



## iGen Model Generator Introduction

This document introduces the use of the Imperas Model Generator *iGen* ;  
A component of the Imperas Professional tools

### Imperas Software Limited

Imperas Buildings, North Weston,  
Thame, Oxfordshire, OX9 2HA, UK  
[docs@imperas.com](mailto:docs@imperas.com)



Author:	Imperas Software Limited
Version:	2.5
Filename:	iGen_Model_Generator_Introduction.doc
Project:	iGen Model Generator Introduction
Last Saved:	Tuesday, 20 April 2021
Keywords:	iGen Model Generator Introduction

## Copyright Notice

Copyright © 2021 Imperas Software Limited All rights reserved. This software and documentation contain information that is the property of Imperas Software Limited. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Imperas Software Limited, or as expressly provided by the license agreement.

## Right to Copy Documentation

The license agreement with Imperas permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any.

## Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

## Disclaimer

IMPERAS SOFTWARE LIMITED., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Table of Contents

1	Preface	4
1.1	Notation.....	4
1.2	Related Documentation.....	4
1.3	Glossary / Terminology .....	5
2	Introduction	6
2.1	Prerequisites.....	6
3	Overview	7
3.1	Obtaining & installing iGen.....	7
3.2	Using iGen .....	7
3.2.1	Simplest usage of iGen is with a Makefile .....	8
3.2.2	Batch mode .....	8
3.2.3	iGen command line arguments .....	9
3.2.4	Listing all iGen API functions .....	9
3.2.5	Getting help on an iGen API function .....	9
3.2.6	Arguments to iGen scripts .....	9
3.2.7	Interactive mode.....	10
3.3	Use of Libraries.....	10
3.4	VLNV .....	11
4	Creating Components	12
4.1	Creating a testbench / harness / platform / module.....	12
4.2	Creating a peripheral model template .....	12
4.3	Creating a processor model.....	12
4.4	Creating an extension library template .....	12
4.4.1	Beginning the extension library .....	12
4.4.2	Formal parameters .....	13
4.4.3	Commands .....	13
4.4.4	Writing the template .....	14
4.4.5	Adding a standard header.....	15
4.5	Choice of names.....	15
5	Using iGen to Create TLM2 Platforms	16
5.1	Example platform.....	16
6	iGen Command Line Arguments	18
7	iGen TCL Commands	21

# 1 Preface

The Imperas simulators can use models described in C or C++ and the models can be exported to be used in simulators and platforms using C, C++, SystemC or SystemC TLM2.

This document describes the Model Generator, iGen, which executes scripts making calls to the iGen Command API. The scripts use TCL (Tool Control Language) as input and iGen creates C templates for simulation models and plugins, and creates interfaces for SystemC TLM2 simulation and creates virtual platforms, testbenches and modules using the OP C API and SystemC TLM2.

## 1.1 Notation

**code** or **code**     Text representing code, a command or output from *iGen*.  
**keyword**           A word with special meaning.

## 1.2 Related Documentation

There are several documents available as PDF:

Getting Started

- Imperas Installation and Getting Started Guide

Interface and API

- OVP Peripheral Modeling Guide
- OVPsim Using OVP Models in SystemC TLM2.0 Platforms

Also, in your installation there is also the online iGen Function API Command reference documentation. This is correct-by-construction Doxygen-like API documentation available at:

`IMPERAS_HOME/doc/api/igen/html/index.html`

References to specific uses of iGen

- iGen Model Generator Introduction
- iGen Platform and Module Creation User Guide
- iGen Peripheral Generator User Guide

Usage of Modules and Peripherals created using iGen

- Simulation Control of Platforms and Modules User Guide
- Advanced Simulation Control of Platforms and Modules User Guide

## 1.3 Glossary / Terminology

**OP API** - OVP Platforms API - C API used for creating and controlling virtual platforms. 2nd generation API, replaces ICM API.

**iGen** - Imperas productivity tool that has a powerful script-based function API that is used to create C/C++/SystemC models and templates. Described in the iGen Model Generator Introduction, and for platforms, in the iGen Platform and Module Generator User Guide.

**OVPsim** - Simulator for Open Virtual Platforms that executes platforms and models coded in the OVP APIs

**CpuManager** - Imperas commercial simulator

**Platform / Module** – (used interchangeably) - a collection of components connected together into a level of hierarchy in a system to be simulated. This is a program in C/C++ making calls into OP API and normally compiled into a shared object/dynamically linked library and loaded by the simulator at run time.

**Testbench / Harness** - program in C/C++ making calls into OP API to connect and control OVP components. Normally linked to the simulator to provide a .exe binary that can be executed. Used to instantiate one or more platforms/modules and control their execution. The main difference, from a platform/module, is that a testbench or harness includes a call to the function `main()` and may include the command line parser.

**Root Module** - used to describe the initial platform/module that instantiates one or more platforms/modules and controls their execution. Used in the testbench / harness.

## 2 Introduction

Imperas simulation technology enables high performance simulation, debug and analysis of platforms containing multiple processors and peripheral models. The technology is designed to be extensible: you can create your own platforms, new models of processors, and other platform components using interfaces and libraries supplied by Imperas. Platform models developed using this technology can be used both with Imperas simulation products and the freely-available OVPsim platform simulator.

iGen is an Imperas productivity tool that has a powerful script based function API that is used to create C/C++/SystemC models and templates.

### 2.1 Prerequisites

Since models and platforms for use with Imperas and OVP tools are written in C, an important prerequisite is that you must be proficient in the C language.

iGen uses the TCL scripting language, so you must have some basic understanding of TCL to at least be able to write scripts that call functions.

In this document examples are shown using a shell. This can be either Linux or Windows/MSYS and so you must be familiar with these shells.

### 3 Overview

*iGen* can produce a C template for a

- hardware design (module, platform, and harness/testbench)
- processor model
- peripheral model
- extension/interception library
- Memory Model Component (MMC)

The C code of a hardware module generated by iGen can often be used as is without modification as in most cases it is purely structural and iGen creates all the necessary C code.

In most cases iGen is not used for generating simulation harnesses and testbenches. These are written directly in C (though iGen can create an initial template).

A peripheral model C template will

- create the basic model files
- create bus ports, net ports and packetnet ports to connect the peripheral to the platform
- construct memory mapped registers

The peripheral template will have no behavior but can provide empty callback functions that can be filled in by the user.

An extension library template will

- construct the basic model files
- install user commands
- parse the arguments of user commands when they are called

The extension library template will have no behavior but will provide empty functions that can be filled in by the user.

The SystemC TLM2.0 platforms create instances of OVP CPUs and peripherals and other components and have been tested with all major SystemC TLM2.0 simulators.

The SystemC TLM2.0 interfaces for OVP CPUs and peripherals have been tested with all major SystemC TLM2.0 simulators.

#### 3.1 Obtaining & installing iGen

*iGen* is available as part of the Imperas DEV and SDK packages. It is assumed you have downloaded and installed one of these.

#### 3.2 Using iGen

iGen is used mostly in a batch mode using a Makefile. It can also be used directly with a command line or also interactively, but most often it is used with the make system.

### 3.2.1 Simplest usage of iGen is with a Makefile

In an Imperas installation Makefiles are provided to control the usage of iGen. The best place to start using them is to find an example of what you are working on, and copy how it is built.

For example, to build a hardware module:

```
> cp -r Examples/PlatformConstruction/simpleCpuMemory .
> cd module
> ls
Makefile
module.op.tcl
```

Look at the files - the `Makefile` includes one of the Makefile provided in the installation, in this case the library `Makefile.module`, and the iGen script file: `module.op.tcl` which calls the iGen API functions that define the module.

Create and compile the module by typing:

```
> make
# iGen Create OP MODULE module
# Host Compiling Module obj/Windows64/module.o
# Host Linking Platform object model.dll
```

This uses iGen to generate the C and compiles it to a shared object that will be used in other modules or the testbench / harness.

Note that the Makefiles are keyed to control their operation based on the filenames provided to them. It is recommended that you use the make system and the filenames specified.

Built in file names and the appropriate Makefiles:

File Name	Makefile to use
module.op.tcl	Makefile.module
pse.tcl	Makefile.pse
platform.tlm.tcl	Makefile.TLM.igen / Makefile.TLM.platform
platform.op.tcl / platform.tcl	Makefile.platform
platform.icm.tcl	Makefile.platform (deprecated)

Note there is also `Makefile.harness` which compiles C files, e.g. `harness.c` to an executable.

### 3.2.2 Batch mode

*iGen* can be used directly in batch mode, executing scripts provided on its command line:

```
shell > igen.exe --batch model.tcl --writec model.c

      IMPERAS IGEN (32-bit) version 20150315.0
      Copyright (c) 2005-2015 by Imperas Limited.
      ALL RIGHTS RESERVED
```



```
This program is proprietary and confidential information of
Imperas Limited and may be used and disclosed only as authorized
in a license agreement controlling such use and disclosure.
...
shell >
```

Specifying an input file (`--batch`) and an output file (`--writec`) puts *iGen* into its batch mode. The input script should include commands to create just one platform / module / library / model.

### 3.2.3 iGen command line arguments

To see all the iGen command line arguments:

```
> igen.exe --help
```

For convenience, these are listed at the end of this document.

### 3.2.4 Listing all iGen API functions

To see all the iGen API functions:

```
> igen.exe --showcommands
```

Also, in your installation there is also the online iGen Function API Command reference documentation. This is correct-by-construction Doxygen-like API documentation available at:

```
IMPERAS_HOME/doc/api/igen/html/index.html
```

### 3.2.5 Getting help on an iGen API function

To get help on an iGen API function, use the `igen --apropos` command line argument with the command or word of interest:

```
> igen.exe --apropos ihwnew
> igen.exe --apropos connect
```

This will show you the built in help for the command.

### 3.2.6 Arguments to iGen scripts

If required, arguments can be passed to the TCL script using the argument `--batchargv`. This flag can be repeated, each occurrence adding another argument. The TCL script can fetch the argument using the TCL variables `$argc` and `$argv`. e.g.:

```
# myscript.tcl

if { $argc != 3 } {
    puts "Expected three arguments"
```

```
    return
} else {
    set one    [lindex $argv 0]
    set two    [lindex $argv 1]
    set three  [lindex $argv 2]
    puts "1:$one 2:$two 3:$three"
}
```

```
shell > igen --batch myscript.tcl --batchargv v1 --batchargv v2 --batchargv v3
```

```
            IMPERAS IGEN version 99999999
    Copyright (c) 2005-2015 by Imperas Limited.
            ALL RIGHTS RESERVED
```

```
This program is proprietary and confidential information of
Imperas Limited and may be used and disclosed only as authorized
in a license agreement controlling such use and disclosure.
```

```
...
1:v1 2:v2 3:v3
...
shell >
```

### 3.2.7 Interactive mode

*iGen* can also be used interactively when learning the commands and their arguments, but not as a method of building models in a development environment.

Here *ihelp* is used to list the Imperas tcl commands:

```
shell > iGen.exe
...
iGen > ihelp
...
...
ihwnew  -   Create a hardware design
...
...
```

All Imperas tcl commands begin with *i* and will accept the *-help* argument:

```
iGen > ihwnew -help
NAME:
    ihwnew  -   Begin creating a new hardware design
...
ARGUMENTS:
    -help
...
    -name <string> (mandatory)
        The VLN name of the hardware design
...
```

## 3.3 Use of Libraries

In most situations, *iGen* does not need to refer to component libraries. However, when writing a TLM platform, information is required from the component specifications, so *iGen* must locate and load the components before writing the output file(s). Note that to

do this, a simulator license will be used for a brief period. When searching for components, iGen uses the same rules as the Imperas simulator; looking in the library in the installation (referred to by the `IMPERAS_VLNV` environment variable) then using any other libraries specified by `-vlnvroot`. Note that it is an error to have a model with the same VLNV in more than one library.

### 3.4 VLNV

The SPIRIT consortium (a group of vendors and users of EDA tools that defined standards for exchange of design information), now part of Accellera ([www.accellera.org](http://www.accellera.org)) stipulates that a model should be identified by Vendor, Library, Name and Version tags. They are to be used as follows.

Vendor	The URL of the organization supplying the model (NB. not the component being modeled). Using this should guarantee unique specification of models. Large organization might consider using sub-domains, e.g. <code>performance.imperas.com</code> , <code>services.imperas.com</code> .
Library	Classification of the component e.g. processor, microcontroller, peripheral.
Name	The component identifier, often combining a functional and numeric identifier e.g. <code>UART16550</code> . Note that OVP models often package many variants as one model, using configuration parameters to select the variant at load-time. As a guideline, a model can conveniently include a variant if its external interface is similar (i.e. has a subset of identical bus and net ports).
Version	The revision (not the variant).

Imperas models are specified using VLNV; all models are shipped in a directory structure exactly matching the VLNV. e.g.

```
<processor vendor="ovpworld.org" library="processor" name="or1k" version="1.0" />
```

is located in a directory called `<root>/ovpworld.org/processor/or1k/1.0`

**NOTE:** Since a Windows file-system is case-insensitive, it is important when creating models that case is not used to distinguish VLNVs

The conformance of VLNV and directory names allows Imperas and OVP tools to access the same model hierarchy as the Imperas Simulator.

## 4 Creating Components

### 4.1 Creating a testbench / harness / platform / module

This is covered in the document:

- iGen Platform and Module Creation User Guide

### 4.2 Creating a peripheral model template

This is covered in the documents:

- iGen Peripheral Generator User Guide
- OVP Peripheral Modeling Guide

### 4.3 Creating a processor model

This is covered in the documents:

- OVP Processor Modeling Guide
- Imperas CPUGenerator Guide

### 4.4 Creating an extension library template

An extension library specification is a machine (and human) readable specification of the interface to an extension library (to create an extension library, refer to the Imperas binary Intercept Technology User Guide). The specification must define:

- VLNV of the extension.
- Name of the shared object (.dll/.so) which implements it
- Symbol of the model's attributes table
- Configuration parameters specific to this model

The following *iGen* commands are used to create an extension library template:

Command name	Action
<code>imodelnewsemihostlibrary</code>	Start a new semihost library
<code>imodeladdformal</code>	Add a formal parameter
<code>iadddocumentation</code>	Add a plain text field
<code>imodeladdsupportedprocessor</code>	Document that the library supports this processor
<code>imodeladdcommand</code>	Add a command specification.
<code>imodeladdformalargument</code>	Add a formal argument to a command.
<code>imodeladdenumeration</code>	Add an enumeration to a formal argument

#### 4.4.1 Beginning the extension library

<code>imodelnewsemihostlibrary</code>	Start a new extension library specification
<code>-name</code>	name of the new extension library
<code>-vendor</code>	VLNV vendor of new extension library
<code>-library</code>	VLNV library of new extension library
<code>-version</code>	VLNV version of new extension library

<b>imodelnewsemihostlibrary</b>	<b>Start a new extension library specification</b>
-header	Optional path to Copyright header to be included in each C file.

This command begins the specification and describes its location on the library. The optional `-header` argument allows the user to specify a text file to be included at the start of the output file.

#### 4.4.2 Formal parameters

Formal parameters are used to configure an extension library. They can be set by the simulator or the platform that loads the library. They are referenced in the platform by hierarchical name.

<b>imodeladdformal</b>	<b>Add a formal parameter to the extension library.</b>
-defaultvalue	The default value of a numeric type.
-help	
-max	Maximum allowed numeric value
-min	Minimum allowed numeric value
-name	Name of this formal parameter
-type	Data type of this parameter (address, bool, double, endian, enum, flag, float, int32, int64, list, pointer, string, stringlist, uns32, uns64)

#### 4.4.3 Commands

Extension library commands can be invoked by the simulator before simulation, or, in an interactive session, whenever the simulator is stopped. A command has a name and can be followed by optional or mandatory arguments.

<b>imodeladdcommand</b>	<b>Add a command specification.</b>
-name	name of the command
-help	When specified, the command is not executed, but help is supplied.
-class	Used by the graphical interface to determine how to present the command. Must be query, status or mode.

Their formal arguments are specified as follows:

<b>imodeladdformalargument</b>	<b>Add a formal argument to a command.</b>
-name	name of the argument
-type	Data type of this parameter (address, bool, double, endian, enum, flag, float, int32, int64, list, pointer, string, stringlist, uns32, uns64)
-mustbespecified	This is not an optional argument

If the type of a formal argument is specified as enumeration, enumeration names are added as follows:

<b>imodeladdenumeration</b>	<b>Add an enumeration to a formal</b>
-name	name of the enumeration
-formal	name of the formal with this enumerated type

<b>iadddocumentation</b>	<b>Adds a documentation field to any object</b>
-handle	hierarchical name of the object to be documented
-name	name of the documentation category (e.g. Description or Limitations)
-text	name of the formal with this enumerated type

*iadddocumentation* adds a text field to a model. Imperas uses the names *Description*, *Limitations* and *Licensing* although any names are accepted. These text fields can be introspected from the binaries of the models so that tools can probe a model and get information about it. Imperas has tools that build documentation from introspecting models. Documentation fields can also be added to ports and formals. Without *-handle*, the field is added to the root of the model. If specified, the handle should match the string returned by one of the *imodeladdxxx* commands.

```
# createSHL.tcl

imodelnewsemlibrary \
  -name      orlkNewlib \
  -vendor    ovpworld.com \
  -library   semihosting \
  -version   1.0 \
  -imagefile model \
  -attributetable modelAttrs

iadddocumentation -name Description -text "Test model"
iadddocumentation -name Limitations -text "Do not use."
```

#### 4.4.4 Writing the template

TCL commands are added to a TCL file which when supplied to iGen, which supplied to *iGen* puts it into batch mode, writing the requested template output. The template contains C code which can be compiled to create the beginnings of an extension library. The user is then expected to write more code to provide the functionality of the library.

Please refer to *Imperas\_Binary\_Intercept\_Technonology\_User\_Guide.pdf* for a description of how to write an extension library.

<b>Argument</b>	<b>File</b>
--batch <TCL input file>	File of TCL commands to construct the extension library.
--writec <output file>	File containing the body of the model, stem of other file names
--userheader <input file>	Prepend this text file to each C file (it must be legal C). Can be overridden from the TCL.
-newargparser	Use the vmi command parser API (the old method is deprecated)

In this example the name 'model' is used, producing the files listed below:

```
shell> iGen.exe          \
  --batch      extension.tcl \
  --writec     model.c      \
  --userheader companyheader.h \
```

```
--newargparser
```

Output files are specified by `-writec model.c`

File	Contains
model.c	Stub functions to be filled by the user
model.igen.c	The command parser and other code that does not need to be edited.
model.igen.h	Prototypes of generated functions.
model.macros.igen.h	Offsets of each command argument in the parsed array.

If no file extension is provided, igen adds the extension `.igen.stubs`

```
shell> igen.exe          \
--batch      extension.tcl \
--writec     model        \
--userheader companyheader.h \
--newargparser
```

File	Contains
model.igen.stubs	Stub functions to be filled by the developer
model.igen.c	The command parser.
model.igen.h	Prototypes of generated functions.
model.macros.igen.h	Offsets of each command argument in the parsed array.

Note that since the stubs file can be modified by the user, igen will not overwrite an existing file unless the `-overwrite` flag is supplied.

#### 4.4.5 Adding a standard header

Some organizations require each source file to include a standard header. Header text can be prepended to a generated file using the `--userheader` command line option.

### 4.5 Choice of names

Component instance names are converted directly to C++ where possible and modified where not. If you want your names to remain unchanged you should therefore avoid:

- C and C++ keywords.
- Names with illegal C identifier characters, e.g. '.' (period).

## 5 Using iGen to Create TLM2 Platforms

A SystemC platform generated by *iGen* uses standard files that interface between SystemC TLM2 functions and the Imperas OP API. The standard interface files are in:

```
ImpPublic/source/host/tlm  
ImpPublic/include/host/tlm
```

Code that initializes the virtual platform is in

```
tlmModule.cpp
```

Components interfaces are in

```
tlmMMC.cpp  
tlmPeripheral.cpp  
tlmProcessor.cpp
```

Port interfaces are in

```
tlmBusDynamicSlavePort.cpp  
tlmBusSlavePort.cpp  
tlmBusMasterPort.cpp  
tlmDMISlave.cpp  
tlmNetInputPort.cpp
```

The Imperas bus model (a simple address decoder) is in

```
tlmDecoder.cpp
```

Two types of memory are in

```
tlmMemory.cpp  
tlmDenseMemory.cpp
```

### 5.1 Example platform

Consider the example platform (generated by *iGen*) in

```
Examples/PlatformConstruction/SystemC_TLM2.0/platform_cpp
```

The input file, read by iGen is

```
platform.tlm.tcl
```

Output files, created by iGen are in

```
platform.*.igen.h
```

The platform is a class, inheriting `sc_core::sc_module`. It constructs SystemC instances of each component in the platform and also an instance of the `tlmPlatform` class, used to initialize the simulator. Buses are modeled using a bus decoder `tlmDecoder`. Memories use `tlmRam`. Note that memories are sparse by default so allocate native storage only as required.

Each component from the Imperas VLNV is implemented as a class. Its definition is in the VLNV source tree (not binary) next to the model. The SystemC platform creates an



instance of a model class to create an instance of the OVP model. A function is created for each instance to set OVP parameters when required.

The `tlmDecoder` decoder template class is parameterized according to the number of bus master and slave connections.

The `tlmPlatform` constructor puts the simulator in a mode which is usually suitable for SystemC simulation (the user might wish to change it) then calls each model constructor. Standard SystemC code binds initiator to acceptor sockets. *iGen* uses the SystemC TLM2 `tlm_analysis_port` to connect single-bit ports that model nets.

## 6 iGen Command Line Arguments

Group	Flag		Argument	Description
Diagnostics				
	--apropos		command	Show igen commands similar to the given argument
	--help	-h		Print list of flags
	--showcommands			Show all igen commands
	--showdomains		[root]	List the (initial) state of each memory domain
	--showenvvars			List all environment variables read by Imperas products
	--showmodeloverrides		[root]	List overrides requested by models in the platform
	--showoverrides		[root]	List all possible platform overrides
	--showsystemoverrides		[root]	List overrides in the platform provided by the simulator
Input				
	--batch	-b	filename	Execute this tcl file
	--batchargv		argument	Argument to --batch file
	--checkmodels			Load and check models when writing a platform
	--exec	-e	string	Execute this tcl string
	--modellibrary		string	Processor VLNV library
	--modelname		string	Processor VLNV name
	--modelvendor		string	Processor VLNV vendor
	--modelversion		string	Processor VLNV version
Library				
	--showlibraryextensions			List semihost and intercept libraries in the paths set by \$IMPERAS_VLNV
	--showlibrary			List models and platforms in the paths set by \$IMPERAS_VLNV
	--showlibrarymmcs			List MMCs in the paths set by \$IMPERAS_VLNV
	--showlibrarymodules			List modules in the paths set

Group	Flag		Argument	Description
				by \$IMPERAS_VLNV
	--showlibraryperipherals			List peripherals in the paths set by \$IMPERAS_VLNV
	--showlibraryplatforms			List platform executables in the paths set by \$IMPERAS_VLNV
	--showlibraryprocessors			List processors in the paths set by \$IMPERAS_VLNV
	--vlnvroot		path	Add to the search path for models
Log				
	--excludem	-x	string	Exclude message category
	--logfile		Filename	Output log file
	--nobanner			Suppress product banner
	--nowarnings	-w		Suppress warnings
	--output	-o	Filename	Output log file
	--quiet	-q		Suppress information messages
	--verbose	-v		Produce verbose output
	--version			Print version information
	--werror	-W		Treat warnings as errors
Output				
	--html			Command help written in HTML
	--icm			Write C using deprecated ICM API, rather than OP
	--newargparser			Intercept library using the new argument parser
	--op			Write C using OP API (this is the default)
	--overwrite			Overwrite any existing file
	--single			Write a single file
	--userheader		filename	Put this file at the top of generated C files
	--writec		filename	Write C model or platform
	--writetlm		filename	Write TLM interface
	--writexml		filename	Write XML from tcl
	--wrappers		Boolean	Write a TLM platform using (deprecated) wrappers
Parameters				

Group	Flag		Argument	Description
	--setParameter		string	Set a processor parameter (param=value)
	--showvariants			Show list of variants

## 7 iGen TCL Commands

iGen Command	Description
iadddocumentation	Add a documentation entry to a model, design or item.
ihwaddbridge	Add a bridge instance to a hardware design.
ihwaddbus	Add a bus instance to a hardware design.
ihwaddbusport	Add an external bus port to a hardware design.
ihwaddclp	Add a command line parser to a platform executable.
ihwaddclparg	Add an argument to the command line parser.
ihwaddenumeration	Add an enumeration value to a formal parameter
ihwaddextensionlibrary	Add an extension library to a peripheral or processor instance
ihwaddfifo	Add a FIFO instance to a hardware design.
ihwaddfifoport	Add an external FIFO port to a hardware design.
ihwaddformalmacro	Include a file (defined in a header file).
ihwaddformalparameter	Create a formal for a parameter to the current module.
ihwaddimagefile	Add a new image file to be loaded onto a bus or processor instance.
ihwaddmemory	Add a memory instance to a hardware design.
ihwaddmmc	Add a Memory Model Component instance to a hardware design.
ihwaddmodule	Add an instance of a module to this hardware design.
ihwaddnet	Add a net to a hardware design.
ihwaddnetport	Add an external net port to a hardware design.
ihwaddpacketnet	Add a packetnet instance to a hardware design.
ihwaddpacketnetport	Add an external packetnet port to a hardware design.
ihwaddperipheral	Add a peripheral instance to a hardware design.
ihwaddprocessor	Add a processor instance to a hardware design.
ihwaddprotection	Add a protected region to a bus.
ihwaddsymbolfile	Load the symbols from this image file onto a bus or

	processor instance.
ihwconnect	Create connections between component ports.
ihwnew	Create a new hardware design.
ihwsetParameter	Set a parameter on a model instance
imodeladdaddressblock	Add an address block to a slave port.
imodeladdbusmasterport	Add a new bus master port to a model.
imodeladdbuslaveport	Add a new bus slave port to a model.
imodeladdcommand	Add a command to an extension library
imodeladdelf	Add an extra elf code supported by this processor
imodeladdenumeration	Add an enumeration value to a formal attribute or argument
imodeladdextensionlibrary	Add an extension library to a peripheral or processor model
imodeladdfield	Add a bit field definition to a memory mapped register.
imodeladdfifoport	Add a new FIFO port to a model.
imodeladdformal	Add a new formal parameter to a model.
imodeladdformalargument	Add a formal argument to a command.
imodeladdformalmacro	Include a macro (defined in the PPM header file).
imodeladdlocalmemory	Add a local memory to an address block.
imodeladdmmregister	Add a memory mapped register to an address block.
imodeladdnetport	Add a new net port to a model.
imodeladdpackage	Put the model in the specified package.
imodeladdpacketnetport	Create a packetnet port.
imodeladdregister	Add a register to a model.
imodeladdreset	Add a reset value to a memory mapped register.
imodeladdsupportedprocessor	Add a supported processor to the list on an intercept library (for documentation only).
imodelnewmemory	Create a new memory model.
imodelnewmmc	Create a new memory mapped component model.
imodelnewperipheral	Create a new peripheral model.

<code>imodelnewprocessor</code>	Create a new processor model.
<code>imodelnewsemihostlibrary</code>	Create a new semihosting library.
<code>imodulnew</code>	Create a new hardware module.
<code>isetattribute</code>	deprecated function - use <code>ihwsetParameter</code>
<code>quit</code>	Finish the session

##