Control File User Guide

Imperas Software Limited
Imperas Buildings, North Weston,
Thame, Oxfordshire, OX9 2HA, UK
docs@imperas.com

Author:	Imperas Software Limited
Version:	1.1.11
Filename:	OVP_Control_File_User_Guide.doc
Project:	Control File User Guide
Last Saved:	Monday, 11 July 2022

Table of Contents

1 Preface

This document describes the capabilities of the Control file to modify configuration of a virtual platform executable and control the Simulator behavior.

1.1 Notation

`Code`	Text representing a code extract or command line
keyword	A word with special meaning.

1.2 Related Documents

- OVPsim and CpuManager User Guide
- Imperas Installation Guide

2 Introduction

Imperas simulation technology enables very high performance simulation, debug and analysis of platforms containing multiple processors and peripheral models.

The *Control File* allows control of the simulation and models when the user has no access to the platform source code or does not wish to recompile it.

Most commands documented here are also available on the command line of Imperas simulator products such as the ISS and harness.

The Control File lets the user modify behavior in the following ways:

- Simulator behavior:
    - Finish times
    - Quantum size
    - Time precision
    - Verbosity and message control.
- Connecting to a debugger
- Setting or overriding model parameters
- Loading programs
- Loading semihost libraries, analysis and verification packages
- Instruction and API tracing

Note that depending on the particular product that you downloaded, some control file features might be missing; the -help option will list the features present in your product.

3 Use of Simulation Control Files

3.1 Control File Format

A control file is written in plain text.

An entry begins with '-' (a second '-' is optional but has no significance) then a case-insensitive keyword optionally followed by space separated values or a single targeted value. Target and value are separated by '='.

```
entry : -<keyword>
        | -< keyword ><space><value>[<space><value>]
        | -< keyword ><space><target>=<value>
```

A target is the hierarchical name of a platform, processor, peripheral model, plug-in or parameter. In some cases the target can be omitted and the value applied to all appropriate targets. Thus in a single processor platform, the target is optional for any entry that refers to the processor.

A value can be a path to a file, an integer, decimal or string. A string containing spaces must be enclosed in "quotes".

```
value: <integer>
        |  <decimal>
        |  <string>
        | "<string with spaces>"
```

A path to a file can be relative to the working directory or absolute, and must be legal on the host operating system.

A control file can contain blank lines and comment lines beginning with #.

3.2 Summary of keywords

Note that the 3rd column specifies if the keyword is available in OVPsim. All keywords are available in CpuManager. Keyword availability varies in other products. Please use the keyword *-help* if in doubt.

3.2.1 Control

Keyword	**Argument**	**OVP**	**Description**
-callcommand	strings	n	Call a command in a plugin. eg. des/plugin/cmd arg1 arg2
-controlfile	filename	y	Read a control file
-dictsize	integer	y	Size in M Bytes of code dictionary (32-512)
-enabletools	[processors]	n	Load VAP tools

Keyword	Argument	OVP	Description
-enabletoolspse	[processors]	n	Load PSE VAP tools
-extlib	string	n	Add an extension library eg. des/instance=/v/l/n/v
-finishafter	string	y	Finish simulation after this many instructions. Eg. 1000 or des/inst=1000
-finishtime	string	y	Finish simulation at this time
-imperasintercepts		y	Intercept special Imperas functions
-nosimulation		y	Finish before simulation begins
-o0	[platform]	y	Disable aggressive JIT code optimization
-parallel	[platform]	n	Enable parallel processor simulation using up to 4 concurrent host threads
-parallelmax	[platform]	n	Enable parallel processor simulation using any number of concurrent host threads
-parallelopt	[platform]	n	Set options for parallel processor simulation on a multicore host
-parallelthreads	integer	n	Maximum number of concurrent host threads for parallel processor simulation
-quantum	double	y	Scheduler time quantum (in seconds) – if zero, the simulator sets a quantum time to execute a single instruction of the fastest processor
-quantumdifftime	double	n	Start time of quantum in which to search for non-determinism
-quantumseed	integer	y	Seed for randomization of schedule order
-quantumtracefile	filename	n	Write VCD file to display quantum scheduling
-stoponcontrolc	[root]	y	Simulator will stop on Ctrl-C (SIGINT)
-timeprecision	string	y	Set the simulator time precision
-volatilenativememcheck		y	Check that the JIT cache has not been left invalid when native memory is changed without notification.
-wallclock	[platform]	y	Limit maximum simulation speed to wallclock time
-wallclockfactor	string	y	Limit maximum simulation speed to wallclock time * factor

3.2.2 Debug

Keyword	Argument	OVP	Description
-consoleport	integer		Integrated debugger will listen for MI connections on this port
-consoleportfile	filename		Integrated debugger will write the MI port

Keyword	Argument	OVP	Description
			number to this file
-debugprocessor	processor or peripheral	y	Select processor(s) or peripheral(s) to attach to debugger.
-debugpseconstructors		n	Begin debug before PSE constructors has run
-debugxchars	number	y	Set the width of the debugger console
-debugychars	number	y	Set the height of the debugger console
-eguioptions	string	n	Pass these options onto the command line of GUI when started
-eguicommands	string	n	Pass these commands to the GUI at startup
-gdbcommandfile	strings	n	GDB will run this startup script. eg. des/cpu1=gdb.cmd
-gdbconsole	[platform]	y	Pop up gdb(s) in console window(s)
-gdbegui	[platform]	n	Start gdb debug in Eclipse (eGui)
-gdbflags	strings	n	Pass additional flags to a gdb eg. des/cpu1=special_flag
-gdbinit	strings	n	Pass a file to the gdb to execute before the prompt is displayed
-gdbpath	strings	y	Set the gdb path for a processor. eg. des/cpu1=/usr/bin/or1k/gdb
-nowait	strings	n	Do not wait for RSP connection before simulation
-port	strings	y	Open this port number to allow a connection to a GDB using RSP
-substitutepath	string	n	Add to list of path substitutes (debug information)
-symbolfile	filename	n	Add to list of executables to read for symbolic info

3.2.3 Diagnostics

Keyword	Argument	OVP	Description
-fetchvalidate		y	Validate instruction fetch addresses at run time (for model development)
-help		y	Print list of flags
-html			-help and -showenvvars in HTML.
-httpvis			Turn on HTTP model visualization (supported models only).
-modeldiags	string	y	Set diagnostics level for peripheral models (can be overridden)
-modulediags	module=int		Set the diagnostic level in the given module
-monitornets			Turn on net monitoring – show all writes
-monitornetschange			Turn on net monitoring – show all changes
-showbuses	[root]	y	List all (static) bus connections

Keyword	**Argument**	**OVP**	**Description**
-showbusses	[root]	y	Alias for showbuses
-showcommands	[root]	n	List commands that can be called with -callcommand
-showdomains	[root]	y	List the (initial) state of each memory domain
-showenvvars		n	List all environment variables read by Imperas products
-showfifo	[fifo]	y	Show one or all FIFO connections
-showload		y	Show where each model is loaded from.
-showmodeloverrides	[root]	y	List overrides requested by models in the platform
-showmodule	[root]	y	Print the root module and its contents
-shownet	[net]	y	Show one or all net connections
-shownets	[net]	y	Show one or all net connections
-showoverrides	[root]	y	List all possible platform overrides
-showpacketnet	[packetnet]	y	Show one or all packetnet connections
-showsystemoverrides	[root]	y	List overrides in the platform provided by the simulator
-verbosedict		y	Show code dictionary statistics

3.2.4 Library

Keyword	Argument	OVP	Description
-showlibraryextensions		y	List semihost and intercept libraries in the paths set by $IMPERAS_VLNV
-showlibrary		y	List models and platforms in the paths set by $IMPERAS_VLNV
-showlibrarymmcs			List MMCs in the paths set by $IMPERAS_VLNV
-showlibrarymodules		y	List modules in the paths set by $IMPERAS_VLNV
-showlibraryperipherals		y	List peripherals in the paths set by $IMPERAS_VLNV
-showlibraryplatforms		y	List platform executables in the paths set by $IMPERAS_VLNV
-showlibraryprocessors		y	List processors in the paths set by $IMPERAS_VLNV
-showvariants	[processor]	y	Show variants in one or all processors
-vlnvmap	string	n	Add to list of VLNV substitute references (mappings)
-vlnvroot	path	y	Add to the search path for models

3.2.5 Log

Keyword	Argument	OVP	Description
-excludem	string	y	Exclude this message category (can be repeated)
-logfile	filename	y	Output log file
-nobanner			Suppress product banner
-nowarnings		y	Suppress warnings
-output	filename	y	Output log file
-quiet		y	Suppress information messages
-semihostlogfile	filename	y	Direct semihost output to this file
-tracefile	filename	y	Direct instruction tracing to this file
-verbose	[string]	y	Produce verbose output. Optional arguments are `commands, debugger, network, program, stats.` With no options, all are written.
-version		y	Print version information
-werror		y	Treat warnings as errors

3.2.6 MPD

Keyword	Argument	OVP	Description
-batch	filename	n	Execute tcl files in batch mode

Keyword	Argument	OVP	Description
-consoleport	port		Listen for MI connections
-consoleportfile	filename		Write a file containing the port number
-interactive		n	Start interactive Tcl session
-mi	[platform]	n	Start interactive session in MI mode
-mpdconsole	[platform]	n	Pop up mpd in a console window
-mpdconsolelogfile	[platform]	n	MPD will write its log to this file
-mpdegui	[platform]	n	Start MPD debug in Eclipse (eGui)
-searchpath	string	n	Add to debugger source search path. e.g. <path or proc=<path>

3.2.7 Parameters

Keyword	Argument	OVP	Description
-override	string	y	Override a value in platform definition. eg. des/inst=123
-variant	string	y	Set processor variant. eg. V1 or des/p1=V1

3.2.8 Processor model

Keyword	Argument	OVP	Description
-cpuflags	Integer	y	Processor model control flags

3.2.9 Program

Keyword	Argument	OVP	Description
-argv	strings	y	Pass all remaining values to the application main (applies to all processors)
-elfusevma	strings	y	Use ELF VMA addresses rather than LMA
-envp	strings	y	Pass values (until the next '-') to the application environment list
-loadlimit32	bool	y	limit the loader to this many bits per transaction.
-loadlimit64	bool	y	limit the loader to this many bits per transaction.
-loadphysical	strings	y	Use ELF physical addresses
-loadsignextend			Sign extend 32b addresses to 64b
-objectloader	filename	n	Add to the list of built-in object loaders
-objfile	filename	y	Load object onto CPU. Set PC to start address
-objfilenoentry	filename	y	Load object onto CPU. Do not set PC to start address
-objfileuseentry	filename	y	Load object onto CPU. Set PC to start address

Keyword	Argument	OVP	Description
-program	filename	y	Execute this program. eg. prog.elf or des/p1=prog.elf
-reset	string	y	Specify processor start address. eg. <addr> or proc=<addr>
-startaddress	string	y	Specify processor start address. eg. <addr> or proc=<addr>

3.2.10 Replay

Keyword	Argument	OVP	Description
-modelrecorddir	directory	n	Models will record their external events in this dir
-modelreplaydir	directory	n	Models will replay their external events from this directory

3.2.11 Trace

Keyword	Argument	OVP	Description
-trace	[processor]	y	Trace instructions as they are executed
-traceafter	string	y	Start tracing instructions after this many have executed
-tracebuffer	[processor]	y	Enable the trace buffer
-tracechange	[processor]	y	Trace changed registers
-tracecount	string	y	Trace this number of instructions
-tracefile	filename	y	Direct instruction tracing to this file
-traceintercept	[processor]	y	Trace intercepted functions

-tracemem	[processor][=XSL]	y	Trace each associated memory access
			X=only code fetches
			S=only system access
			L=only loads/stores

-tracepseintercept	[peripheral]	y	Trace intercepted functions
-traceregs	[processor]	y	Dump registers after each instruction is executed
-traceregsafter	[processor]	y	Dump registers after each instruction is executed
-traceregsbefore	[processor]	y	Dump registers before each instruction is executed
-traceshareddata	[processor]	y	Trace the use of vmirtWriteListeners
-traceshowicount	[processor]	y	Show instruction count with each instruction

Example:

```
# A simulation control file "control.ic"

    -extlib            plat1/cpu1/ext1=vapTools
    -override          plat1/cpu1/ext1/functiontrace on

    -objfileuseentry  plat1/cpu2=boot.elf      # boot code
    -symbolfile       plat1/cpu1=linux.elf     # symbols for linux
    -callcommand      plat1/cpu1/dumpTLB       # show initial contents of TLB

# end of file
```

3.3 Specifying control files on the Imperas simulator command line

A control file can be specified to the Imperas simulator using the command line argument `-controlfile`. This argument can be repeated to specify more control files. e.g.

```
shell> harness.exe -modulevendor mips.ovpworld.org -modulename MipsMalta \
      -controlfile control1.ic \
      -controlfile control2.ic
```

3.4 Specifying control files with the IMPERAS_TOOLS environment variable

Control files can be specified using the environment variable `IMPERAS_TOOLS`. Filenames are separated by `:` (Linux) or `;` (Windows) This method can be used with CpuManager when the simulator is a shared object that has no command line e.g.

```
shell> export IMPERAS_TOOLS="control.ic:control2.ic"

shell> mySystemC.exe    # simulator using CpuManager
```

3.5 Keyword descriptions

3.5.1 -argv <argument1> <argument2> ….

Pass all remaining arguments to the application `main(int argc, char *argv[])`.
This feature is only of use when a single processor is using a semihost library.
Note that all subsequent command line arguments are absorbed and passed to the application so this MUST BE USED AFTER any simulator arguments.

3.5.2 -batch <TCL command file> …

If the product supports interactive debug , the debugger will be started in TCL mode and will execute the given script or scripts. If -`interactive` is also specified a prompt will be printed and the simulator will wait for TCL commands after executing the given scripts.

Summary of debugger keywords:

Keyword	**interactive**	**language**	**note**
-batch <file>	N	TCL	execute this tcl file first

-debugpseconstructors	Y	GDB	stop before PSE constructors run

3.5.3 -callcommand <cmdstring>

Call a command in a model or plugin. Commands are specific to the model or plugin. <cmdstring> must start with the hierarchical name of a command in a processor or plugin, followed by the command's arguments, if any, separated by spaces. Commands with arguments must be enclosed in quotes. Use -`showcommands` to get a list of available commands.
Use `-callcommand ”command -help”` to learn about a command's arguments.

e.g.

```
-showcommands

-callcommand  “plat1/proc1/debug -help”

-callcommand  “plat1/proc1/ostrace/trace -on”
```

Any value returned by a command will be printed by the simulator with a label:

```
 Info (CMD_CR) <return value>
```

3.5.4 -cpuflags [processor=]<integer>

e.g.
```
-cpuflags plat1/proc2=0xA3
```
or
```
-cpuflags 0xFF
```

Some processor models have test modes and other non-standard functionality controlled by this value, which the model retrieves using the `vmirtProcessorFlags()` function. The value specified using this command in the control file will override a value specified in the platform. The interpretation of the flags value is processor model specific.

3.5.5 -consoleport <portnumber>

Integrated debugger will listen for MI connections on this port. The port number can be zero in which case a port is allocated from the host's pool. See `-consoleportfile.`

The same flag is available to `mpd.exe` - the Imperas multiprocessor debugger.

This allows the debugger to be controlled from two sources at the same time - usually a graphical debug program such as eclipse, and some kind of external test system.

3.5.6 -consoleportfile <file>

Used with -`consoleport 0.` A file of this name will be written containing the port number that was allocated from the host's pool.

3.5.7 -controlfile <file>

Read another control file. This option allows a control file to refer to other control files.

3.5.8 -debugprocessor <targets>

When using a single-processor debugger, such as gdb, in a multi-processor platform, you must specify which processor to debug. e.g.

```
-debugprocessor  plat1/proc2    # connect gdb to this processor
```

Most processor models can be connected to the Imperas remote Multi-Processor debugger. This allows simultaneous debug of more than one processor in the same platform. In this case you should specify the `-port` option; the `-debugprocessor` option is not required. Please refer to the Imperas_Debugger_User_Guide.

Some Imperas simulators are able to connect simultaneously to several debuggers in which case this command can be repeated. e.g.

```
# (don't debug 1 and 2)
-debugprocessor  plat1/proc3 plat1/proc4   # connect 1st gdb to this processor
                                           # connect 2nd gdb to this processor
# (don't debug 5 and 6)
```

When debugging with MPD -`debugprocessor` can be used to reduce the number of processors or models present in MPD, useful when debugging very large platforms. Without this flag, all processors and peripherals are present for debug. With the flag, only the specified models are present. Note that -`debugprocessor` can be repeated or followed by a list of instance names. Names can include the glob characters ? and * (when working in a shell environment, be careful that specification with glob characters does not match filenames in the current directory).

3.5.9 -debugpseconstructors

When used with -`batch, -mpdconsole` or -`mpdegui` this command prevents the peripheral model constructors from running before the debugger starts. Use this command when debugging peripheral model constructors. Note that until the peripheral model constructors have run, peripheral model user views will not be available.

3.5.10 -debugxchars, -debugychars

Specify the dimensions of the popup debug console (see `-gdbconsole` and `-mpdconsole`).

3.5.11 -dictsize <Mb>

Change the size of the simulator code dictionary (in megabytes). Values are allowed in the range 32 to 512.

3.5.12 -eguioptions <string>

Pass <string> options to the Imperas Eclipse Debugger command line when starting.

3.5.13 -eguicommands <string>

Pass <string> commands to the Imperas Eclipse Debugger when starting.

3.5.14 -elfusevma [<proc>]

e.g.

```
-elfusevma plat1/proc1
```

or

```
-elfusevma   # there is only one processor in this platform
```

By default the ELF reader loads to the Load Memory Addresses (LMA) specified in the ELF file rather the Virtual Memory Addresses (VMA). The LMA can differ from the VMA in ELF files where data is to be copied by a program's startup code, usually from ROM to RAM.

This option may be used to override the default behavior and use the VMA instead of the LMA. This would be needed if for some reason the startup code that performs the copy is not included in the application.

Note that in release 20130630 and earlier the ELF loader would load to the VMA rather than the LMA. Later releases use the LMA by default and this option my be used to get the previous behavior.

3.5.15 -enabletools <target>

e.g.

```
-enabletools

-enabletools plat1/proc*
```

Load all analysis tools for all processors or a subset of all processors. Please refer to the Imperas VAP Tools User Guide.

3.5.16 -enabletoolspse <target>

e.g.

```
-enabletools

-enabletools plat1/pse1
```

Load all analysis tools for all peripheral models or a subset of all models. Please refer to the Imperas VAP Tools User Guide.

3.5.17 -endian <big|little>

Set the endian of the processor. This argument is only required if the program ELF file does not contain this information and if the processors has endian control that is not set by the platform.

3.5.18 -envp <string>

Pass the string to the application `main(…. char *envp)` argument.
This feature is only of use when a single processor is using a semihost library.

3.5.19 -excludem <string>

e.g.

```
-excludem OP_     # suppress all OP messages
```

Suppress simulator messages with this prefix or part of prefix. Use this to improve the readability of simulator output when too many messages are appearing.

3.5.20 -extlib <target>=<library>

Loads an extension library or plug-in onto an application processor. The value of <library> can be a VLNV (See Imperas Simulator Guide) of a library in the VLNV tree, or a path to a shared object. The target can include wild-cards, can contain just the processor name (omitting the module hierarchy) or, if there is only one processor, be omitted completely.

If the target completely matches a processor name, an instance name for the library is chosen by the simulator. If the target name has an extra field then this name is taken by the extension library instance.

For example:
The library is installed on the processor `plat1/proc1`
The installed library will be given a made-up name.
The library is found in the Imperas library by its VLNV:

```
-extlib plat1/proc1=imperas.com/intercept/vapTools/1.0
```

The library is installed on the processor `plat1/proc3`
The installed library is in the local directory `myLocalDir`
The library is called `model`, `model.so`. or `model.dll`

```
-extlib plat1/proc3=myLocalDir/model
```

The installed library gets its name `vap1` from here
The library is found in the Imperas library by its VLNV

```
-extlib plat1/module1/proc2/vap1=imperas.com/intercept/vapTools/1.0
```

There is only one processor in this design.
There is only one item called vapTools in the Imperas library.

```
-extlib vapTools
```

If the library has parameters that may be overridden or commands that can be called, it is useful to specify the name of the library instance rather than letting the simulator choose a name.

3.5.21 -fetchvalidate

This option is used during processor model development. It causes the simulator to validate each memory fetch at run-time.

3.5.22 -finishafter <integer>

e.g.

```
-finishafter plat1/proc1=12374     # finish after this many instructions
```

Stop the simulation when the specified processor has executed this many instructions. Note that the round-robin scheduling might cause other processors to execute more or fewer than this number.

e.g.

```
-finishafter 10000                 # finish after this many instructions
```

This number applies to all processors.

3.5.23 -finishtime <decimal>

e.g.

```
# stop after 2 and three quarter seconds
-finishtime 2.75
```

Finish the simulation after this time in seconds. Use this to limit the execution time of a simulation should no other event cause it to finish. If a finish time is not specified, the simulator will run indefinitely or until some other event causes the simulator to stop.

3.5.24 -gdbcommandfile [processor=]<path>

e.g.

```
-gdbconsole
-debugprocessor plat1/proc1
-gdbcommandfile plat1/proc1=mystartup.cmd
```

When using -`gdbconsole` specify this command file for execution by the gdb attached to the specified processor. Use this to set complex breakpoints or conditions prior to getting control of the debugger.

3.5.25 -gdbconsole

The simulator will pop-up a console window (or xterm) running a gdb connected to a specified processor (see -`debugprocessor`). The path to an appropriate gdb is usually specified in the processor model. See `-gdbpath` to change the gdb.

3.5.26 -gdbegui

The simulator will connect to the Imperas Eclipse debugger (eGUI) if it is running (on the same computer) or else start the Eclipse debugger and then connect to it. It will use gdb as the debug agent and select the appropriate gdb for the target processor.

3.5.27 -gdbflags [processor=]<flags>

Pass the given flags to every gdb or to the gdb attached to the named processor.

3.5.28 -gdbinit <string>

After starting a gdb, give this string to the gdb command line.

3.5.29 -gdbpath [processor=]<path>

e.g.

```
-gdbconsole
-debugprocessor plat1/proc4
-gdbpath        plat1/proc4=/home/me/mine/gdb
```

or

```
-gdbpath        /home/me/mine/gdb   # apply to all processors
```

Override the path to the gdb used to debug this processor. Use this when the gdb specified in the processor model is incorrect, missing from your installation or when you are debugging a new processor model.

3.5.30 -help

Print commonly used command line options.

3.5.31 -helpall

Print all command line options.

3.5.32 -html

Change the the format of some of the commands in this document to html.
Applies to `-help` and `-showenvvars`.

3.5.33 -httpvis

Enable HTTP visualization. Models that support this mode start a simple web server. The port number is indicated in the simulator log. The user can then point a web browser at the model to see a visual representation of the model output. Some models also take switch and dial inputs from the browser.

3.5.34 -imperasintercepts

Causes the simulator to intercept special function names. Refer to Simulation_Control_of_Platforms_and_Modules_User_Guide

3.5.35 -loadlimit32, -loadlimit64

Limits the size of each transaction to this many bits. This slows the loading of a program but should be used if bus transactions are being modeled and there is a maximum transaction size, e.g. in a SystemC TLM platform.

3.5.36 -loadphysical [processor]

e.g.

```
-loadphysical plat1/proc1
```

or

```
-loadphysical   # there is only one processor in this platform
```

This directs the program loader to load code at the physical addresses in the program file instead of the logical addresses which is more usual. This is very unusual. Refer to the documentation of your compiler and linker for more information.

Note that in release 20130630 and earlier the ELF loader would load to the VMA rather than the LMA. This option could be used to work around this problem, but could cause additional issues with symbols. Later releases use the LMA by default and so this option is no longer needed in most cases.

3.5.37 -loadsignextend [processor]

e.g.

```
-loadsignextend plat1/proc1
```

Some ELF files specify 32-built addresses but require that addresses are sign-extended to 64 bits before loading into memory. This option enables that behaviour.

3.5.38 -logfile <file>

The simulator will write its output to the given file name as well as to the output stream. The file is prefixed with the simulator’s command options.

3.5.39 -logflush

The simulator log file is updated as soon as each line of text is written. By default the contents might be buffered for a while.

3.5.40 -mi

Puts the Imperas Multiprocessor Debugger command line into MI (gdb machine interface) mode. This is required when mpd is connected to a graphical debugger.

3.5.41 -modeldiags [model=]value

e.g.

```
-modeldiags plat1/dma1=0xF   # turn on one model’s diagnostics
```

or

```
-modeldiags 0xFF             # all models, maximum output
```

Most peripheral models have diagnostic output which can be turned on using this keyword. Refer to the Imperas Peripheral Modeling Guide for mode information.

3.5.42 -modulediags [model=]value

e.g.

```
-modulediags plat1/mod1=0xF   # turn on one module’s diagnostics
```

or

```
-modulediags 255              # all modules, all output
```

Most peripheral models have diagnostic output which can be turned on using this keyword. Refer to the Imperas Peripheral Modeling Guide for mode information.

3.5.43 -modelrecorddir <path>

e.g.

```
-recorddir /tmp/model_records   # all models save their output in here
```

Some peripheral models are able to record all their external input for replay in subsequent simulations. Applied to all peripheral models with external inputs, this makes the simulation deterministic so that bugs or other events of interest happen at exactly the same simulated time. Without this feature, some bugs are impossible to reproduce. Note that all models with external (and hence asynchronous) inputs must use this feature to achieve determinism. Please refer to Record and Replay in the Peripheral modeling guide.

3.5.44 -modelreplaydir <path>

e.g.

```
-replaydir /tmp/model_records   # all models get their input from here
```

Some peripheral models are able to record all their external input for replay in subsequent simulations. Applied to all peripheral models with external inputs, this makes the simulation deterministic so that bugs or other events of interest happen at exactly the same simulated time. Without this feature, some bugs are impossible to reproduce. Note that all models with external (and hence asynchronous) inputs must use this feature to achieve determinism. Please refer to Record and Replay in the Peripheral modeling guide.

For each peripheral that supports replay (which can be determined by looking for record and replay parameters to the model), the simulator looks for a file of the correct name in the given replay directory.

3.5.45 -mpdconsole

The simulator will pop up a console (or xterm) containing the remote multiprocessor debugger, and wait for input from this console. This feature is useful for connecting the MPD to your SystemC platform or other third party simulator that uses OVP models.

3.5.46 -mpdconsolelogfile <file>

MPD will write its log to the given file.

Notes:

- This file is separate from the simulator log file.
- MPD can be instructed to redirect the simulator log back to itself (TCL command `iredirect -debugger`), which will then be included in the MPD log file.
- The file idebug.log is always written by MPD. It includes all commands executed by MPD, MPD output and redirected simulator output.

3.5.47 -mpdegui

The simulator will connect to the Imperas Eclipse debugger (eGUI) if it is running (on the same computer) or else start the Eclipse debugger and then connect to it. It will use the Imperas multiprocessor debugger as the debug agent.

3.5.48 -monitornets

When a net is written by a model in the design the simulator will write the details to the output stream (and to the log file if specified). In a platform with two levels of hierarchy,nets can be specified like this:

Example	**Monitors**
-monitornets	all nets
-monitornets top/mod/net	one named net
-monitornets top/mod	all nets in the module and its sub-modules
-monitornets irq	any net in the hierarchy with this name
-monitornets irq*	wildcards * - any characters ? - single character

Use -`verbose` to confirm which nets are monitored.
Use -`shownets` to get a list of all nets in the platform.

3.5.49 -monitornetschange

When a net is written with a new value by a model in the design the simulator will write the details to the output stream (and to the log file if specified). Nets can be specified as for -`monitornets` (see above).

3.5.50 -nosimulation

The simulator will exit before application code is executed. This can be used to check that a design loads correctly.

3.5.51 -nowait

Used with `-port`, the `-nowait` option lets the simulator start executing instructions without connecting to a debugger, but continue listening for a connection during simulation.

3.5.52 -nowarnings

Suppress simulator output with the ‘Warning’ severity. Use this with care; it might hide important information about an error in your configuration.

3.5.53 -objectloader <path>

e.g.

```
-objectloader mydir/myLoader    # .so or .dll extension is added automatically
```

Provide a loader to load your own object files. Please refer to the CpuManager User Guide for details of the object loader API.

3.5.54 -o0

Turn off aggressive JIT code optimization. Use only if so instructed by Imperas.

3.5.55 -program or -objfile or -objfileuseentry [processor=]<path>

e.g.

```
-objfileuseentry plat1/proc2=mydir/myprogram.elf  # program for one processor
```

or

```
-program mydir/myprogram.elf              # program for all processors
```

(The three keywords are synonymous) Load a program for execution by an application processor. The program must be in a format supported by the simulator and compiled for the application processor. The simulator will set the PC of the application processor to the program's entry point before starting. Debug and global symbols will be read by the simulator for use in trace output and analysis tools. Use this option when simulating an application program without an OS or boot program.

3.5.56 -objfilenoentry [processor=]<path>

This is the same as `-objfileuseentry` but does NOT set the PC before starting simulation. Use this option when simulating a processor starting from its reset vector, or to load additional object files.

3.5.57 -output <file>

Write a copy of standard output to the given file. See `-logfile`.

3.5.58 -override target=value

e.g.

```
-override plat1/proc1/variant=VAR99           # a model parameter
-override plat1/proc2/mips=150                # a simulator parameter
-override plat1/proc3/extlib1/libLog=log.txt  # an extension library parameter
```

Set the value of a model or simulator parameter. Use the `-showoverrides` argument to see a list of parameters accepted by the simulator and the loaded models.

3.5.59 -parallelmax

Enable parallel processor simulation using any number of concurrent host threads. Refer to Advanced_Simulation_Control_of_Platforms_and_Modules_User_Guide.

3.5.60 -parallelopt

Set options for parallel processor simulation on a multicore host. Refer to Advanced_Simulation_Control_of_Platforms_and_Modules_User_Guide.

3.5.61 -parallel

Enable parallel processor simulation using up to 4 concurrent host threads. Refer to Advanced_Simulation_Control_of_Platforms_and_Modules_User_Guide.

3.5.62 -parallelperipherals

Enable parallel peripheral simulation using any number of concurrent host threads. Refer to Advanced_Simulation_Control_of_Platforms_and_Modules_User_Guide.

3.5.63 -parameter <parameter>=<value>

Set the value of a model or simulator parameter. Use the `-showoverrides` argument to see a list of parameters accepted by the simulator and the loaded models.
Equivalent to `-override`.

3.5.64 -port <integer>

e.g.
```
-port 3000
```

Specify the port number for the simulator to listen on for connection to a debugger.

If the number is zero, a port is chosen from the host's pool and its number printed in the simulator log. To automate connection to the simulator, set the environment variable IMPERAS_PORT_FILE to a path, start the simulator then wait until this file is written. It contains the chosen port number which can then be used by the debugger.

If you wish to debug application code on a single processor in your platform, use `-port` to specify a port number. If your platform contains more than one processor use `-debugprocessor` to specify which processor to debug. Having started your simulator and your gdb, use the gdb command:
```
        target remote <hostname>:<portnumber>
```
to connect to the port.

By default, the simulator will wait for connection before any instructions are executed.

Some port numbers are used by other services. If the simulator reports this, try another number.

3.5.65 -profile <number of categories>

Enable processor model instruction profiling, showing this number of profile categories (Supported on Linux only).

3.5.66 -program <file>

Load a program onto a given processor. Refer to `-objfile.`

3.5.67 -quantum

e.g.
```
-quantum 0.002    # 2mS quantum; twice the default.
```

Sets the duration (in seconds) of the time-slice or quantum executed by each processor in turn. The default is 0.001s.

This keyword works if the platform calls opRootModuleSimulate(). If the simulation uses its own scheduler and calls opProcessorSimulate() then changing the quantum has no effect. This is the case in a SystemC platform.

3.5.68 -quantumdifftime
Specify the start time of the quantum in which to search for non-determinism.

3.5.69 -quantumseed <integer>
e.g.
```
-quantumseed 1234
```

Use a random schedule order for each processor instead of the default round-robin. The integer specifies a pseudo random seed. This applies if opRootModuleSimulate() is used. If the simulation uses your own scheduler and opProcessorSimulate(), it has no effect.

3.5.70 -quantumtracefile <file>
Write a VCD file showing quantum scheduling.

3.5.71 -quiet
Suppress simulator and model ‘Info’ messages.

3.5.72 -registerset <register>=<value>
Set a peripheral register to this initial value. This will happen after the peripheral model initialization code has executed (initialization code is called from `main()`).
e.g.
```
-registerset plat1/periph1/reg1=0xff
```

3.5.73 -reset or -startaddress [processor=]address
e.g.
Set the start address of the given processor:
```
-reset plat1/proc1=0xBF00000
```

Set the start address of the only processor:
```
-reset 0xBF00000
```

Set the PC of the specified processor to this value before starting the simulation. This will override a start address found in the application program, and override the reset vector in the model.

3.5.74 -searchpath
Tell the debugger where to find source files that have been moved since the program was compiled. In a multicore platform each processor could have source in a different location, so paths can be prefixed with the processor instance name. e.g.
```
-searchpath procA=/home/user/applicationSource
```
or
```
-searchpath localsource/src
```

3.5.75 -semihostlogfile <filename>
Send the semihost output to this file.

This prevents text from application code getting mixed up with output from the simulator, though the temporal relationship between semihost output and simulator events will be lost.

3.5.76 -setprotection

Modify the simulator memory protection according to the program headers of the loaded program. This command is only useful in a platform that does not model specific RAM and ROM components.

In a simple platform it is usually convenient to create memory in the whole address range of the processor. By default, the memory will allow read, write and execute operations. If `-setprotection` is used, the memory will only allow read and write operations. Where code is loaded, execute permission will be allowed as well. This prevents the processor from accidentally executing from memory where no code was loaded.

3.5.77 -showfifo [fifoname]

Print all the FIFOs and their connections. Use this to check your design is correctly constructed. If a FIFO name is supplied, only that FIFO is printed.

3.5.78 -shownets [netname]

Print all the nets, their connections and their current value. Use this to check your design is correctly constructed. (-shownet is also accepted). If a net name is supplied, only that net is printed.

3.5.79 -showpacketnet [packetnetname]

Print all the packetnets and their connections. Use this to check your design is correctly constructed. If a packetnet name is supplied, only that packetnet is printed.

3.5.80 -showbuses

Print all the buses and their connections. Use this to check your design is correctly constructed. (-showbusses is also accepted). If a bus name is supplied, only that bus is printed.

3.5.81 -showcommands

Print a list of model commands that may be called with `-callcommand.`

3.5.82 -showdomains

Print a list of memory domains. Use this to check any dynamically mapped components.

3.5.83 -showlibrary

Show all models found in the libraries at the paths set by $IMPERAS_VLNV.

3.5.84 -showload

Print when each model instance is loaded and the file it was loaded from. Use this to confirm that the search path is behaving as you expect.

3.5.85 -showmodule <module>
Show the contents of the given module.

3.5.86 -showoverrides
Print a list of platform, simulator and model parameters that can be set with `-override`.
Cut entries from the output of this command and paste into a control file with suitable values.
-showmodeloverrides excludes parameters that control the simulator.
-showsystemoverrides excludes parameters that control the models in the platform.

3.5.87 -showlibraryperipherals
Show the peripheral models found in the libraries at the paths set by $IMPERAS_VLNV.

3.5.88 -showlibraryprocessors
Show the processor models found in the libraries at the paths set by $IMPERAS_VLNV.

3.5.89 -showlibrarymodules
Show the modules found in the libraries at the paths set by $IMPERAS_VLNV.

3.5.90 -showlibraryvendors
Show the vendor names found in the libraries at the paths set by $IMPERAS_VLNV.

3.5.91 -showvariants [processor]
Show all the variants supported in the given processor model. If no processor is specified, all are listed. Also accepts an output file name, e.g.

```
-showvariants              # variants of each processor to the simulator log
-showvariants var.txt      # variants of each processor to file var.txt
-showvariants cpu0=var.txt # variants of each cpu0 to file var.txt
```

3.5.92 -startaddress
Refer to `-reset.`

3.5.93 -stoponcontrolc
The simulator will stop when it receives a control-C (SIGINT on Linux).

3.5.94 -substitutepath <path>=<path>
e.g.
```
-substitutepath /home/develop/main=/local/debug/main
```

Use this to inform the debugger that source files used to compile the application being debugged are now at a different disk location. In the example, the application was compiled in /home/develop, so the debug symbols in the application program file will contain those references and the debugger will look there when debugging the program. If, since compiling the code, the files have been moved to /local/debug, the debugger will fail to find them. Use this to tell the debugger where they are now.

3.5.95 -symbolfile [processor=]<path>[@text:<address>]

e.g.

```
-symbolfile plat1/proc2=mydir/myprogram.elf # program for one processor

-symbolfile myprogram.elf@text:0xf000 # program for any processor, loaded at 0xf000
```

Sometimes the application program is not loaded explicitly by the simulator. It might be read from a simulated disk, or uncompressed by a boot loader. Use this command to pass the symbols to the simulator for use by intercept libraries, disassembly and tracing. The symbol file must be in a format accepted by one of the installed object loaders.

The optional `@text:<address>` will override the address of the text segment.

3.5.96 -timeprecision <integer>

Change the simulator time precision. This is the number to which time calculations are rounded. The default value is 1e-9.

3.5.97 -trace [processor]

e.g.

```
-trace p              # tracing of all processors
```

or

```
-trace plat1/proc2    # tracing of one processor
```

Turn on basic instruction tracing on the specified processor. If no processor is specified, all processors are traced.

NOTE: Tracing will slow the simulation by several orders of magnitude.

3.5.98 -traceafter [processor=]<integer>

Begin tracing after this many instructions on the specified processor. If no processor is specified, all processors are traced.

e.g.

```
-traceafter plat1/proc2=10000000    # tracing of one processor
```

or

```
-traceafter 100000000
```

3.5.99 -tracebuffer

Turn on a circular trace buffer which continually records the last 256 instructions executed. The trace buffer may be displayed from MPD and will also be written at the end of simulation.

NOTE: Use of the trace buffer will slow the simulation slightly.

3.5.100 -tracechange

When tracing is enabled, after each trace line, report any new register values.

3.5.101 -tracefile <filename>

All processor trace data goes to this file instead of to standard output or to the simulator log file.

3.5.102 -traceintercept [processor]

Turns on tracing of intercepts for the given processor, or all processors if unspecified. The entry, parameter values and return from every intercepted function will be recorded in the simulator log. This does not apply to peripheral models – see `-tracepseintercept`

3.5.103 -traceregs [processor]

When tracing is enabled, after each trace line, report all register values.

3.5.104 -tracelowpc and -tracehighpc [processor=]<integer>

When tracing is enabled, use these parameters to restrict the PC that is traced. The instructions traced can be limited to only only those instructions with a PC equal to or lower than the address specified by tracelowpc and/or with a PC equal to or higher than the address specified by tracehighpc.

3.5.105 -tracesharedata

Report activity of the vmirtWriteListeners API. See the VMI Run-time Programmers Reference.

3.5.106 -tracemem [processor=] [XSL]

Trace each memory access associated with the instruction. This command accepts a string containing any of X,S or L. X traces each code-fetch, S traces each system access and L traces each load and store.

3.5.107 Traceshowicount [processor]

When tracing is enabled, display the instruction number since the start of simulation in each trace line.

3.5.108 -tracemode [processor]

When tracing is enabled, display the current processor mode in each trace line.

3.5.109 -tracepse [pse]

Trace instructions executed by this peripheral model.

3.5.110 -tracepseintercept [peripheral]

Turns on tracing of intercepts for the given peripheral, or all peripherals if unspecified. The entry, parameter values and return from every intercepted function will be recorded in the simulator log. For processors, see `-traceintercept`

3.5.111 -variant [processor=]string

Override the variant of the specified processor. If no processor is specified, all processors will be set to the given variant.

e.g.
```
-variant plat1/proc2=VAR99  # set the variant in a multi-processor platform
```
or
```
-variant VAR99              # set the variant in a single-processor platform
```

The values allowed are processor model specific. If an invalid value is specified, a list of valid values will be displayed.

NOTE: Changing the variant might change the processor’s I/O interface causing connection errors when the platform is constructed. Use with care.

3.5.112 -verbose

Produce more information while starting, running and finishing simulation. These arguments are recognized:

Argument to -verbose	**Action**
`commands`	Report when model commands are called
`debugger`	Report when debuggers are launched
`network`	TCP/IP statistics from the virtual network
`program`	Information from the program loader
`stats`	Processor performance statistics

Without arguments all information is printed.

3.5.113 -verbosedict

Report information on the code dictionary at the end of simulation.

3.5.114 -vlnvmap v/l/n/v=v/l/n/v

e.g.
```
-vlnvmap ovpworld.org/processor/or1k/1.0=develop/processor/or1k/1.0
```

Replace a platform component without recompiling the platform.

3.5.115 -vlnvroot <path>

e.g.
```
-vlnvroot /home/develop/library
```

Add an additional directory to search for models.

3.5.116 -volatilenativememcheck

Check that the JIT cache has not been left invalid when native memory is changed without notification.

If your platform imports natively allocated memory into the simulation and then code is executed from this area, the simulator must be notified if the memory is changed from outside the simulator.

Enabling this (slow) mode will check for and report un-notified changes each time the code is executed.

3.5.117 -wallclockfactor <decimal>

e.g.

```
-wallclockfactor 2.0      # limit maximum simulation speed to twice real-time
```

Limit maximum simulation speed to this many times of the wall clock (real time). Prevents time from racing forwards when there is no simulation activity. Use this if your simulation connects to a terminal or other GUI into which you wish to type. This applies if opRootModuleSimulate() is used. If the simulation uses your own scheduler and uses opProcessorSimulate(), it has no effect.

3.5.118 -wallclock

Equivalent to `-wallclockfactor 1.0`

3.5.119 -werror

Treat warnings as if they are errors. A message with ‘Error’ severity during platform construction will prevent simulation. This option causes ‘Warning’ severity messages to be treated the same way.

##