Default: 5.
+dvt_pf_debug	SystemVerilog and VHDL Only	Print debug information during power format build phase.
+dvt_prepend_init	GLOBAL	You can use a +dvt_prepend_init section to specify directives like +define, +dvt_setenv, +incdir etc. that are prepended to all +dvt_init sections. All directives between +dvt_prepend_init and the next +dvt_init will be "copied" in all subsequent +dvt_init sections.
+dvt_preprocess_translate_pragmas +<pragma1> +<pragma2>+...	SystemVerilog and VHDL Only	Instructs DVT to skip analyzing the code between pragmas such as // <pragma> translate_off // <pragma> translate_on You can specify any number of pragmas as arguments to this directive, separated by '+' like for example +dvt_preprocess_translate_pragmas+pragma+synopsys +synthesis ' *Note:*' In VHDL, the code background will be highlighted, but it will still be analyzed.
+dvt_pss_cpp	Add $PSS_CPP_HOME/include/pss.h as topfile and $PSS_CPP_HOME/include as C include dir. Falls back to $DVT_PSS_CPP_HOME if $PSS_CPP_HOME is not defined.	

+dvt_semantic_checks_timeout +<timeout>	GLOBAL SystemVerilog and VHDL Only	Popup semantic checking dialog asking to continue or stop when full compilation semantic checking takes more than the specified timeout (in seconds). Default value: 30.
+dvt_setenv +<NAME>[=VALUE]		Define an environment variable. Its value is visible for subsequent directives and during parsing.
+dvt_skip_compile +<simple_pattern>		Instructs DVT to skip analyzing the files whose absolute path **matches** the specified <simple_pattern>. In a simple pattern you can use wildcards such as '*' (any string) and '?' (any character). Such skipped files are decorated distinctively in the Navigator View:  .
+dvt_skip_compile +not +<simple_pattern>		Instructs DVT to skip analyzing the files whose absolute path **does not match** the specified <simple_pattern>. In a simple pattern you can use wildcards such as '*' (any string) and '?' (any character). Such skipped files are decorated distinctively in the Navigator View:  .
+dvt_skip_compile +regex +<regex_pattern>		Instructs DVT to skip analyzing the files whose absolute path **matches** the specified <regex_pattern>. Such skipped files are decorated distinctively in the Navigator View:  .
+dvt_skip_compile +regex+not +<regex_pattern>		Instructs DVT to skip analyzing the files whose absolute path **does not match** the specified <regex_pattern>. Such skipped files are decorated distinctively in the Navigator View:  .
+dvt_skip_ext +<ext>		Do not parse top files with <ext> extension. The dot (.) for specifying <ext> is optional. For example *+dvt_skip_ext+.gv* and *+dvt_skip_ext+gv* are equivalent.
+dvt_skip_protected_code +true	GLOBAL SystemVerilog Only	Do not analyze code encolsed in `protect ... `endprotect pragmas.
+dvt_systemc	Add $SYSTEMC_HOME/ src/systemc.h as	

	topfile and $SYSTEMC_HOME/src as C include dir. Falls back to $DVT_SYSTEMC_HOME if $SYSTEMC_HOME is not defined.	
+dvt_test+<path>	e Language Only	Specify a top file and mark it as test. For example, the e Language test files have a special status, see e Language Test Files.
+dvt_upf +<upf_file>	SystemVerilog and VHDL Only	Specify a UPF file for compilation.
+dvt_undefine +<DEFINE_NAME>	SystemVerilog Only	Undefines <DEFINE_NAME> preprocessing symbol. Equivalent with `undef <DEFINE_NAME>. The +dvt_undefine directives are applied on top of all other specified +defines. Ordering relative to other specified top files is relevant.
+dvt_wreal	SystemVerilog Only	Enables **wreal** extended syntax for Verilog/ SystemVerilog files.
-extinclude	vcs.vlogan Compatibility Mode-Specific	The included files are parsed using the syntax as specified by directives, that is using by ext syntax (if explicit) or the syntax for unmapped extensions. It overrides the default behavior.
+incdir+<path> -incdir <path>	SystemVerilog Only	Indicate search directories for files included with `include preprocessing directive.
+libext+<ext1> +<ext2>+<extN> -libext <ext1>,<ext2>,<extN>	SystemVerilog Only -libext is **ius.irun** mode specific	Specify accepted extensions for files in the library directories. Extensions must include the '.' dot character. In **ius.irun** compatibility mode, either plus '+' or comma ',' may be used as extension list separator for either directive. **Note:** there are no default extensions, .v and .sv don't have a special status.
-libmap <path>		Specify the Verilog library map file.
+librescan -librescan		When DVT finds an unresolved module reference in a library file or directory, it will scan for the unresolved reference starting from the first specified library; by default (librescan not specified) it starts scanning from the library that introduced the unresolved reference and continues using the specified libraries order.

-lps_1801 <upf_file> -lps_cpf <upf_file>	ius.irun Compatibility Mode-Specific SystemVerilog and VHDL Only	Specify a UPF or CPF power format file for compilation.
-makelib **<lib_name>** -makelib /path/to/ **<lib_name>** -makelib /some/ path: **<lib_name>** ... -endlib	ius.irun Compatibility Mode-Specific	Compiles files specified inside a - **makelib** ... - **endlib section** into the **<lib_name>** library. Files in makelib sections are compiled before files in the enclosing invocation. Directives in the makelib section only apply to the makelib section files. Directives in the enclosing invocation apply to all files in the invocation. The - **work** directive is ignored within a makelib section.
+nctop +<config_name>	GLOBAL SystemVerilog Only	Specify top configuration name.
-objext <ext>	ius.irun Compatibility Mode-Specific	Equivalent to -o_ext +<ext>[,<ext>]
-ovm -uvm		In **dvt and vcs.vlogan Compatibility Modes** it is equivalent with **+incdir+/path/to/xvm/src** **/path/to/xvm/src/xvm_pkg.sv** where **/path/to/xvm** is **$XVM_HOME** or **$DVT_XVM_HOME** if **$XVM_HOME** is not defined, where XVM is a shorthand for OVM / UVM. In the **ius.irun and questa.vlo Compatibility Modes /path/to/xvm** is automatically located within the IUS resp. Questa installation dirs. See ius.irun Compatibility Mode and questa.vlog Compatibility Mode for more details.
-ovmhome -uvmhome	ius.irun Compatibility Mode-Specific	Load the OVM / UVM library from the specified <path>. See ius.irun Compatibility Mode for more details.
-pa_upf <upf_file>	questa.vlog and questa.vcom	Specify a UPF power format file for compilation.

	Compatibility Mode-Specific SystemVerilog and VHDL Only	
-realport	vcs.vlogan Compatibility Mode-Specific SystemVerilog Only	Enables **wreal** extended syntax for Verilog/ SystemVerilog files.
-sndefine <arg>	ius.irun Compatibility Mode-Specific	Equivalent to +define+<arg>
-snpath <path>	ius.irun Compatibility Mode-Specific	Equivalent to +dvt_setenv+SPECMAN_PATH= $SPECMAN_PATH:<path>
-sv	ius.irun Compatibility Mode-Specific	All files that would be parsed according to the **File Extension to Language Syntax Mapping** or **Language Syntax for Unmapped Extensions** with a Verilog syntax flavor will be parsed with SystemVerilog 2012 instead. Has precedence over -v1995.
-sv_lib <file_path>	GLOBAL	Specify a shared object C/C++ library. The <file_path> should be specified without the **.so** extension. Provided that the library contains debug info, DVT will Auto-Link the C/C++ source files from which the library was compiled. The .so extension is automatically appended to the specified path. **Implementation note:** If <file_path>.so is not found, the tool will try to locate and load <file_path> instead.
-sv_liblist <file_path>	GLOBAL	Specify a shared object bootstrap file. The file contains a list of shared object C/C++ library paths, one per line. For each library in the bootstrap file, provided that the library contains debug info, DVT will Auto-Link the C/C++ source files from which the library was compiled. The .so extension is automatically appended to the paths specified in the bootstrap file.
-sv root <directory_path>	GLOBAL	The root directory path is prepended to any relative path that will be specified following this directive, using either -sv_lib or -sv_liblist or inside the shared object bootstrap file.
-sverilog	vcs.vlogan Compatibility Mode-Specific	Sets the syntax for unmapped extensions to SystemVerilog. This directive has precedence over +v2k.

-<syntax>_ext [+]<ext>[,<ext>]	ius.irun Compatibility Mode-Specific	Files with <ext> extension will be parsed using the specified <syntax>. If the optional + is specified, the mapping will be added to the default **File Extension to Language Syntax Mapping**. Otherwise, the default mapping of the specified <syntax> is overridden. If you specify the override directive multiple times for the same <syntax>, the default **File Extension to Language Syntax Mapping** will be overridden only the first time. You can specify more extensions at once, comma-separated, for example - *vlog_ext .svh,.svp*. The dot (.) for specifying <ext> is mandatory. The following directives are supported: -a_ext, -amsvhdl_ext, -amsvlog_ext, -as_ext, -c_ext, -cpp_ext, -dynlib_ext, -e_ext, -o_ext, -spice_ext, -sysv_ext, -vhcfg_ext, -vhdl_ext. See ius.irun Compatibility Mode for more details regarding <syntax>.
+systemverilogext+<ext>	vcs.vlogan Compatibility Mode-Specific	All files with <ext> extension are parsed using the SystemVerilog syntax.
-top <config_name>	GLOBAL SystemVerilog Only	Specify top configuration name.
+UVM_TESTNAME	GLOBAL	The name of the UVM test which will be automatically created under uvm_root.
-upf <upf_file>	vcs.vlogan and vcs.vhdlan Compatibility Mode-Specific SystemVerilog and VHDL Only	Specify a UPF power format file for compilation.
-v <path>		Specify a Verilog library file.
-v1995 -v95	ius.irun Compatibility Mode-Specific	All files that would be parsed according to the **File Extension to Language Syntax Mapping** or **Language Syntax for Unmapped Extensions** with Verilog 2001 will be parsed instead with a reduced keywordset variant of Verilog 2001. The reduced keywordset does not contain the keywords **automatic**, **localparam**, **generate**, **endgenerate**, and **genvar**.
-v200x	ius.irun Compatibility Mode-Specific	All files that would be parsed according to the **File Extension to Language Syntax Mapping** or **Language Syntax for Unmapped Extensions** with a VHDL

		syntax flavor (but not VHDL AMS) will be parsed with VHDL 2000 instead. Has precedence over -v93.
-v93	ius.irun Compatibility Mode-Specific	All files that would be parsed according to the **File Extension to Language Syntax Mapping** or **Language Syntax for Unmapped Extensions** with a VHDL syntax flavor (but not VHDL AMS) will be parsed with VHDL 93 instead.
+v2k	vcs.vlogan Compatibility Mode-Specific	Sets the syntax for unmapped extensions to Verilog 2001.
+verilog1995ext +<ext>	vcs.vlogan Compatibility Mode-Specific	All files with <ext> extension are parsed using the Verilog 1995 syntax.
+verilog2001ext +<ext>	vcs.vlogan Compatibility Mode-Specific	All files with <ext> extension are parsed using the Verilog 2001 syntax.
-vhdl87	vcs.vhdlan Compatibility Mode-Specific	Sets the syntax for unmapped extensions to VHDL 1076-1987.
-vhdlext <ext>	ius.irun Compatibility Mode-Specific	Equivalent to -vhdl_ext +<ext>[,<ext>]
-vlogext <ext>	ius.irun Compatibility Mode-Specific	Equivalent to -vlog_ext +<ext>[,<ext>]
-w <lib> -work <lib>	vcs.vhdlan Compatibility Mode-Specific	Compile intro library <lib>.
-work <lib>		Compile into library <lib>.
-wreal <res_func>	vcs.vlogan Compatibility Mode-Specific SystemVerilog Only	Enables **wreal** extended syntax for Verilog/ SystemVerilog files.
-y <path>		Specify a Verilog library directory.
-pkgsearch <lib>	ius.irun Compatibility Mode-Specific SystemVerilog Only	Specify the library search order for Verilog packages. You can specify multiple libraries by using this option multiple times.

-liblist <lib1>[+<lib2>+...]	vcs.vlogan Compatibility Mode-Specific SystemVerilog Only	Specify the library search order for Verilog packages.

NOTE: GLOBAL directives are effective for all invocations. They are not reset by +dvt_init directives.

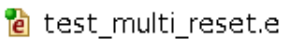
3.10 e Language Test Files

Test Files are parsed independently on top of the Top Files.

A Test File can be specified using the following directive:

+dvt_test+/path/to/test_file.e

All Test Files and the files imported by Test Files, but not part of Top Files and the files imported by Top Files, are decorated with a **green bullet**:



3.11 e Language SPECMAN_PATH

The SPECMAN_PATH environment variable is used by the e Language parser to find imported files.

It can be defined or altered by using the +dvt_setenv+ directive.

When relative paths are provided, they are automatically transformed into paths relative to the compilation root directory. By default the compilation root is the project directory. For more details see Paths.

Examples:

Define/overwrite shell-inherited value	+dvt_setenv+SPECMAN_PATH=/my/specman/path1:/my/specman/path1
Append to shell-inherited value	+dvt_setenv+SPECMAN_PATH=$SPECMAN_PATH:/my/specman/path
Prepend to shell-inherited value	+dvt_setenv+SPECMAN_PATH=/my/specman/path:$SPECMAN_PATH
Append <project_root>/e directory to the SPECMAN_PATH	+dvt_setenv+SPECMAN_PATH=$SPECMAN_PATH:e

Backward-compatibility note: starting with DVT version 3.2, if a .build file is used to configure the project, the .edt_specman_path file is ignored.

Adding VIPs to $SPECMAN_PATH using sn_which.sh

You can specify a list of VIPs that should automatically be located by DVT using **sn_which.sh** and added to the $SPECMAN_PATH (if available in the console where DVT was started). The Specman version defines (e.g define SPECMAN_VERSION_###) are also detected using the same mechanism.

By default, the following VIPs are located and added automatically: evc_util, vr_ad, ovm_e, uvm_e

3.12 SystemVerilog OVM or UVM Library Compilation

Typically the OVM or UVM libraries are compiled using directives like:

+incdir+/path/to/xvm/src
/path/to/xvm/src/xvm_pkg.sv

Using the -ovm or -uvm directives

You can use the - *ovm* or - "-uvm" directives to compile the OVM or UVM libraries. They are shortcuts for the explicit directives above.

For - "-uvm":

1. If the $UVM_HOME system variable is specified, use that library.
2. If the $DVT_UVM_HOME system variable is specified, use that library. If you launch DVT using one of the utility scripts that ship with DVT, $DVT_UVM_HOME is set by default to the most recent uvm library in $DVT_PREDEFINED_PROJECTS/libs.

The - "-ovm" directive works in a similar way.

NOTE: At any time you can set a system variable using for example:

+dvt_setenv+UVM_HOME=/path/to/xvm

3.13 Xilinx Libraries Compilation

In order to compile Xilinx libraries:

- specify the required libraries using the +dvt_init_xilinx directive
- specify the Xilinx installation path (ISE or Vivado), unless $DVT_XILINX_HOME system variable is set

For example:

+dvt_init_xilinx+UNISIM+UNIMACRO_VER

+dvt_setenv+DVT_XILINX_HOME=/apps/xilinx/Vivado/2014.2/

The available libraries are UNISIM, UNIMACRO, UNIFAST, XILINXCORELIB, CPLD, SIMPRIM, SECUREIP_VER, UNISIMS_VER, UNIFAST_VER, UNIMACRO_VER, SIMPRIMS_VER, XILINXCORELIB_VER, UNI9000_VER, CPLD_VER, RETARGET.

3.14 Intel(Altera) Quartus Libraries Compilation

In order to compile Intel(Altera) Quartus libraries:

- specify the required libraries using the +dvt_init_altera directive
- specify the Quartus installation path, unless $QUARTUS_ROOTDIR system variable is set

For example:

+dvt_init_altera+ALTERA+ALTERA_MF
+dvt_setenv+QUARTUS_ROOTDIR=/apps/altera/13.0sp1/quartus

The available VHDL libraries are ALTERA_MF, ALTERA, ALTERA_LNSIM, LPM, MAX, MAXII, MAXV, STRATIX, STRATIXII, STRATIXIIGX, HARDCOPYII, HARDCOPYIII, HARDCOPYIV, CYCLONE, CYCLONEII, CYCLONEIII, CYCLONEIIILS, SGATE, STRATIXGX, ALTGXB, STRATIXGX_GXB, STRATIXIIGX_HSSI, ARRIAGX_HSSI, ARRIAII, ARRIAII_HSSI, ARRIAII_PCIE_HIP, ARRIAIIGZ, ARRIAIIGZ_HSSI, ARRIAIIGZ_PCIE_HIP, ARRIAGX, STRATIXIII, STRATIXIV, STRATIXIV_HSSI, STRATIXIV_PCIE_HIP, CYCLONEIV, CYCLONEIV_HSSI, CYCLONEIV_PCIE_HIP, CYCLONEIVE, HARDCOPYIV_HSSI, HARDCOPYIV_PCIE_HIP, STRATIXV, STRATIXV_HSSI, STRATIXV_PCIE_HIP, ARRIAVGZ, ARRIAVGZ_HSSI, ARRIAVGZ_PCIE_HIP, ARRIAV, CYCLONEV.

The available Verilog libraries are ALTERA_MF_VER, ALTERA_VER, ALTERA_LNSIM_VER, LPM_VER, MAX_VER, MAXII_VER, MAXV_VER, STRATIX_VER, STRATIXII_VER, STRATIXIIGX_VER, ARRIAGX_VER, HARDCOPYII_VER, HARDCOPYIII_VER, HARDCOPYIV_VER, CYCLONE_VER, CYCLONEII_VER, CYCLONEIII_VER, CYCLONEIIILS_VER, SGATE_VER, STRATIXGX_VER, ALTGXB_VER, STRATIXGX_GXB_VER, STRATIXIIGX_HSSI_VER, ARRIAGX_HSSI_VER, ARRIAII_VER, ARRIAII_HSSI_VER, ARRIAII_PCIE_HIP_VER, ARRIAIIGZ_VER, ARRIAIIGZ_HSSI_VER, ARRIAIIGZ_PCIE_HIP_VER, STRATIXIII_VER, STRATIXIV_VER, STRATIXIV_HSSI_VER, STRATIXIV_PCIE_HIP_VER, STRATIXV_VER, STRATIXV_HSSI_VER, STRATIXV_PCIE_HIP_VER, ARRIAVGZ_VER, ARRIAVGZ_HSSI_VER, ARRIAVGZ_PCIE_HIP_VER, ARRIAV_VER, ARRIAV_HSSI_VER, ARRIAV_PCIE_HIP_VER, CYCLONEV_VER, CYCLONEV_HSSI_VER, CYCLONEV_PCIE_HIP_VER, CYCLONEIV_VER, CYCLONEIV_HSSI_VER, CYCLONEIV_PCIE_HIP_VER, CYCLONEIVE_VER, HARDCOPYIV_HSSI_VER, HARDCOPYIV_PCIE_HIP_VER.

Chapter 4. Compile Waivers

You can use waivers to change the severity (promote/demote) or disable the problems reported by DVT during compilation.

Promote	Warning -> Error
Demote	Error -> Warning
Disable	Hide
Restore	Restore a waived problem to its default (originally reported) severity

Waivers are applied in order. Multiple waivers may be applied to the same problem.

Compile Waivers Examples

Use Case	**Solution**
I want to hide all problems from a library that I do not control.	<waiver name="Disable all from library folder." severity="DISABLED"><match path="/path/to/library/*"/></waiver>
I want to hide some errors that I don't care about.	<waiver name="Disable all that match a specific message pattern." severity="DISABLED"><match message="*some message pattern*"/></waiver>
I want to turn NON_STANDARD warnings into errors.	<waiver name="Promote NON_STANDARD warnings to errors." severity="ERROR"><match message="NON_STANDARD*"/></waiver>
I want to see only the problems reported on files in a specific directory.	Create a waiver to disable all problems: <waiver name="Disable all" severity="DISABLED"><match path="*"/></waiver> After it, create a waiver to restore the severity of the problems inside a specific path: <waiver name="Restore my problems" severity="DEFAULT"><match path="/my/path"/></waiver>
I want to hide some DVT false alarms until the issue causing them is fixed.	Use a message based and/or path based waiver.

Each problem message is in the form:

<CHECK_ID>: <Failure Details>

or, for Non-top files:

_<CHECK_ID>: <Failure Details>

See Semantic Checks for a complete list of all checks and their identifiers.

This allows you to change the severity by check id using a waiver like:

<waiver name="Disable all <CHECK_ID>" severity="DISABLED"><match message="*<CHECK_ID>:*"/

Compile Waivers File Syntax (XML)

```
<!--
  XML file header; required.
-->
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE waivers PUBLIC "-//DVT//waivers" "waivers.dtd" >

<!--
  Root tag; required.
  Version attribute is required.
  The latest syntax version, illustrated by this example, is version 1.
-->
<waivers version="1">
  <!--
    You can include waivers from other waiver files. The syntax
    of the included files is the same as this. You may use environment
    variables in the path to included waiver files.
  -->
  <include path="$COMMON/path/to/included_waivers.xml"/>
  <!--
    The root tag must contain at least one waiver.
    The waiver tag must specify the NEW severity of the problems
    waived by this waiver; it can be one of ERROR, WARNING, DISABLED or DEFAULT
  -->
  <waiver name="Optional short name of the waiver" severity="DISABLED">
     <description>An optional verbose description of the waiver.</description>
     <!--
        Each waiver must contain at least one match tag.
        Each match tag must specify a message pattern, a path pattern or both.
        A match tag matches a problem if ALL specified patterns match.
        The waiver will waive a problem if ANY of the match tags matches.
        NOTE:
           path pattern is NOT matched against the project relative path
           patterns may contain * or ? wildcards
     -->
     <match
        message="pattern to match against the problem's message"
        path="/pattern/to/match/against/the/problems/absolute/path"/>
     <match message="*message pattern*" path="/path/pattern*"/>
     <match message="*only by message*"/>
```

```
    <match path="/only/by/path*"/>
  </waiver>
  <!-- Further example waivers -->
  <waiver name="Disable by message" severity="DISABLED">
    <description>
      This waiver disables all problems containing 'JUST_A_FOO_PROBLEM'
      inside their message.
    </description>
    <match message="*JUST_A_FOO_PROBLEM*"/>
  </waiver>
  <waiver name="Demote by path" severity="WARNING">
    <description>
      This waiver turns into warnings all problems reported
      under '/path/to/foo'
    </description>
    <match path="/path/to/foo/*"/>
  </waiver>
  <waiver name="Promote by path OR message" severity="ERROR">
    <description>
      This waiver turns into errors all problems that
      contain 'JUST_A_FOO_PROBLEM' inside their message OR were reported
      under '/path/to/foo'.</description>
    <match message="*JUST_A_FOO_PROBLEM*"/>
    <match path="/path/to/foo/*"/>
  </waiver>
  <waiver name="Disable by path AND message" severity="DISABLED">
    <description>
      This waiver disables all problems that contain 'JUST_A_FOO_PROBLEM'
      inside their message AND were reported
      under '/path/to/foo'.</description>
    <match message="*JUST_A_FOO_PROBLEM*" path="/path/to/foo/*"/>
  </waiver>
</waivers>
```

'"NOTE:"' Backslashes '\' are always treated as path separators, regardless of the OS. Therefore, you cannot u

Chapter 5. XML Preferences File Syntax

```
<!-- The XML file header, required. -->
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE spechtml PUBLIC "-//DVT//specador-preferences" "specador-preferences.dtd">
<spechtml version="6">

  <!-- GENERAL OPTIONS -->

  <!-- Location where documentation will be generated. Relative paths are solved as relative to the current di
  <location>specador_html_doc</location>

  <!-- Delete all files in the destination directory before generating documentation. -->
  <clean-location>false</clean-location>

  <!-- When running in GUI mode, choose whether to open or not the generated documentation in browser. -
  <open-in-browser>true</open-in-browser>

  <!-- Title -->
  <title>Documentation Title</title>

  <!-- The content of the overview file will be embedded in the first page.
  Relative paths are solved as relative to the current directory. -->
  <overview-file>README.TXT</overview-file>

  <!-- Beautify comments: start sentences with capital letter, append dot after sentences, bfm -> BFM, dut ->
  <enhance-comments>true</enhance-comments>

  <!-- Syntax for comments formatting: auto, naturaldocs, javadoc or none. -->
  <doc-formatting-type>auto</doc-formatting-type>

  <!-- The user defined navigation menu (HTML or XML format) will be embedded in the main menu.
  Relative paths are solved as relative to the current directory. Default: none. -->
  <user-defined-html>user_defined.html</user-defined-html>

  <!-- Add "Created by <username> ..." watermark. -->
  <created-by-user-watermark>true</created-by-user-watermark>

  <!-- Generate documentation only for public API. -->
  <public-only>true</public-only>

  <!-- API matching the below filters with not be included in the generated documentation. Default: none. --
  <!-- You can specify a comma-separated list of name patterns. Patterns may contain: * = any string, ? = any
  <filter-string>type1, type*, ty?pe</filter-string>

  <!-- You can specify one or more file or directory paths. -->
  <filter-path>/filter/path1</filter-path>
  <filter-path>/filter/path2</filter-path>
```

```
<!-- Generate UML inheritance diagram for each class, struct or unit. -->
<export-uml-inheritance-diagram>true</export-uml-inheritance-diagram>

<!-- Generate UML inheritance diagram for all class, struct or unit in each package. -->
<export-uml-package-inheritance-diagram>true</export-uml-package-inheritance-diagram>

<!-- Generate UML collaboration diagram for each class, struct or unit. -->
<export-uml-collaboration-diagram>true</export-uml-collaboration-diagram>

<!-- Generate UML direct associations diagram for each class, struct or unit. -->
<export-uml-direct-associations-diagram>true</export-uml-direct-associations-diagram>

<!-- Use orthogonal edge routing for class diagrams. -->
<vlog-orthogonal-class-diagrams>false</vlog-orthogonal-class-diagrams>

<!-- Generate design block diagram for each module, entity. -->
<export-design-block-diagram>false</export-design-block-diagram>

<!-- Generate design flow diagram for each module, entity. -->
<export-design-flow-diagram>false</export-design-flow-diagram>

<!-- Generate design schematic diagram for each module, entity. -->
<export-design-schematic-diagram>false</export-design-schematic-diagram>

<!-- Generate finite-state machine diagrams for all state variables found in the module, entity. -->
<export-fsm-diagrams>false</export-fsm-diagrams>

<!-- CROSS-LINK WITH PRE-GENERATED DOCUMENTATION -->

<!-- Link to the elements for which documentation is already generated in the directories specified below, i
<external-doc-path>/path/to/external/doc1</external-doc-path>
<external-doc-path>/path/to/external/doc2</external-doc-path>

<!-- Add entries in the main menu with links to the external documentation. -->
<show-external-doc-refs-in-navbar>false</show-external-doc-refs-in-navbar>

<!-- SYSTEMVERILOG SPECIFIC -->

<!-- Generate modules documentation. -->
<export-vlog-modules>true</export-vlog-modules>

<!-- Generate interfaces documentation. -->
<export-vlog-interfaces>true</export-vlog-interfaces>

<!-- Generate programs documentation. -->
<export-vlog-programs>true</export-vlog-programs>

<!-- Generate macros documentation. -->
<export-vlog-macros>true</export-vlog-macros>

<!-- Generate ifndef guards documentation. Default: false. -->
```

```
  <export-vlog-ifndef-guards>true</export-vlog-ifndef-guards>

  <!-- Generate control defines documentation. Default: true. -->
  <export-vlog-control-defines>false</export-vlog-control-defines>

  <!-- Generate elements in the global scope (typedefs, classes, functions, tasks etc.). -->
  <export-vlog-global-scope>true</export-vlog-global-scope>

  <!-- Generate packages documentation. -->
  <export-vlog-packages>true</export-vlog-packages>

  <!-- Generate specific package documentation. -->
  <export-vlog-package>uvm_pkg</export-vlog-package>
  <export-vlog-package>ubus_pkg</export-vlog-package>

  <!-- Generate assertions documentation. -->
  <export-vlog-assertions>true</export-vlog-assertions>

  <!-- Generate covergroups documentation. -->
  <export-vlog-covergroups>true</export-vlog-covergroups>

  <!-- Generate interface signals documentation. -->
  <hide-vlog-interface-signals>false</hide-vlog-interface-signals>

  <!-- VHDL SPECIFIC -->

  <!-- Generate libraries documentation. -->
  <export-vhdl-libraries>true</export-vhdl-libraries>

  <!-- Generate specific library documentation. -->
  <export-vhdl-library>ieee</export-vhdl-library>
  <export-vhdl-library>std</export-vhdl-library>

  <!-- E LANGUAGE SPECIFIC -->

  <!-- Generate macro documentation. -->
  <export-e-macro>true</export-e-macro>

  <!-- Generate packages documentation. -->
  <export-e-packages>true</export-e-packages>

  <!-- Generate specific package documentation. -->
  <export-e-package>main</export-e-package>
  <export-e-package>vt</export-e-package>

</spechtml>
```

Chapter 6. XML Menu File Syntax

You can create your own custom navigation menu using a settings directive like:

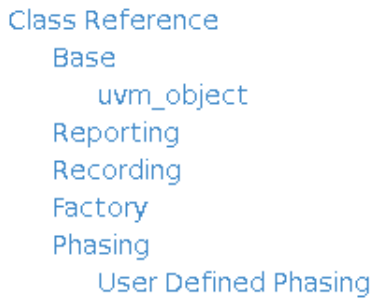
<user-defined-html>my_menu.xml</user-defined-html>

The XML menu syntax is:

```
<!-- The XML file header, required. -->
<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<menu version="1"        <!-- Optional. Syntax version.
                         Default latest. -->

 title="Intro"           <!-- Optional. A title for the menu root node.
                         Default basename, if basename.txt is specified for src. -->

 position="top"          <!-- Optional. Menu position.
                         Default bottom. -->

 src="some.txt"          <!-- Optional. A text file to process (recognize NaturalDocs or Javadoc syntax).
                         Default none. -->

 dest="some.html"        <!-- Optional. A file to link the menu entry to.
                         Default basename.html, if basename.txt is specified for src.
                         Otherwise no link. -->

 defaultPage="some.html" <!-- Optional. The default content page when opening index.html.
                         By default is the overview page, if any. -->

 overview="false"        <!-- Optional. Add Overview menu entry, unless false.
                         By default the Overview menu entry points to the Overview page.
                         The Overview page can be generated from the specified overview-file. -->
 >

   <item
    title="1 Menu"     <!-- Optional. A title for the menu node.
                         Default basename, if basename.txt is specified for src. -->

    src="chap1.txt"    <!-- Optional. A text file to process (recognize NaturalDocs or Javadoc syntax).
                         Default none. -->

    dest="ch1.html"    <!-- Optional. A file to link the menu entry to.
                         Default basename.html, if basename.txt is specified for src.
                         Otherwise no link. -->
    >
     <item src="a.txt" title="1.1 Submenu"/>
     <item src="b.txt" title="1.2 Submenu">
       <item src="a.txt" title="1.2.1 Submenu"/>
     </item>
```

```
    </item>
</menu>
```

For example you get a menu like:



from:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<menu version="1" title="Class Reference" src="docs/html/src/overviews/intro.txt" position="top" defaultPa
  <item src="docs/html/src/overviews/base.txt" title="Base">
    <item dest="uvm_pkg-uvm_object.html" title="uvm_object"/>
  </item>
  <item src="docs/html/src/overviews/reporting.txt" title="Reporting"/>
  <item src="docs/html/src/overviews/recording.txt" title="Recording"/>
  <item src="docs/html/src/overviews/factory.txt" title="Factory"/>
  <item src="docs/html/src/overviews/phasing.txt" title="Phasing">
    <item src="docs/html/src/overviews/test-phasing.txt" title="User Defined Phasing"/>
  </item>
</menu>
```

Chapter 7. Comments Formatting

There are two formatting styles supported: *JavaDoc* & *NaturalDocs*.

You can use **HTML syntax** in *JavaDoc* comments.

7.1 JavaDoc

```
/**
 * <h1>Function Description</h1> using <b>HTML Tags</b> and {@literal <b> JavaDoc </b> }
 * <ul><li>HTML list element 1</li><li>HTML list element 2</li></ul>
 * For more details: {@link http://www.dvteclipse.com/documentation/sv/Export_HTML_Documentation.html DVT Documentation}
 *
 * @param slave_name - first param
 * @param min_addr - second param
 * @param max_addr - third param
 * @return min_addr
 *
 * @see get_type
 * @see build_phase
 *
 * @author Author's name
 * @version 1.0
 */
function void set_slave_address_map(string slave_name,
  int min_addr, int max_addr);
  ubus_slave_monitor tmp_slave_monitor;
  if( bus_monitor != null ) begin
    // Set slave address map for bus monitor
    bus_monitor.set_slave_configs(slave_name, min_addr, max_addr);
  end
  // Set slave address map for slave monitor
  $cast(tmp_slave_monitor, lookup({slave_name, ".monitor"}));
  tmp_slave_monitor.set_addr_range(min_addr, max_addr);
  return min_addr;
endfunction : set_slave_address_map
```

public void

set_slave_address_map(string slave_name, int min_addr, int max_addr)

Function Description

using **HTML Tags** and JavaDoc

- HTML list element 1
- HTML list element 2

For more details: DVT Documentation

Returns:
min_addr

Arguments:
slave_name - first param
min_addr - second param
max_addr - third param

See Also:
get_type
build_phase

Version:
1.0.

Author:
Author's name

The table below lists the JavaDoc tags that DVT recognizes. For more details see http://en.wikipedia.org/wiki/Javadoc.

@param	**Valid for: functions, tasks** Adds a parameter with the specified argument-name followed by the specified description to the *"Arguments"* section
@return	**Valid for: functions** Adds a *"Returns"* section with the description text. This text should describe the return type and permissible range of values
@see	Adds a *"See Also"* heading with a link or text entry that points to reference. A doc comment may contain any number of @see tags, all grouped under the same heading
@author	Adds an *"Author"* entry
@deprecated	Adds a comment indicating that this API should no longer be used (even though it may continue to work)
@version	Adds a *"Version"* subheading with the specified version-text
@since	Adds a *"Since"* heading with the specified since-text
{@link LINK_ADDRESS LINK_TEXT}	Inserts an in-line link with visible text label that points to the documentation for the specified package, class or member name of a referenced class. This tag is valid in all doc comments. For more details see below.
{@literal text}	Displays text without interpreting the text as HTML markup or nested javadoc tags. This enables you to use regular angle brackets (< and >) instead of the HTML entities (<and >) in doc comments, such as in parameter types (<Object>), inequalities (3 < 4), or arrows (<-)

JavaDoc Links

An in-line link in a comment can be created using this tag **{@link LINK_ADDRESS LINK_TEXT}**.There are two types of links:

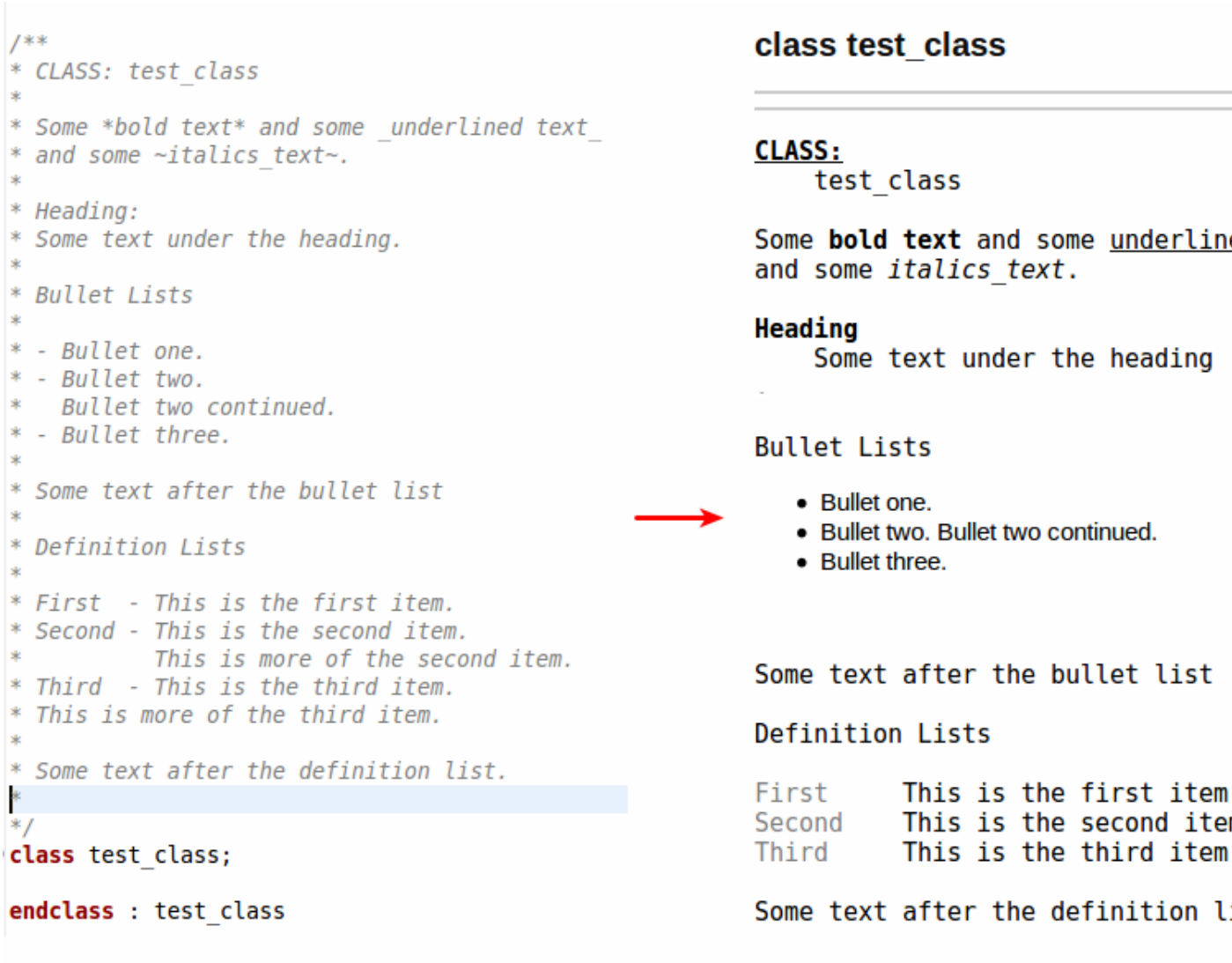
- **Internal Links** -> point to data inside Documentation. In this case **LINK_ADDRESS** must respect the following notation: **Package_Name::Class_Name.Method_Name** for an absolute path or

TYPE_NAME.INNER_TYPE_NAME or just **TYPE_NAME** for relative paths. In case of a relative path a link will be created to the best match for that type with regard to its scope inside the project.
NOTE: Using relative paths could generate broken links if there are different data types with the same name inside the project!

- **External Links** -> point to external web pages or files. For webpages **LinkAddress** must start with **http://** and for files with **file://** followed by the resource's address. For example: {@link https://www.dvteclipse.com} or {@link file://external-res.pdf}

For both types of links **LINK_TEXT** is optional and it can be used to show a user defined text instead of link's path.

7.2 NaturalDocs



The table below lists the NaturalDocs syntax that DVT recognizes. For more details see http://www.naturaldocs.org.

<LinkAddress>	Inserts an in-line link with visible text label that points to the documentation for the specified package, class or member name of a referenced class. This tag is valid in all doc comments. For more details see below.
~Italic Text~	Use tilda (~) for *Italic Text*.
Bold Text	Use star (*) for **Bold Text**.
Headings	You can add headings to your output just by ending a line with a colon and having a blank line above it.
Bullet Lists	You can add bullet lists by starting a line with **a dash, an asterisk, an o, or a plus**. Bullets can have blank lines between them if you want, and subsequent lines don't have to be

	indented. You end a list by skipping a line and doing something else.
Definition Lists	You can add a definition list by using the format below, specifically “text space dash space text”. Like bullet lists, you can have blank lines between them if you want, subsequent lines don’t have to be indented, and you end the list by skipping a line and doing something else.
Images	You can include images in your documentation by writing *(see filename)*. If you put it alone on a line it will be embedded in place.
Code and Text Diagrams	, or :'". If you have a vertical line or text box with the comment, you must separate these symbols from it with a space.

NaturalDocs Links

An in-line link in a comment can be created using **<LINK_ADDRESS LINK_TEXT>**.There are two types of links:

- **Internal Links** -> point to data inside Documentation. In this case **LINK_ADDRESS** must respect the following notation: **Package_Name::Class_Name.Method_Name** for an absolute path or **TYPE_NAME.INNER_TYPE_NAME** or just **TYPE_NAME** for relative paths. In case of a relative path a link will be created to the best match for that type with regard to its scope inside the project.
  NOTE: Using relative paths could generate broken links if there are different data types with the same name inside the project!

- **External Links** -> point to external web pages or files. For webpages **LinkAddress** must start with **http://** and for files with **file://** followed by the resource's address. For example: < https://www.dvteclipse.com> or <file://external-res.pdf>

For both types of links **LINK_TEXT** is optional and it can be used to show a user defined text instead of link's path.

Chapter 8. Customizing Documentation

Generated documentation using new HTML style can be customized using the files **<html_doc>/css/custom.css** and **<html_doc>/js/custom.js**.

The new HTML style is based on Bootstrap [http://http://getbootstrap.com/] and jQuery [http://jquery.com/] frameworks and customizations can be done using jQuery API and by changing Bootstrap's default styles.

Custom CSS

The custom Cascading Style Sheet is included last and can overwrite any style defined above.

Example: changing the color of collapsible panels holding the element names.

```
.panel-default > .panel-heading {
   background-color: Chocolate;
   color: Cornsilk;
}
```

Custom JavaScript

The custom.js JavaScript file is included at the end of the HTML and it can completely change the content of the page.

Adding a custom "CONFIDENTIAL" warning to TOC and every page header can be done using jQuery.

```
$('body').prepend('<div class="alert alert-warning text-center" role="alert">CONFIDENTIAL</div>');
```

Customizing the TOC page and the content pages separately can be achieved by testing the window name:

```
if (window.name === "content")
   $('body').prepend('<div class="alert alert-warning text-center" role="alert">This is the content frame</div>

if (window.name === "toc")
   $('body').prepend('<div class="alert alert-warning text-center" role="alert">This is the toc frame</div>');
```

A similar approach can be used to customize individual pages based on the file name:

```
if (window.location.pathname.split('/').pop() === "summary-overview.html")
   $('body').prepend('<div class="alert alert-warning text-center" role="alert">This is the the summary page<
```

Customizing the "Generated from" links that point to the relative path of the file used to generate that page can be achieved by maching all the links with the href starting with *../sv* and replacing the href prefix with the new prefix *http://some/external/location/sv* :

```
if (window.name === "content") {
```

```
 $('hr')
 .next('a[href^="../sv"]')
 .attr('href', function() {
  return this.href.replace(/^file:.*\/sv/, 'http://some/external/location/sv')
 })
 .attr('target', function() {
  this.target = "_blank";
 });
}
```

Chapter 9. What is New?

#.# - major version - Includes new features, major enhancements, architectural changes, bug fixes.
Since 2015, a major version is named in sync with the release year, for example the first major version of 201
NOTE: When switching to a new major version it is recommended to start in a new workspace.

#.#.# - minor version - Includes bug fixes, minor enhancements.

18.1.9 (12 April 2018)

Bugfixes

- DVT-11399 Build config warnings are not reported when -ignore_compile_errors is used

17.1.43 (1 February 2018)

Bugfixes

- DVT-11194 Export all libraries/packages if empty <export-vhdl-library/> or <export-vlog-package/> tags are specified

17.1.41 (19 January 2018)

Enhancements

- DVT-11023 Show a warning if _JAVA_OPTIONS or JAVA_TOOL_OPTIONS system variables are set before running the tool

17.1.36 (24 November 2017)

Enhancements

- DVT-10883 Increased default heap size to 3g and default stack size to 4m for all 64 bits distros

17.1.30 (13 October 2017)

Feature

- DVT-10273 Ability to generate an XML file with all supported preferences

Bugfixes

- DVT-10637 UVM filters for UML diagrams are not taken into account

17.1.28 (28 September 2017)

Features

- DVT-10582 Add hyperlinks to design schematic and flow diagrams

17.1.26 (14 September 2017)

Bugfixes

- DVT-10508 specador.sh doesn't return the correct exit code

17.1.17 (12 July 2017)

Enhancements

- DVT-10139 Add events from all SystemVerilog scopes containing events

Bugfixes

- DVT-10096 Remove hyperlinks from e Language events that are not covered
- DVT-10097 Events are missing class documentation

17.1.16 (30 June 2017)

Enhancements

- DVT-10024 Ability to set the location of distribution's Eclipse and JRE folders using DVT_ECLIPSE_HOME and DVT_JAVA_HOME

17.1.15 (16 June 2017)

Deprecated

- DVT-10031 Removed Graphviz UML Diagrams and the flags used to generate them

Features

- DVT-10032 Added new SVG UML Diagrams

17.1.7 (10 April 2017)

Bugfixes

- DVT-8800 Comment lines with words containing the element name are stripped from documentation
- DVT-9796 Specador: Go to element from global search does not work for mixed-language documentation

17.1.1 (24 February 2017)

Features

- DVT-3079 Generate Finite-State Machine Diagrams

Enhancements

- DVT-7673 Fail when files passed as arguments do not exist

16.1.37 (24 February 2017)

Bugfixes

- DVT-9375 Diagrams from referenced documentation are regenerated if the reference is outside of the project

16.1.35 (1 February 2017)

Enhancements

- DVT-9146 Add the diagram-max-nof-nodes in DTD for auto complete

Bugfixes

- DVT-9365 Global scope API filter does not work
- DVT-9418 Macros are documented even if excluded but Global Scope is selected

16.1.31 (9 December 2016)

Enhancements

- DVT-8159 Ignore @brief tags lines in comments
- DVT-9132 Add covergroup information in class, struct, unit pages
- DVT-9131 Ability to skip a header comment candidate that matches a simple pattern or regex when using +dvt_extract_comment_header+
- DVT-9134 Ignore invalid HTML tags when parsing comments as JavaDoc

Bugfixes

- DVT-9293 Fix NullPointerException when +dvt_auto_snps_vip_* flags are used

16.1.27 (28 October 2016)

Bugfixes

- DVT-9122 Build config: irun location is not correctly inferred when compiling in batch mode

16.1.22 (12 September 2016)

Bugfixes

- DVT-8948 When testbench classes reside under a program, they are not available in the main index

16.1.16 (8 July 2016)

Enhancements

- DVT-8135 Check that executed script is part of the same distribution where $DVT_HOME points to

16.1.9 (9 May 2016)

Features

- DVT-8567 Ability to add block, flow and schematic design diagrams using export-design-block-diagram, export-design-flow-diagram, export-design-schematic-diagram

Bugfixes

- DVT-7496 Expand on e Language checks page groups doesn't work

16.1.2 (3 March 2016)

Bugfixes

- DVT-8326 No documentation generated for inner enums, structs or classes
- DVT-8340 Wrong Java path in MacOS distros

16.1.1 (24 February 2016)

Enhancements

- DVT-7978 Updated JRE in distribution to version 1.8.0u66
- DVT-8275 Build with Java 8, minimal JRE required version increased to 1.8

15.1.37 (23 December 2015)

Enhancements

- DVT-8156 SystemVerilog: Add preference to enable/disable "Ifndef Guard Defines" extraction to HTML, do not generate by default
- DVT-8157 SystemVerilog: Add preference to enable/disable "Control Defines" extraction to HTML

15.1.34 (28 November 2015)

Enhancements

- DVT-8111 Add a preference to skip class diagram generation if maximum number of nodes exceeds a specified threshold

15.1.33 (20 November 2015)

Performance

- DVT-8092 SystemVerilog: Improve the performance of assertions and packages analysis

15.1.32 (18 November 2015)

Bugfixes

- DVT-8085 Sometimes license checkout fails when using the latest FlexLM server (11.13.1)

15.1.27 (8 October 2015)

Bugfixes

- DVT-7927 Wrong package comment processing when used for the overview page

15.1.25 (22 September 2015)

Enhancements

- DVT-7836 Ability to create URL for HTML frame content in order to simplify sharing links to specific pages

15.1.24 (18 September 2015)

Enhancements

- DVT-7832 Ability to use a specific package documentation as the overview page
- DVT-7833 Ability to filter UVM API from index and macros pages in order to avoid clutter
- DVT-7834 Show functions and tasks in generated documentation for interfaces
- DVT-7835 Provide legend for class diagrams in generated documentation

Bugfixes

- DVT-7888 Wrong progress report when linking external documentations

15.1.22 (2 September 2015)

Enhancements

- DVT-7829 Add diagram generation process timeout (1 minute)

15.1.18 (10 August 2015)

Performance

- DVT-7739 Improve Specator HTML search box performance

Enhancements

- DVT-7737 Add preference to show brief comment in index tables

15.1.17 (3 August 2015)

Features

- DVT-7727 New specador.bat Windows script

Enhancements

- DVT-6930 Ability to link header comment to first significant element in file

15.1.16 (27 July 2015)

Deprecated

- -gen_html_doc_from_settings is deprecated, use -preferences instead
- -get_html_doc flag is deprecated, use -title instead

Enhancements

- DVT-7664 Enhance progress reporting in batch mode - what file is currently generated, how long it takes
- DVT-7665 Use both extern and implementation function argument comments when generating documentation
- DVT-7667 Don't modify the capitalization of the first word in the sentence if that word is in fact the identifier name
- DVT-7669 Ability to pass custom menu by command line
- DVT-7670 Ability to pass title by command line

Bugfixes

- DVT-7663 Use portable awk syntax in scripts
- DVT-7687 Avoid silent exit after a StackOverflowError or OutOfMemoryError

15.1.11 (20 May 2015)

Bugfixes

- DVT-7474 License error due to a NullPointerException in FlexLM

15.1.10 (15 May 2015)

Bugfixes

- DVT-7449 RuntimeExceptions are thrown when generating documentation that contains some diagram types

15.1.1 (27 February 2015)

Enhancements

- DVT-7065 Build for Java 7, minimal JRE required version increased to 1.7

3.5.35 (30 January 2015)

Bugfixes

- DVT-6284 Diagrams in Specador should reflect architectures not entities in VHDL

3.5.32 (18 December 2014)

Bugfixes

- DVT-6900 Generated design diagrams are empty

3.5.30 (28 November 2014)

Bugfixes

- DVT-6854 No license found when using Specador for SystemVerilog

3.5.26 (31 October 2014)

Enhancements

- DVT-6766 Use new HTML look & feel by default

3.5.25 (23 October 2014)

Enhancements

- DVT-6431 VHDL: Added architecture instances and sub-instances

3.5.24 (10 October 2014)

Enhancements

- DVT-6278 Added compile waivers in functionality
- DVT-6710 Added support for clocking blocks

3.5.23 (2 October 2014)

Features

- DVT-6686 XML user defined menu - see XML Menu File Syntax

Enhancements

- DVT-6213 Refine parameters documentation
- DVT-6688 Show parameters in the inheritance tree
- DVT-6689 Refine interfaces documentation (ports, variables, modports, clocking blocks)

3.5.19 (21 August 2014)

Bugfixes

- DVT-6553 Specador compiles all files according to -lang switch in a mixed language build configuration regardless of extension mappings
- DVT-6565 Specador complains about non-existing irun executable even if not in ius.irun compatibility mode
- DVT-6570 FileNotFoundException (Not a directory) when generating documentation using the new HTML style

3.5.18 (1 August 2014)

Bugfixes

- DVT-6470 Specador "-h" shows help but also prints an error

3.5.17 (25 July 2014)

Features

- DVT-5560 Ability to customize the HTML look & feel when new HTML style is used
- DVT-6487 Ability to inject HTML in generated documentation when new HTML style is used

Bugfixes

- DVT-6526 Watermark footer missing in new HTML style

3.5.16 (8 July 2014)

Features

- DVT-6485 Specador: New HTML look & feel

Enhancements

- DVT-6279 SystemVerilog API defined under a class should appear only under that class and not under global API
- DVT-6434 SystemVerilog remove covergroups, assertions, functions, tasks, variables from index page and search

Bugfixes

- DVT-5550 Specador: Method argument comments are not extracted
- DVT-6210 Specador: Fixed JavaDoc @link hyperlink extraction when similar links are used

3.5.14 (24 June 2014)

Bugfixes

- DVT-6353 VHDL Overloaded functions are not visible

3.5.13 (13 June 2014)

Enhancements

- DVT-6282 Validate the settings XML before compilation

Bugfixes

- DVT-6331 Wrong hyperlinks when including external documentation directories
- DVT-6333 Show the external documentation title under the 'Referenced Documentation' section in TOC

3.5.12 (10 June 2014)

Bugfixes

- DVT-6285 & DVT-6287 Exceptions when generating documentation in certain configurations
- DVT-6286 Exception when generating documentation with module diagrams with ports
- DVT-6288 FileNotFoundException when documentation is generated with diagrams in the same location for multiple projects
- DVT-6304 Progress dialog should also include design diagrams and design diagrams with ports

3.5.11 (30 May 2014)

- First version

Chapter 10. Legal Notices

Chapter 11. Third Party Licenses

The following software may be included in this product:

- **Eclipse Platform and Eclipse Plugins** (see license [LICENSE_ECLIPSE.TXT])
- **Java SE Runtime Environment (JRE)** (see license [LICENSE_JRE.TXT])
- **ANTLR v2** (see license [LICENSE_ANTLR.TXT])
- **FreeMarker** (see license [LICENSE_FREEMARKER.TXT])
- **Graphviz** (see license [LICENSE_GRAPHVIZ.TXT])
- **jtcl** (see license [LICENSE_JTCL.TXT])
- **PMD** (see license [LICENSE_PMD.TXT])
- **P4Eclipse** (see license [DISCLAIMER_P4ECLIPSE.TXT])
- **SVNKit** (see license [LICENSE_SVNKIT.TXT])
- **viPlugin** (DVT includes a version of Michael Bartl's viPlugin)
- **UVVM** (see license [LICENSE_UVVM.TXT])
- **OSVVM** (see license [LICENSE_OSVVM.TXT])

Changes to open source projects distributed with DVT can be found in **$DVT_HOME/misc/patch**.