

Platform Specification Format Reference Manual

UG1044 (v2014.2) June 4, 2014

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
06/04/2014	2014.2	In Appendix B, Hardware Software Interface Reference , added: <ul style="list-style-type: none">• hsm::get_nodes• hsm::get_comp_params• hsm::create_node• hsm::create_comp_param
05/15/2014	2014.1	Added Appendix B, Hardware Software Interface Reference .
04/02/2014	2014.1	Vivado Design Suite release 2014.1. Initial release of document.

Table of Contents

Revision History	2
Chapter 1: Introduction	
Files	5
File and IP Naming Rules	6
Chapter 2: Microprocessor Software Specification (MSS)	
Overview	7
MSS Format	7
Global Parameters	9
Instance-Specific Parameters	9
Chapter 3: Microprocessor Library Definition (MLD)	
Overview	15
Requirements	15
Library Definition Files	15
MLD Format Specification	16
MLD Parameter Description Section	22
Design Rule Check (DRC) Section	31
Tool Generation (Generate) Section	31
Chapter 4: Microprocessor Driver Definition (MDD)	
Overview	32
Requirements	32
Driver Definition Files	33
MDD Format Specification	33
MDD Parameter Description	37
Design Rule Check (DRC) Section	43
Driver Generation (Generate) Section	44
Appendix A: Additional Resources and Legal Notices	
Xilinx Resources	45
Solution Centers	45

References 45
Please Read: Important Legal Notices 46

Appendix B: Hardware Software Interface Reference

Overview of Tcl Capabilities in HSI 47
First Class Tcl Objects and Relationships 48
Tcl Commands Listed by Category 51
Hardware TCL Commands 53
Software TCL Commands 87
HSI Property and Parameter TCL Commands 132
Other HSI Equivalent Tcl Procedures for Libgen Driver Tcl 160

Introduction

Xilinx® software tools are designed to operate in a data-driven manner. Meta-data files capture information about IPs, drivers, and software libraries used in the software tools. ASCII files also capture both the software information about your design. The set of all these meta-data formats is referred to as the Platform Specification Format or PSF.

Files

MSS - Microprocessor Software Specification

The MSS file contains directives for customizing libraries, drivers, processor, and OS.

Refer to [Chapter 2, Microprocessor Software Specification \(MSS\)](#), for more information.

MLD - Microprocessor Library Definition

An MLD file contains directives for customizing software libraries and operating systems.

Refer to [Chapter 3, Microprocessor Library Definition \(MLD\)](#) for more information.

MDD - Microprocessor Driver Definition

An MDD file contains directives for customizing software drivers.

Refer to [Chapter 4, Microprocessor Driver Definition \(MDD\)](#), for more information.

File and IP Naming Rules

File and IP names must be lower-case to ensure consistency across the following:

- OS: Linux (case-sensitive) vs. Win (case-insensitive)
- HDL: Verilog (case-sensitive) vs. VHDL (case-insensitive)

A lower-case naming convention is used in these combinations. For example: `MYCORE` and `mycore` mean two different files in Linux, whereas in Windows, they are the same.

Assembly of lower-level cores into the top-level are merged by name reference; the names must match.

Version Scheme

The format of file version levels is X.Y

- X: major revision
- Y: minor revision

Version Setting for MSS

In the body of the MSS file, add the following statement:

```
PARAMETER VERSION = 2.1.0
```

The version is specified as a literal of the form 2.1.0.

Microprocessor Software Specification (MSS)

This chapter describes the Microprocessor Software Specification (MSS) format.

Overview

The MSS file contains directives for customizing operating systems (OSs), libraries, and drivers.

MSS Format

An MSS file is case insensitive and any reference to a file name or instance name in the MSS file is also case sensitive.

Comments can be specified anywhere in the file. A pound (#) character denotes the beginning of a comment, and all characters after it, right up to the end of the line, are ignored. All white spaces are also ignored and carriage returns act as sentence delimiters.

MSS Keywords

The keywords that are used in an MSS file are as follows:

BEGIN

The keyword begins a driver, processor, or file system definition block. `BEGIN` should be followed by the `driver`, `processor`, or `filesystem` keywords.

END

This keyword signifies the end of a definition block.

PARAMETER

The MSS file has a simple `name = value` format for statements. The `PARAMETER` keyword is required before `NAME` and `VALUE` pairs. The format for assigning a value to a parameter is `parameter name = value`. If the parameter is within a `BEGIN-END` block, it is a local assignment; otherwise it is a global (system level) assignment.

Requirements

The syntax of various files that the embedded development tools use is described by the Platform Specification Format (PSF). The current PSF version is 2.1.0. The MSS file should also contain version information in the form of `parameter Version = 2.1.0`, which represents the PSF version 2.1.0.

MSS Example

An example MSS file is as follows:

```
parameter VERSION = 2.1.0

BEGIN OS
parameter PROC_INSTANCE = my_microblaze
parameter OS_NAME = standalone
parameter OS_VER = 1.0
parameter STDIN = my_uartlite_1
parameter STDOUT = my_uartlite_1
END

BEGIN PROCESSOR
parameter HW_INSTANCE = my_microblaze
parameter DRIVER_NAME = cpu
parameter DRIVER_VER = 1.0
parameter XMDSTUB_PERIPHERAL = my_jtag
END

BEGIN DRIVER
parameter HW_INSTANCE = my_intc
parameter DRIVER_NAME = intc
parameter DRIVER_VER = 1.0
END

BEGIN DRIVER
parameter HW_INSTANCE = my_uartlite_1
parameter DRIVER_VER = 1.0
parameter DRIVER_NAME = uartlite
END

BEGIN DRIVER
parameter HW_INSTANCE = my_uartlite_2
parameter DRIVER_VER = 1.0
parameter DRIVER_NAME = uartlite
END
```

```
BEGIN DRIVER
parameter HW_INSTANCE = my_timebase_wdt
parameter DRIVER_VER = 1.0
parameter DRIVER_NAME = timebase_wdt
END
```

```
BEGIN LIBRARY
parameter LIBRARY_NAME = XilMfs
parameter LIBRARY_VER = 1.0
parameter NUMBYTES = 100000
parameter BASE_ADDRESS = 0x80f00000
END
```

```
BEGIN DRIVER
parameter HW_INSTANCE = my_jtag
parameter DRIVER_NAME = uartlite
parameter DRIVER_VER = 1.0
END
```

Global Parameters

These parameters are system-specific parameters and do not relate to a particular driver, file system, or library.

PSF Version

This option specifies the PSF version of the MSS file. This option is mandatory, and is formatted as:

```
parameter VERSION = 2.1.0
```

Instance-Specific Parameters

These parameters are OS-, processor-, driver-, or library-specific. The parameters must be within a BEGIN and END block.

OS, Driver, Library, and Processor Block Parameters Summary

The following list shows the parameters that can be used in OS, driver, library and processor blocks.

PROC_INSTANCE	DRIVER_NAME
HW_INSTANCE	DRIVER_VER
OS_NAME	LIBRARY_NAME
OS_VER	LIBRARY_VER

OS, Driver, Library, and Processor Block Parameters Definitions

PROC_INSTANCE

This option is required for the OS associated with a processor instances specified in the MHS file, and is formatted as:

```
parameter PROC_INSTANCE = instance_name
```

All OSs require processor instances to be associated with the OSs. The instance name that is given must match the name specified in the MHS file.

HW_INSTANCE

This option is required for drivers associated with peripheral instances specified in the MHS file and is formatted as:

```
parameter HW_INSTANCE = instance_name
```

All drivers in software require instances to be associated with the drivers. Even a processor definition block should refer to the processor instance. The instance name that is given must match the name specified in the BD file.

OS_NAME

This option is needed for processor instances that have OSs associated with them and is formatted as:

```
parameter OS_NAME = standalone
```

OS_VER

The OS version is set using the OSVER option and is formatted as:

```
parameter OS_VER = 1.0
```

This version is specified in the following format: x.y, where x and y are digits. This is translated to the OS directory searched as follows:

```
OS_NAME_vx_y
```

The MLD (Microprocessor Library Definition) files needed for each OS should be named OS_NAME.mld and should be present in a subdirectory `data/` within the driver directory. Refer to [Chapter 3, Microprocessor Library Definition \(MLD\)](#), for more information.

DRIVER_NAME

This option is needed for peripherals that have drivers associated with them and is formatted as:

```
parameter DRIVER_NAME = uartlite
```

Library Generator copies the driver directory specified to the `OUTPUT_DIR/processor_instance_name/libsrc` directory and compiles the drivers using makefiles provided.

DRIVER_VER

The driver version is set using the `DRIVER_VER` option, and is formatted as:

```
parameter DRIVER_VER = 1.0
```

This version is specified in the following format: `x.y`, where `x` and `y` are digits. This is translated to the driver directory searched as follows:

```
DRIVER_NAME_vx_y
```

The MDD (Microprocessor Driver Definition) files needed for each driver should be named `DRIVER_NAME_v2_1_0.mdd` and should be present in a subdirectory `data/` within the driver directory. Refer to [Chapter 4, Microprocessor Driver Definition \(MDD\)](#), for more information.

LIBRARY_NAME

This option is needed for libraries, and is formatted as:

```
parameter LIBRARY_NAME = xilmfs
```

The tool copies the library directory specified in the `OUTPUT_DIR/processor_instance_name/libsrc` directory and compiles the libraries using makefiles provided.

LIBRARY_VER

The library version is set using the `LIBRARY_VER` option and is formatted as:

```
parameter LIBRARY_VER = 1.0
```

This version is specified in the following format: `x.y`, where `x` and `y` are digits. This is translated to the library directory searched by the tool as follows:

```
LIBRARY_NAME_vx_y
```

The MLD (Microprocessor Library Definition) files needed for each library should be named `LIBRARY_NAME.mld` and should be present in a subdirectory `data/` within the library directory. See [Chapter 3, Microprocessor Library Definition \(MLD\)](#), for more information.

MDD/MLD Specific Parameters

Parameters specified in the MDD/MLD file can be overwritten in the MSS file and formatted as

```
parameter PARAM_NAME = PARAM_VALUE
```

See [Chapter 3, Microprocessor Library Definition \(MLD\)](#), and [Chapter 4, Microprocessor Driver Definition \(MDD\)](#), for more information.

OS-Specific Parameters Summary

The following list identifies all the parameters that can be specified only in an OS definition block.

STDIN

Identify the standard input device with the `STDIN` option, which is formatted as:

```
parameter STDIN = instance_name
```

STDOUT

Identify the standard output device with the `STDOUT` option, which is formatted as:

```
parameter STDOUT = instance_name
```

Example: MSS Snippet Showing OS Options

```
BEGIN OS
parameter PROC_INSTANCE = my_microblaze
parameter OS_NAME = standalone
parameter OS_VER = 1.0
parameter STDIN = my_uartlite_1
parameter STDOUT = my_uartlite_1
END
```

Processor-Specific Parameter Summary

Following is a list of all of the parameters that can be specified only in a processor definition block.

[XMDSTUB_PERIPHERAL](#)
[COMPILER](#)
[ARCHIVER](#)
[COMPILER_FLAGS](#)
[EXTRA_COMPILER_FLAGS](#)

Processor-Specific Parameter Definitions

XMDSTUB_PERIPHERAL

The peripheral that is used to handle the XMDStub should be specified in the `XMDSTUB_PERIPHERAL` option. This is useful for the MicroBlaze™ processor only, and is formatted as follows:

```
parameter XMDSTUB_PERIPHERAL = instance_name
```

COMPILER

This option specifies the compiler used for compiling drivers and libraries. The compiler defaults to `mb-gcc` or `powerpc-eabi-gcc` depending on whether the drivers are part of the MicroBlaze™ processor or PowerPC® processor instance. Any other compatible compiler can be specified as an option, and should be formatted as follows:

```
parameter COMPILER = dcc
```

This example denotes the Diab compiler as the compiler to be used for drivers and libraries.

ARCHIVER

This option specifies the utility to be used for archiving object files into libraries. The archiver defaults to `mb-ar` or `powerpc-eabi-ar` depending on whether or not the drivers are part of the MicroBlaze or PowerPC processor instance. Any other compatible archiver can be specified as an option, and should be formatted as follows:

```
parameter ARCHIVER = ar
```

This example denotes the archiver `ar` to be used for drivers and libraries.

COMPILER_FLAGS

This option specifies compiler flags to be used for compiling drivers and libraries. If the option is not specified, the tool automatically uses platform and processor-specific options. This option should not be specified in the MSS file if the standard compilers and archivers are used. The `COMPILER_FLAGS` option can be defined in the MSS if there is a need for custom compiler flags that override generated flags. The `EXTRA_COMPILER_FLAGS` option is recommended if compiler flags must be appended to the ones already generated. Format this option as follows:

```
parameter COMPILER_FLAGS = ""
```

EXTRA_COMPILER_FLAGS

This option can be used whenever custom compiler flags need to be used in addition to the automatically generated compiler flags, and should be formatted as follows:

```
parameter EXTRA_COMPILER_FLAGS = -g
```

This example specifies that the drivers and libraries must be compiled with debugging symbols in addition to the generated COMPILER_FLAGS.

Example MSS Snippet Showing Processor Options

```
BEGIN PROCESSOR
parameter HW_INSTANCE = my_microblaze
parameter DRIVER_NAME = cpu
parameter DRIVER_VER = 1.00.a
parameter DEFAULT_INIT = xmdstub
parameter XMDSTUB_PERIPHERAL = my_jtag
parameter STDIN = my_uartlite_1
parameter STDOUT = my_uartlite_1
parameter COMPILER = mb-gcc
parameter ARCHIVER = mb-ar
parameter EXTRA_COMPILER_FLAGS = -g -O0
parameter OS = standalone
END
```

Microprocessor Library Definition (MLD)

This chapter describes the Microprocessor Library Definition (MLD) format, Platform Specification Format 2.1.0.

Overview

An MLD file contains directives for customizing software libraries and generating Board Support Packages (BSP) for Operating Systems (OS). This document describes the MLD format and the parameters that can be used to customize libraries and OSs.

Requirements

Each OS and library has an MLD file and a Tcl (Tool Command Language) file associated with it. The MLD file is used by the Tcl file to customize the OS or library, depending on different options in the MSS file. For more information on the MSS file format, see [Chapter 2, Microprocessor Software Specification \(MSS\)](#).

The OS and library source files and the MLD file for each OS and library must be located at specific directories to find the files and libraries.

Library Definition Files

Library Definition involves defining Data Definition (MLD) and a Data Generation (Tcl) files.

Data Definition File

The MLD file (named as `<library_name>.mld` or `<os_name>.mld`) contains the configurable parameters. A detailed description of the various parameters and the MLD format is described in MLD Parameter Description Section, page 20.

Data Generation File

The second file (named as `<library_name>.tcl` or `<os_name>.tcl`, with the filename being the same as the MLD filename) uses the parameters configured in the MSS file for the OS or library to generate data.

Data generated includes, but is not limited to, generation of header files, C files, running DRCs for the OS or library and generating executables. The Tcl file includes procedures that are called by the tool at various stages of its execution. Various procedures in a Tcl file include: `DRC` (the name of the DRC given in the MLD file); `generate` (tool defined procedure) called after OS and library files are copied; `post_generate` (tool defined procedure) called after `generate` has been called on all OSs, drivers, and libraries; and `execs_generate` (a tool-defined procedure) called after the BSPs, libraries, and drivers have been generated.

Note: An OS/library does not require a data generation file (Tcl file).

MLD Format Specification

The MLD format specification involves the MLD file Format specification and the Tcl file Format specification. The following subsections describe the MLD.

MLD File Format Specification

The MLD file format specification involves the description of parameters defined in the Parameter Description section.

Parameter Description Section

This data section describes configurable parameters in an OS/library. The format used to describe this section is discussed in [MLD Parameter Description Section, page 22](#).

Tcl File Format Specification

Each OS and library has a Tcl file associated with the MLD file. This Tcl file has the following:

DRC Section

This section contains Tcl routines that validate your OS and library parameters for consistency.

Generation Section

This section contains Tcl routines that generate the configuration header and C files based on the library parameters.

Examples

This section explains the MLD format through an example MLD file and its corresponding Tcl file.

Example: MLD File for a Library

The following is an example of an MLD file for the xilmfs library.

```
OPTION psf_version = 2.1.0 ;
```

`OPTION` is a keyword identified by the tool. The option name following the `OPTION` keyword is a directive to the tool to do a specific action.

The `psf_version` of the MLD file is defined to be 2.1 in this example. This is the only option that can occur before a `BEGIN LIBRARY` construct now.

```
BEGIN LIBRARY xilmfs
```

The `BEGIN LIBRARY` construct defines the start of a library named `xilmfs`.

```
OPTION NAME = uartlite ;
OPTION VERSION = 4.0 ;
```

The `NAME` option indicates the name of the driver. The `VERSION` option indicates the version of the driver.

```
OPTION OS_TYPE = DTS;
```

The `OS_TYPE` option indicates the type of OS. When we create software design for `DTS os_type`, Tool maps the peripherals with the drivers which supports `DTS os_type`.

```
OPTION DRC = mfsdrc ;
OPTION COPYFILES = all;
```

The `COPYFILES` option indicates the files to be copied for the library. The `DRC` option specifies the name of the Tcl procedure that the tool invokes while processing this library. The `mfs_drc` is the Tcl procedure in the `xilmfs.tcl` file that would be invoked while processing the `xilmfs` library.

```
PARAM NAME = numbytes, DESC = "Number of Bytes", TYPE = int, DEFAULT = 100000, DRC =
drc_numbytes ;
PARAM NAME = base_address, DESC = "Base Address", TYPE = int, DEFAULT = 0x10000, DRC
= drc_base_address ;
PARAM NAME = init_type, DESC = "Init Type", TYPE = enum, VALUES = ("New file
system"=MFSINIT_NEW, "MFS Image"=MFSINIT_IMAGE, "ROM Image"=MFSINIT_ROM_IMAGE),
DEFAULT = MFSINIT_NEW ;
PARAM NAME = need_utils, DESC = "Need additional Utilities?", TYPE = bool, DEFAULT
= false ;
```

`PARAM` defines a library parameter that can be configured. Each `PARAM` has the following properties associated with it, whose meaning is self-explanatory: `NAME`,

DESC, TYPE, DEFAULT, RANGE, DRC. The property VALUES defines the list of possible values associated with an ENUM type.

```
BEGIN INTERFACE file
  PROPERTY HEADER="xilmfs.h" ;
  FUNCTION NAME=open, VALUE=mfs_file_open ;
  FUNCTION NAME=close, VALUE=mfs_file_close ;
  FUNCTION NAME=read, VALUE=mfs_file_read ;
  FUNCTION NAME=write, VALUE=mfs_file_write ;
  FUNCTION NAME=lseek, VALUE=mfs_file_lseek ;
END INTERFACE
```

An Interface contains a list of standard functions. A library defining an interface should have values for the list of standard functions. It must also specify a header file where all the function prototypes are defined.

PROPERTY defines the properties associated with the construct defined in the BEGIN construct. Here HEADER is a property with value "xilmfs.h", defined by the file interface. FUNCTION defines a function supported by the interface.

The open, close, read, write, and lseek functions of the file interface have the values mfs_file_open, mfs_file_close, mfs_file_read, mfs_file_write, and mfs_file_lseek. These functions are defined in the header file xilmfs.h.

```
BEGIN INTERFACE filesystem
```

BEGIN INTERFACE defines an interface the library supports. Here, file is the name of the interface.

```
PROPERTY HEADER="xilmfs.h" ;
FUNCTION NAME=cd, VALUE=mfs_change_dir ;
FUNCTION NAME=opendir, VALUE=mfs_dir_open ;
FUNCTION NAME=closedir, VALUE=mfs_dir_close ;
FUNCTION NAME=readdir, VALUE=mfs_dir_read ;
FUNCTION NAME=deletedir, VALUE=mfs_delete_dir ;
FUNCTION NAME=pwd, VALUE=mfs_get_current_dir_name ;
FUNCTION NAME=rename, VALUE=mfs_rename_file ;
FUNCTION NAME=exists, VALUE=mfs_exists_file ;
FUNCTION NAME=delete, VALUE=mfs_delete_file ;
END INTERFACE
```

```
END LIBRARY
```

END is used with the construct name that was used in the BEGIN statement. Here, END is used with INTERFACE and LIBRARY constructs to indicate the end of each of INTERFACE and LIBRARY constructs.

Example: Tcl File of a Library

The following is the xilmfs.tcl file corresponding the xilmfs.mld file described in the previous section. The mfs_drc procedure would be invoked for the xilmfs library while

running DRCs for libraries. The generate routine generates constants in a header file and a c file for the xilmfs library based on the library definition segment in the MSS file.

```

proc mfs_drc {lib_handle} {
    puts "MFS DRC ..."
}
proc mfs_open_include_file {file_name} {
    set filename [file join "../..../include/" $file_name]
    if {[file exists $filename]} {
        set config_inc [open $filename a]
    } else {
        set config_inc [open $filename a]
        xprint_generated_header $config_inc "MFS Parameters"
    }
    return $config_inc
}
proc generate {lib_handle} {

    puts "MFS generate ..."
    file copy "src/xilmfs.h" "../..../include/xilmfs.h"
    set conffile [mfs_open_include_file "mfs_config.h"]
    puts $conffile "#ifndef _MFS_CONFIG_H"
    puts $conffile "#define _MFS_CONFIG_H"
    set need_utils [xget_value $lib_handle "PARAMETER" "need_utils"]
    puts $conffile "#include <xilmfs.h>"
    set value [xget_value $lib_handle "PARAMETER" "numbytes"]
    puts $conffile "#define MFS_NUMBYTES $value"
    set value [xget_value $lib_handle "PARAMETER" "base_address"]
}

```

Example: MLD File for an OS

An example of an MLD file for the standalone OS is given below:

```
OPTION psf_version = 2.1.0 ;
```

OPTION is a keyword identified by the tool. The option name following the OPTION keyword is a directive to the tool to do a specific action. Here the psf_version of the MLD file is defined to be 2.1. This is the only option that can occur before a BEGIN OS construct at this time.

```
BEGIN OS standalone
```

The BEGIN OS construct defines the start of an OS named standalone.

```
OPTION DESC = "Generate standalone BSP";
OPTION COPYFILES = all;
```

The DESC option gives a description of the MLD. The COPYFILES option indicates the files to be copied for the OS.

```
PARAM NAME = stdin, DESC = "stdin peripheral ", TYPE = peripheral_instance,
REQUIRES_INTERFACE = stdin, DEFAULT = none;
PARAM NAME = stdout, DESC = "stdout peripheral ", TYPE = peripheral_instance,
REQUIRES_INTERFACE = stdout, DEFAULT = none ;
PARAM NAME = need_xilmalloc, DESC = "Need xil_malloc?", TYPE = bool, DEFAULT = false
;
```

PARAM defines an OS parameter that can be configured. Each PARAM has the following, associated properties: NAME, DESC, TYPE, DEFAULT, RANGE, DRC. The property VALUES defines the list of possible values associated with an ENUM type.

```
END OS
```

END is used with the construct name that was used in the BEGIN statement. Here END is used with OS to indicate the end of OS construct.

Example: Tcl File of an OS

The following is the `standalone.tcl` file corresponding to the `standalone.mld` file described in the previous section. The generate routine generates constants in a header file and a c file for the `xilmfs` library based on the library definition segment in the `MSS` file.

```

proc generate {os_handle} {
    global env

    set need_config_file "false"

    # Copy over the right set of files as src based on processor type
    set sw_proc_handle [get_sw_processor]
    set hw_proc_handle [get_cells [get_property HW_INSTANCE $sw_proc_handle] ]
    set proctype [get_property IP_NAME $hw_proc_handle]
    set procname [get_property NAME $hw_proc_handle]

    set enable_sw_profile [get_property CONFIG.enable_sw_intrusive_profiling $os_handle]
    set mb_exceptions false

    switch $proctype {
        "microblaze" {
            foreach entry [glob -nocomplain [file join $mbsrcdir *]] {
                # Copy over only files that are not related to exception handling.
                # All such files have exception in their names.
                file copy -force $entry "./src/"
            }
            set need_config_file "true"
            set mb_exceptions [mb_has_exceptions $hw_proc_handle]
        }
        "ps7_cortexa9" {
            set procdrv [get_sw_processor]
            set compiler [get_property CONFIG.compiler $procdrv]
            if {[string compare -nocase $compiler "armcc"] == 0} {
                set ccdir "./src/cortexa9/armcc"
            } else {
                set ccdir "./src/cortexa9/gcc"
            }
            foreach entry [glob -nocomplain [file join $cortexa9srcdir *]] {
                file copy -force $entry "./src/"
            }
            foreach entry [glob -nocomplain [file join $ccdir *]] {
                file copy -force $entry "./src/"
            }
            file delete -force "./src/armcc"
            file delete -force "./src/gcc"
            if {[string compare -nocase $compiler "armcc"] == 0} {
                file delete -force "./src/profile"
                set enable_sw_profile "false"
            }
            set file_handle [xopen_include_file "xparameters.h"]
            puts $file_handle "#include \"xparameters_ps.h\""
            puts $file_handle ""
            close $file_handle
        }
        "default" {puts "unknown processor type $proctype\n"}
    }
}

```

MLD Parameter Description Section

This section gives a detailed description of the constructs used in the MLD file.

Conventions

[] Denotes optional values.

<>Value substituted by the MLD writer.

Comments

Comments can be specified anywhere in the file. A “#” character denotes the beginning of a comment and all characters after the “#” right up to the end of the line are ignored. All white spaces are also ignored and semi-colons with carriage returns act as sentence delimiters.

OS or Library Definition

The OS or library section includes the OS or library name, options, dependencies, and other global parameters, using the following syntax:

```

OPTION psf_version = <psf version number>
BEGIN LIBRARY/OS <library/os name>
  [OPTION drc = <global drc name>]
  [OPTION depends = <list of directories>]
  [OPTION help = <help file>]
  [OPTION requires_interface = <list of interface names>]
  PARAM <parameter description>
  [BEGIN CATEGORY <name of category>
    <category description>
  END CATEGORY]
  BEGIN INTERFACE <interface name>
    .....
  END INTERFACE]
END LIBRARY/OS
  
```

MLD or MDD Keyword Summary

The keywords that are used in an MLD or MDD file are as follows:

BEGIN	APP_LINKER_FLAGS	CATEGORY
END	BSP	PARAM
PSF_VERSION	OS_STATE	PROPERTY
DRC	REQUIRES_INTERFACE	NAME
OPTION	REQUIRES_OS	VERSION
OS	HELP	DESC
COPYFILES	DEP	TYPE
DEPENDS	INTERFACE	DEFAULT
SUPPORTED_PERIPHERALS	HEADER	GUI_PERMIT
LIBRARY_STATE	FUNCTION	ARRAY
APP_COMPILER_FLAGS		

MLD or MDD Keyword Definitions

The keywords that are used in an MLD or MDD file are as follows:

Note: The keyword *ARRAY* can only be used in MLD files. It is not allowed for MDD files.

BEGIN

The **BEGIN** keyword begins one of the following: *os*, *library*, *driver*, *block*, *category*, *interface*, *array*.

END

The **END** keyword signifies the end of a definition block.

PSF_VERSION

Specifies the PSF version of the library.

DRC

Specifies the DRC function name. This is the global DRC function, which is called by the GUI configuration tool or the command-line tool. This DRC function is called once you enter all the parameters and MLD or MDD writers can verify that a valid OS, library, or driver can be generated with the given parameters.

OPTION

Specifies that the name following the keyword `option` is an option to the GUI tools.

OS

Specifies the type of OS. If it is not specified, then OS is assumed as standalone type of OS.

COPYFILES

Specifies the files to be copied for the OS, library, or driver. If `ALL` is used, then the tool copies all the OS, library, or driver files.

DEPENDS

Specifies the list of directories that needs to be compiled before the OS or library is built.

SUPPORTED_PERIPHERALS

Specifies the list of peripherals supported by the OS. The values of this option can be specified as a list, or as a regular expression. For example:

```
option supported_peripherals = (microblaze)
```

Indicates that the OS supports all versions of `microblaze`. Regular expressions can be used in specifying the peripherals and versions. The regular expression (RE) is constructed as follows:

Single-Character REs

- Any character that is not a special character (to be defined) matches itself.
- A backslash (followed by any special character) matches the literal character itself. That is, this "escapes" the special character.
- The special characters are: `+ * ? . [] ^ $`
- The period (`.`) matches any character except the new line. For example, `.umpty` matches both `Humpty` and `Dumpty`.
- A set of characters enclosed in brackets (`[]`) is a one-character RE that matches any of the characters in that set. For example, `[akm]` matches either an "a", "k", or "m". A range of characters can be indicated with a dash. For example, `[a-z]` matches any lower-case letter. However, if the first character of the set is the caret (`^`), then the RE matches any character except those in the set. It does not match the empty string. Example: `[^akm]` matches any character except "a", "k", or "m". The caret loses its special meaning if it is not the first character of the set.

Multi-Character REs

- A single-character RE followed by an asterisk (`*`) matches zero or more occurrences of the RE. Thus, `[a-z]*` matches zero or more lower-case characters.
- A single-character RE followed by a plus (`+`) matches one or more occurrences of the RE. Thus, `[a-z]+` matches one or more lower-case characters.
- A question mark (`?`) is an optional element. The preceding RE can occur zero or once in the string -- no more. Thus, `xy?z` matches either `xyz` or `xz`.

- The concatenation of REs is a RE that matches the corresponding concatenation of strings. For example, `[A-Z][a-z]*` matches any capitalized word.
- For example, the following matches a version of the axidma:

```
OPTION supported_peripherals = (axi_dma_v[3-9]_[0-9][0-9]_[a-z]
axi_dma_v[3-9]_[0-9]);
```

LIBRARY_STATE

Specifies the state of the library. Following is the list of values that can be assigned to `LIBRARY_STATE`:

ACTIVE

An active library. By default the value of `LIBRARY_STATE` is `ACTIVE`.

DEPRECATED

This library is deprecated.

OBSOLETE

This library is obsolete and will not be recognized by any tools. Tools error out on an obsolete library and a new library should be used instead.

APP_COMPILER_FLAGS

This option specifies what compiler flags must be added to the application when using this library. For example:

```
OPTION APP_COMPILER_FLAGS = "-D MYLIBRARY"
```

The GUI tools can use this option value to automatically set compiler flags automatically for an application.

APP_LINKER_FLAGS

This option specifies that linker flags must be added to the application when using a particular library or OS. For example:

```
OPTION APP_LINKER_FLAGS = "-lxilkernel"
```

The GUI tools can use this value to set linker flags automatically for an application.

BSP

Specifies a boolean keyword option that can be provided in the MLD file to identify when an OS component is to be treated as a third party BSP. For example

```
OPTION BSP = true;
```

This indicates that the SDK tools will offer this OS component as a board support package. If set to `false`, the component is handled as a native embedded software platform.

OS_STATE

Specifies the state of the operating system (OS). Following is the list of values that can be assigned to OS_STATE:

ACTIVE

This is an active OS. By default the value of OS_STATE is ACTIVE.

DEPRECATED

This OS is deprecated.

OBSOLETE

This OS is obsolete and will not be recognized by the tools. Tools error out on an obsolete OS and a new OS must be specified.

REQUIRES_INTERFACE

Specifies the interfaces that must be provided by other OSs, libraries, or drivers in the system.

REQUIRES_OS

Specifies the list of OSs with which the specified library will work. For example:

```
OPTION REQUIRES_OS = (standalone xilkernel_v4_[0-9][0-9])
```

The GUI tools use this option value to determine which libraries are offered for a given operating system choice. The values in the list can be regular expressions as shown in the example.

Note: This option must be used on libraries only.

HELP

Specifies the HELP file that describes the OS, library, or driver.

DEP

Specifies the condition that must be satisfied before processing an entity. For example to include a parameter that is dependent on another parameter (defined as a DEP, for dependent, condition), the DEP condition should be satisfied. Conditions of the form (operand1 OP operand2) are the only supported conditions.

INTERFACE

Specifies the interfaces implemented by this OS, library, or driver. It describes the interface functions and header files used by the library/driver.

```
BEGIN INTERFACE <interface name>
  OPTION DEP=<list of dependencies>;
  PROPERTY HEADER=<name of header file where the function is declared>;
  FUNCTION NAME=<name of interface function>, VALUE=<function name of library/driver
implementation> ;
END INTERFACE
```

HEADER

Specifies the HEADER file in which the interface functions would be defined.

FUNCTION

Specifies the FUNCTION implemented by the interface. This is a name-value pair in which name is the interface function name and value is the name of the function implemented by the OS, library, or driver.

CATEGORY

Defines an unconditional block. This block gets included based on the default value of the category or if included in the MSS file.

```
BEGIN CATEGORY <category name>
  PARAM name = <category name>, DESC=<param description>, TYPE=<category type>,
DEFAULT=<default>, GUI_PERMIT=<value>, DEP = <condition>
  OPTION DEPENDS=<list of dependencies>, DRC=<drc name>, HELP=<help file>;
  < parameters or categories description>
END CATEGORY
```

Nested categories are not supported through the syntax that specifies them. A category is selected in a MSS file by specifying the category name as a parameter with a boolean value TRUE. A category must have a PARAM with category name.

PARAM

The MLD file has a simple *name = value* format for most statements. The PARAM keyword is required before every such NAME, VALUE pairs. The format for assigning a value to a parameter is *param name = <name>, default = value*. The PARAM keyword specifies that the parameter can be overwritten in the MSS file.

PROPERTY

Specifies the various properties of the entity defined with a BEGIN statement

NAME

Specifies the name of the entity in which it was defined. (Examples: `param` and `property`.) It also specifies the name of the library if it is specified with option.

VERSION

Specifies the version of the library.

DESC

Describes the entity in which it was defined. (Examples: `param` and `property`.)

TYPE

Specifies the type for the entity in which it was defined. (Example: `param`.) The following types are supported:

bool

Boolean (true or false).

int

Integer

string

String value within " " (quotes).

enum

List of possible values that a parameter can take.

library

Specify other library that is needed for building the library/driver.

peripheral_instance

Specify other hardware drivers that is needed for building the library.

DEFAULT

Specifies the default value for the entity in which it was defined.

GUI_PERMIT

Specifies the permissions for modification of values. The following permissions exist:

NONE

The value cannot be modified at all.

ADVANCED_USER

The value can be modified by all. The SDK GUI does not display this value by default. This is displayed only for the advanced option in the GUI.

ALL_USERS

The value can be modified by all. The SDK GUI displays this value by default. This is the default value for all the values. If `GUI_PERMIT = NONE`, the category is always active.

ARRAY

ARRAY can have any number of PARAMs, and only PARAMs. It cannot have CATEGORY as one of the fields of an array element. The size of the array can be defined as one of the properties of the array. An array with default values specified in the default property leads to its size property being initialized to the number of values. If there is no size property defined, a size property is created before initializing it with the default number of elements. Each parameter in the array can have a default value. In cases in which size is defined with an integer value, an array of size elements would be created wherein the value of each element would be the default value of each of the parameters.

```

BEGIN ARRAY <array name>
  PROPERTY desc = <array description> ;
  PROPERTY size = <size of the array>;
  PROPERTY default = <List of Values for each element based on the size of the array>
  # array field description as parameters
  PARAM name = <name of parameter>, desc = "description of param", type = <type of
param>, default = <default value>
  .....
END ARRAY
  
```

Design Rule Check (DRC) Section

```
proc mydrc { handle } {  
  
}
```

The DRC function could be any Tcl code that checks your parameters for correctness. The DRC procedures can access (read-only) the Platform Specification Format database (which the tool builds using the MHS and the MSS files) to read the parameter values that you set. The *handle* is associated with the current library in the database. The DRC procedure can get the OS and library parameters from this handle. It can also get any other parameter from the database by first requesting a handle and using the handle to get the parameters.

For errors, DRC procedures call the Tcl error command `error "error msg"` that displays in an error dialog box.

For warnings, DRC procedures return a string value that can be printed on the console.

On success, DRC procedures return without any value.

Tool Generation (Generate) Section

```
proc mygenerate { handle } {  
  
}
```

`Generate` could be any Tcl code that reads your parameters and generates configuration files for the OS or library. The configuration files can be C files, Header files, Makefiles, etc. The generate procedures can access (read-only) the Platform Specification Format database (which the tool builds using the MSS files) to read the parameter values of the OS or library that you set. The *handle* is a handle to the current OS or library in the database. The generate procedure can get the OS or library parameters from this handle. It can also get any other parameter from the database by first requesting a handle and using the handle to get the parameter.

Microprocessor Driver Definition (MDD)

This chapter describes the Microprocessor Driver Definition (MDD) format, Platform Specification Format 2.1.0.

Overview

An MDD file contains directives for customizing software drivers. This document describes the MDD format and the parameters that can be used to customize drivers.

Requirements

Each device driver has an MDD file and a Tcl file associated with it. The MDD file is used by the Tcl file to customize the driver, depending on different options configured in the MSS file. For more information on the MSS file format, refer to [Chapter 2, Microprocessor Software Specification \(MSS\)](#).

The driver source files and the MDD file for each driver must be located at specific directories in order to find the files and the drivers.

Driver Definition Files

Driver Definition involves defining a Data Definition file (MDD) and a Data Generation file (Tcl file).

- **Data Definition File:** The MDD file (`<driver_name>.mdd`) contains the configurable parameters. A detailed description of the parameters and the MDD format is described in [MDD Parameter Description, page 37](#).
- **Data Generation File:** The second file (`<driver_name>.tcl`, with the filename being the same as the MDD filename) uses the parameters configured in the MDD file for the driver to generate data. Data generated includes but not limited to generation of header files, C files, running DRCs for the driver and generating executables. The Tcl file includes procedures that are called by the tool at various stages of its execution.

Various procedures in a Tcl file includes: the DRC (name of the DRC given in the MDD file), generate (tool defined procedure) called after driver files are copied, post_generate (tool defined procedure) called after generate has been called on all drivers and libraries, and execs_generate called after the libraries and drivers have been generated.

Note: A driver does not require the data generation file (Tcl file).

MDD Format Specification

The MDD format specification involves the MDD file Format specification and the Tcl file Format specification which are described in the following subsections.

MDD File Format Specification

The MDD file format specification describes the parameters defined in the Parameter Description section. This data section describes configurable parameters in a driver. The format used to describe these parameters is discussed in [MDD Parameter Description, page 37](#).

Tcl File Format Specification

Each driver has a Tcl file associated with the MDD file. This Tcl file has the following sections:

DRC Section

This section contains Tcl routines that validate your driver parameters for consistency.

Generation Section

This section contains Tcl routines that generate the configuration header and C files based on the driver parameters

Example

This section explains the MDD format through an example of an MDD file and its corresponding Tcl file.

MDD: File Example

The following is an example of an MDD file for the uartlite driver:

```
OPTION psf_version = 2.1;
```

`OPTION` is a keyword identified by the tool. The option name following the `OPTION` keyword is a directive to the tool to do a specific action. Here the `psf_version` of the MDD file is defined as 2.1. This is the only option that can occur before a `BEGIN DRIVER` construct.

```
BEGIN DRIVER uartlite
```

The `BEGIN DRIVER` construct defines the start of a driver named `uartlite`.

```
PARAM NAME = level, DESC = "Driver Level", TYPE = int, DEFAULT = 0, RANGE = (0, 1);
```

`PARAM` defines a driver parameter that can be configured. Each `PARAM` has the following properties associated with it: `NAME`, `DESC`, `TYPE`, `DEFAULT`, `RANGE`.

```
DTGPARAM name = "timeout-sec", type = int, default = 10;
```

`DTGPARAM` defines device tree specific driver parameter that can be configured. Each `DTGPARAM` has the following properties associated with it: `NAME`, `DESC`, `TYPE`, `DEFAULT`. It is similar to `PARAM` but specific to device-tree.

```
BEGIN BLOCK, DEP = (level = 0)
```

`BEGIN BLOCK, DEP` allows conditional inclusion of a set of parameters subject to a condition fulfillment. The condition is given by the `DEP` construct. Here the set of

parameters defined inside the BLOCK would be processed by the tool only when "level" parameter has a value 0.

```
OPTION NAME = uartlite ;
OPTION VERSION = 4.0 ;
```

The NAME option indicates the name of the driver. The VERSION option indicates the version of the driver.

```
OPTION SUPPORTED_OS_TYPES = (DTS);
```

The SUPPORTED_OS_TYPES option specifies the list of OS types that it supports. When we create software design for DTS OS_TYPES, Tool maps the peripherals with the drivers which supports DTS OS_TYPES.

```
OPTION DEPENDS = (common_v1_0);
OPTION COPYFILES = (xuartlite_1.c xuartlite_1.h Makefile);
OPTION DRC = uartlite_drc;
```

The DEPENDS option specifies that the driver depends on the sources of a directory named common_v1_00_a. The area for searching the dependent directory is decided by the tool. The COPYFILES option indicates the files to be copied for a "level" 0 uartlite driver. The DRC option specifies the name of the Tcl procedure that the tool invokes while processing this driver. The uartlite_drc is the Tcl procedure in the uartlite.tcl file that would be invoked while processing the uartlite driver.

```
BEGIN INTERFACE stdin
```

BEGIN INTERFACE defines an interface the driver supports. The interface name is stdin.

```
PROPERTY header = xuartlite_1.h;
FUNCTION name = inbyte, value = XUartLite_RecvByte;
END INTERFACE
```

An Interface contains a list of standard functions. A driver defining an interface should have values for the list of standard functions. It must also specify a header file in which all the function prototypes are defined.

PROPERTY defines the properties associated with the construct defined in the BEGIN construct. The header is a property with the value xuartlite_1.h, defined by the stdin interface. FUNCTION defines a function supported by the interface. The inbyte function of the stdin interface has the value XUartLite_RecvByte. This function is defined in the header file xuartlite_1.h.

```
BEGIN INTERFACE stdout
PROPERTY header = xuartlite_1.h;
FUNCTION name = outbyte, value = XUartLite_SendByte;
END INTERFACE
BEGIN INTERFACE stdio
PROPERTY header = xuartlite_1.h;
```

```

        FUNCTION name = inbyte, value = XUartLite_RecvByte;
        FUNCTION name = outbyte, value = XUartLite_SendByte;
    END INTERFACE
BEGIN ARRAY interrupt_handler
    PROPERTY desc = "Interrupt Handler Information";
    PROPERTY size = 1, permit = none;
    PARAM name = int_handler, default = XIntc_DefaultHandler, desc = "Name of
Interrupt Handler", type = string;
PARAM name = int_port, default = Interrupt, desc = "Interrupt pin associated with the
interrupt handler", permit = none;
END ARRAY

```

The ARRAY construct defines an array of parameters. The `interrupt_handler` is the name of the array. The description (DESC) of the array and the size (SIZE) are defined as properties of the array `interrupt_handler`. The construct `GUI_PERMIT` is a directive to the tool that you cannot change the size of the array. The array defines `int_handler` and `int_port` as parameters of an element of the array.

```

    END BLOCK
BEGIN BLOCK, dep = (level = 1)
    OPTION depends = (common_v1_00_a uartlite_vxworks5_4_v1_00_a);
    OPTION copyfiles = all;
BEGIN ARRAY interrupt_handler
    PROPERTY desc = "Interrupt Handler Information";
    PROPERTY size = 1, permit = none;
    PARAM name = int_handler, default = XUartLite_InterruptHandler, desc = "Name
of Interrupt Handler", type = string;
    PARAM name = int_port, default = Interrupt, desc = "Interrupt pin associated
with the interrupt handler", permit = none;
END ARRAY
PARAM name = connect_to, desc = "Connect to operating system", type = enum, values =
{"VxWorks5_4" = VxWorks5_4, "None" = none}, default = none;
    END BLOCK
END DRIVER

```

END is used with the construct name that was used in the BEGIN statement. Here END is used with BLOCK and DRIVER constructs to indicate the end of each BLOCK and DRIVER construct.

Example: Tcl File

The following is the `uartlite.tcl` file corresponding to the `uartlite.mdd` file described in the previous section. The "uartlite_drc" procedure would be invoked for the `uartlite` driver while running DRCs for drivers. The generate routine generates constants in a header file and a c file for `uartlite` driver, based on the driver definition segment in the MSS file.

```

proc uartlite_drc {drv_handle} {
    puts "UartLite DRC"
}

proc generate {drv_handle} {
    set level [xget_value $drv_handle "PARAMETER" "level"]

```

```

    if {$level == 0} {
        xdefine_include_file $drv_handle "xparameters.h" "XUartLite" "NUM_INSTANCES"
        "C_BASEADDR" "C_HIGHADDR"
    }
    if {$level == 1} {
        xdefine_include_file $drv_handle "xparameters.h" "XUartLite" "NUM_INSTANCES"
        "C_BASEADDR" "C_HIGHADDR" "DEVICE_ID" "C_BAUDRATE" "C_USE_PARITY" "C_ODD_PARITY"
        xdefine_config_file $drv_handle "xuartlite_g.c" "XUartLite" "DEVICE_ID"
        "C_BASEADDR" "C_BAUDRATE" "C_USE_PARITY" "C_ODD_PARITY"
    }
}

```

MDD Parameter Description

This section gives a detailed description of the constructs used in the MDD file.

Conventions

[] - Denotes optional values.

<> - Value substituted by the MDD writer.

Comments

Comments can be specified anywhere in the file. A pound (#) character denotes the beginning of a comment, and all characters after it, right up to the end of the line, are ignored. All white spaces are also ignored and semicolons with carriage returns act as sentence delimiters.

Driver Definition

The driver section includes the driver name, options, dependencies, and other global parameters, using the following syntax:

```

OPTION psf_version = <psf version number>
BEGIN DRIVER <driver name>
    [OPTION drc = <global drc name>]
    [OPTION depends = <list of directories>]
    [OPTION help = <help file>]
    [OPTION requires_interface = <list of interface names>]
    PARAM <parameter description>
    [BEGIN BLOCK,dep = <condition>
        .....
    END BLOCK]
    [BEGIN INTERFACE <interface name>
        .....
    END INTERFACE]
END DRIVER

```

MDD Keyword Summary

BEGIN	REQUIRES_INTERFACE	NAME
END	HELP	VERSION
PSF_VERSION	DEP	DESC
DRC	BLOCK	TYPE
OPTION	INTERFACE	DEFAULT
SUPPORTED_OS_TYPES	HEADER	GUI_PERMIT
COPYFILES	FUNCTION	
DEPENDS	PARAM	
SUPPORTED_PERIPHERALS	DTGPARAM	
DRIVER_STATE	PROPERTY	

MDD Keyword Definitions

BEGIN

The `BEGIN` keyword begins with one of the following: `library`, `drive`, `block`, `category`, or `interface`.

END

The `END` keyword signifies the end of a definition block.

PSF_VERSION

Specifies the PSF version of the library.

DRC

Specifies the DRC function name. This is the global DRC function, which is called by the GUI configuration tool or the command line tool. This DRC function is called when you enter all the parameters and the MLD or MDD writers can verify that a valid library or driver can be generated with the given parameters.

OPTION

Specifies the name following the keyword `OPTION` is an option to the tool. The following five options are supported: `COPYFILES`, `DEPENDS`, `SUPPORTED_PERIPHERALS`, and `DRIVER_STATE`. The following sections describe these options.

SUPPORTED_OS_TYPES

Specifies the list of supported OS types. If it is not specified, then driver is assumed as standalone driver.

COPYFILES

Specifies the list of files to be copied for the driver. If `ALL` is specified as the value, the tool copies all the driver files.

DEPENDS

Specifies the list of directories on which a driver depends for compilation.

SUPPORTED_PERIPHERALS

Specifies the list of peripherals supported by the driver. The values of this option can be specified as a list or as a regular expression. The following example indicates that the driver supports all versions of `opb_jtag_uart` and the `opb_uartlite_v1_00_b` version:

```
option supported_peripherals = (xps_uartlite_v1_0, xps_uart16550)
```

Regular expressions can be used in specifying the peripherals and versions. The regular expression (RE) is constructed as follows:

Single-Character REs

Any character that is not a special character (to be defined) matches itself.

A backslash (followed by any special character) matches the literal character itself. That is, it escapes the special character.

The special characters are: `+ * ? . [] ^ $`

The period matches any character except the newline. For example, `.umpty` matches both `Humpty` and `Dumpty`.

A set of characters enclosed in brackets (`[]`) is a one-character RE that matches any of the characters in that set. For example, `[akm]` matches an `a`, `k`, or `m`. A range of characters can be indicated with a dash. For example, `[a-z]` matches any lower-case letter. However, if the first character of the set is the caret (`^`), then the RE matches any character except those in the set. It does not match the empty string.

Example: `[^akm]` matches any character except `a`, `k`, or `m`. The caret loses its special meaning if it is not the first character of the set.

Multi-Character REs

- A single-character RE followed by an asterisk (`*`) matches zero or more occurrences of the RE. Therefore, `[a-z]*` matches zero or more lower-case characters.
- A single-character RE followed by a plus (`+`) matches one or more occurrences of the RE. Therefore, `[a-z]+` matches one or more lower-case characters.

- A question mark (?) is an optional element. The preceding RE can occur no times or one time in the string. For example, `xy?z` matches either `xyz` or `xz`.
- The concatenation of REs is an RE that matches the corresponding concatenation of strings. For example, `[A-Z][a-z]*` matches any capitalized word.

The following example matches any version of `xps_uartlite`, `xps_uart16550` and `mdm`.

```
OPTION supported_peripherals = (xps_uartlite_v[0-9]+_[1-9][0-9]_[a-z]
xps_uart16550 mdm);
```

DRIVER_STATE

Specifies the state of the driver. The following are the list of values that can be assigned to `DRIVER_STATE`:

ACTIVE

This is an active driver. By default the value of `DRIVER_STATE` is `ACTIVE`.

DEPRECATED

This driver is deprecated and is scheduled to be removed.

OBSOLETE

This driver is obsolete and is not recognized by any tools. Tools error out on an obsolete driver, and a new driver should be used instead.

REQUIRES_INTERFACE

Specifies the interfaces that must be provided by other libraries or drivers in the system.

HELP

Specifies the help file that describes the library or driver.

DEP

Specifies the condition that needs to be satisfied before processing an entity. For example to enter into a `BLOCK`, the `DEP` condition should be satisfied. Conditions of the form `(operand1 OP operand2)` are supported.

BLOCK

Specifies the block is to be entered into when the `DEP` condition is satisfied. Nested blocks are not supported.

INTERFACE

Specifies the interfaces implemented by this library or driver and describes the interface functions and header files used by the library or driver.

```
BEGIN INTERFACE <interface name>
  OPTION DEP=<list of dependencies>;
  PROPERTY HEADER=<name of header file where the function is declared>;
  FUNCTION NAME=<name of interface function>, VALUE=<function name of
library/driver implementation> ;
END INTERFACE
```

HEADER

Specifies the header file in which the interface functions would be defined.

FUNCTION

Specifies the function implemented by the interface. This is a name-value pair where *name* is the interface function name and *value* is the name of the function implemented by the library or driver.

PARAM

Generally, the MLD/MDD file has a *name = value* format for statements. The `PARAM` keyword is required before every such `NAME, VALUE` pair. The format for assigning a value to a parameter is `param name = <name>, default= value`. The `PARAM` keyword specifies that the parameter can be overwritten in the MSS file.

DTGPARAM

The `DTGPARAM` keyword is specially used for the device-tree specific parameters that can be configured. Driver defines these `DTGPARAMs` if it needs to dump any parameters in the Tool DTG generated DTS file.

PROPERTY

Specifies the various properties of the entity defined with a `BEGIN` statement

NAME

Specifies the name of the entity in which it was defined (example: `PARAM, PROPERTY`). It also specifies the name of the driver if it is specified with option.

VERSION

Specifies the version of the driver.

DESC

Describes the entity in which it was defined (example: `PARAM, PROPERTY`).

TYPE

Specifies the type for the entity in which it was defined (example: `PARAM`). The following are the supported types:

bool

Boolean (true or false)

int

Integer

string

String value within " " (quotes).

enum

List of possible values, that this parameter can take.

library

Specify other library that is needed for building the library or driver.

peripheral_instance

Specify other hardware drivers needed for building the library or driver. Regular expressions can be used to specify the peripheral instance. Refer to [SUPPORTED_PERIPHERALS, page 39](#) for more details on regular expressions.

DEFAULT

Specifies the default value for the entity in which it was defined.

GUI_PERMIT

Specifies the permissions for modification of values. The following permissions exist:

NONE

The value cannot be modified at all.

ADVANCED_USER

The value can be modified by all. The SDK GUI does not display this value by default. It is displayed only as an advanced option in the GUI.

ALL_USERS

The value can be modified by all. The SDK GUI displays this value by default. This is the default value for all the values.

If `GUI_PERMIT = NONE`, the category is always active.

Design Rule Check (DRC) Section

```
proc mydrc { handle }
```

The DRC function can be any Tcl code that checks your parameters for correctness. The DRC procedures can access (read-only) the Platform Specification Format database (built by the tool using the MSS files) to read the parameter values you set. The "handle" is a handle to the current driver in the database. The DRC procedure can get the driver parameters from this handle. It can also get any other parameter from the database, by first requesting a handle and using the handle to get the parameters.

- For errors, DRC procedures would call the Tcl error command `error "error msg"` that displays in an error dialog box.
- For warnings, DRC procedures return a string value that can be printed on the console.
- On success, DRC procedures just return without any value.

Driver Generation (Generate) Section

```
proc mygenerate { handle }
```

`Generate` could be any Tcl code that reads your parameters and generates configuration files for the driver. The configuration files can be C files, Header files, or Makefiles.

The `generate` procedures can access (read-only) the Platform Specification Format database (built by the tool using the MSS files) to read the parameter values of the driver that you set.

The `handle` is a handle to the current driver in the database.

The `generate` procedure can get the driver parameters from this handle. It can also get any other parameters from the database by requesting a handle and then using the handle to get the parameter.

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see [Xilinx Support](#).

Solution Centers

See the [Xilinx Solution Centers](#) for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

References

- *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing* ([UG973](#))
- *Zynq-7000 All Programmable SoC Software Developers Guide* ([UG821](#))

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at <http://www.xilinx.com/legal.htm#tos>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at <http://www.xilinx.com/legal.htm#tos>.

© Copyright 2014 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.

Hardware Software Interface Reference

Overview of Tcl Capabilities in Hardware Software Interface

The Tool Command Language (Tcl) is the scripting language integrated in the Hardware Software Interface (HSI) environment.

Tcl lets you perform interactive queries to design tools in addition to executing automated scripts. Tcl offers the ability to “ask” questions interactively of design databases, particularly around tool and design settings and state. Examples are: querying IP, Driver, BSP and Driver configuration. Querying interrupt and other connectivity information.

The following sections describe some of the basic capabilities of Tcl with HSI.

Note: This manual is not a comprehensive reference for the Tcl language. It is a reference to the specific capabilities of the HSI Tcl shell, and provides reference to additional Tcl programming resources.

First Class Tcl Objects and Relationships

Object Types and Definitions

Hardware

- **HardwareDb**

HardwareDb represents hardware design loaded in memory. SDK, PetaLinux and third party tools can have multiple HardwareDb objects.
- **Port**

Port is a special type of pin on the top-level module or entity. Ports are normally attached to I/O pads and connect externally to the FPGA device.
- **InterfacePort**

Interface port is a special type of bus-interface on the top level module or entity. Interface Ports are normally attached to I/O pads and connect externally to the FPGA device.
- **Net**

Net is a wire or list of wires that eventually be physically connected directly together. Nets are always sorts a list of pins together.
- **InterfaceNet**

Interface net is a list of wires that eventually be physically connected directly together. Interface nets are always sorts a list of interface pins together.
- **Pin**

A pin is a point of logical connectivity on a cell. A pin allows the internals of a cell to be abstracted away and simplified for easier use on cell. Examples of pins include clock, data, reset, and output pins of a flop.
- **InterfacePin**

Interface pin is a point of logical connectivity on a cell. It allows the internals of a cell to be abstracted away and simplified for easier use on cell. Examples of interface pins include M_AXI_DP interface of microblaze, and S_AXI interface of gpio.
- **Cell**

Cell is instantiation of IP in the hardware design. Examples of cells include instantiation of microblaze, gpio, axi_dma .
- **MemoryRange**

It represents memory range associated with the peripherals in the memory map of the processor.

- SupportingDesignFile

It represents files associated with the hardware design

Software

- SoftwareDb

It represents one software design or microprocessor software specification(MSS). SDK, Petalinux and other third party tools can have multiple SoftwareDb objects.

- SwCore

It represents driver/library/OS present in the software repositories. Examples are cpu, gpio, standalone, xilffs, etc.

- SwProcessor

SwProcessor is the driver mapped to processor instance in the hardware design. Examples are cpu driver mapped for microblaze cell.

- SwDriver

SwDriver is the driver mapped to peripheral instance in the hardware design. Examples are gpio driver mapped for axi_gpio cell.

- SwLibrary

SwLibrary is the library added in the software design. Examples are xilflash, xilffs, etc.

- SwOS

SwOS is OS in the software design. Examples are standalone, xilkernel, etc.

- SwInterface

SwInterface is interface of library/driver and describes the interface functions and header files used by the library/drive. Examples are stdin, stdout of uart driver.

- SwArray

SwArray is array defined in driver/library/os. It contains any number of PARAMs and PROPERTYs which describes size, description of array and default values of elements in array. It represents software array of the driver/library/os. Examples are mem_table, shm_table of xilkernel bsp.

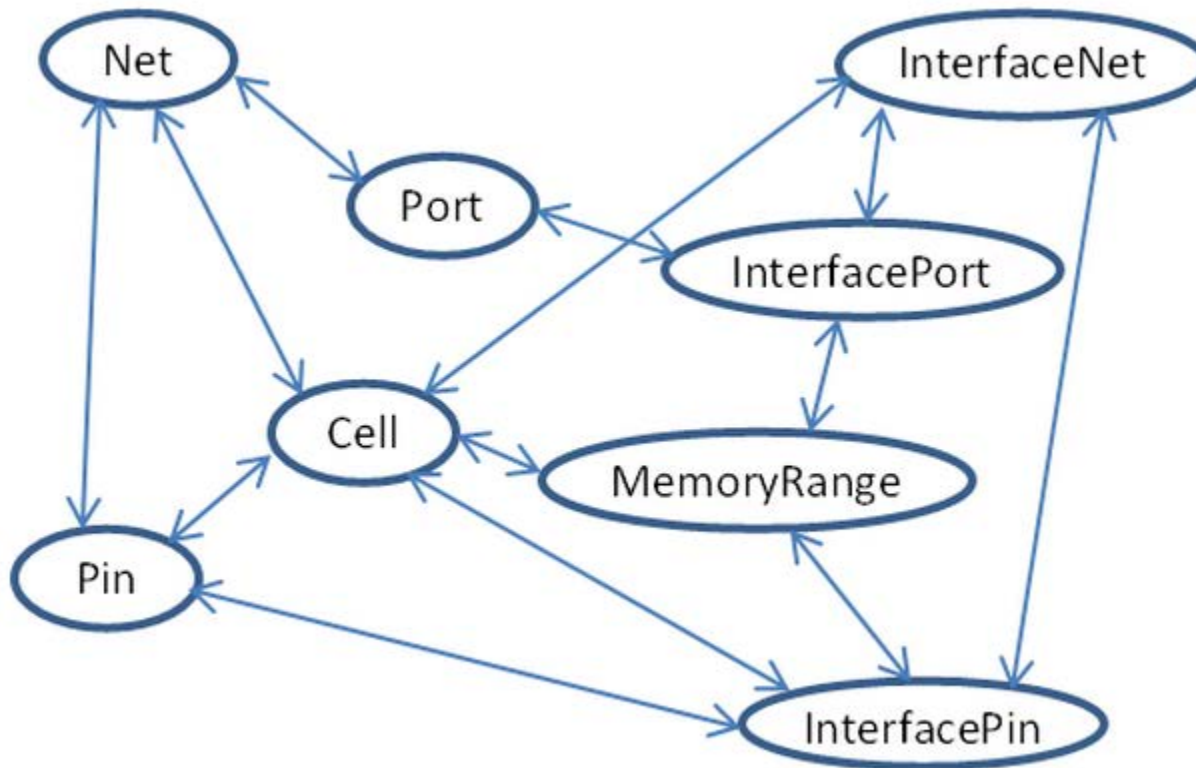
- SwDTNode

SwDTNode represents node in Device Tree (DTS) file.

- SwParam

SwParam represents parameters of node.

Object Relationships



Tcl Commands Listed by Category

Categories

- [Hardware](#)
- [Software](#)
- [PropertyAndParameter](#)

Hardware

- [hsm::open hw design](#)
- [hsm::close hw design](#)
- [hsm::current hw design](#)
- [hsm::get hw designs](#)
- [hsm::get hw files](#)
- [hsm::get mem ranges](#)
- [hsm::get cells](#)
- [hsm::get ports](#)
- [hsm::get nets](#)
- [hsm::get pins](#)
- [hsm::get intf ports](#)
- [hsm::get intf nets](#)
- [hsm::get intf pins](#)

Software

- [hsm::create sw design](#)
- [hsm::open sw design](#)
- [hsm::close sw design](#)
- [hsm::current sw design](#)
- [hsm::get sw designs](#)
- [hsm::set repo path](#)
- [hsm::get sw cores](#)
- [hsm::delete objs](#)

- [hsm::get_drivers](#)
- [hsm::get_libs](#)
- [hsm::get_os](#)
- [hsm::get_sw_processor](#)
- [hsm::get_arrays](#)
- [hsm::get_sw_interfaces](#)
- [hsm::get_nodes](#)
- [hsm::get_comp_params](#)
- [hsm::create_node](#)
- [hsm::create_comp_param](#)

PropertyAndParameter

- [create_property](#)
- [get_param](#)
- [get_property](#)
- [list_param](#)
- [list_property](#)
- [list_property_value](#)
- [report_param](#)
- [report_property](#)
- [reset_param](#)
- [reset_property](#)
- [set_param](#)
- [set_property](#)

Hardware TCL Commands

hsm::open_hw_design

Open a hardware design from disk file.

Syntax:

```
hsm::open_hw_design [-quiet] [-verbose] [<file>]
```

Returns:

Hardware design object. Returns nothing if the command fails.

Usage:

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<file>]	Hardware design file to open

Categories:

Hardware

Description:

Open a Hardware design in the tool. The Hardware design must be exported previously using the Vivado product. Users can open multiple hardware designs at same time.

This command returns a Hardware design object of the opened Hardware design, or returns an error if the command fails.

Arguments:

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the set_msg_config command.

<file>

The path and file name of the Hardware design to open in the tool. The name must include the file extension.

Examples:

The following opens the specified IP subsystem design in the current project:

```
open_hw_design C:/Data/project1/project1.sdk/SDK/SDK_Export/hw/  
design_1.xml
```

OR

```
open_hw_design C:/Data/project1/project1.sdk/design_1_wrapper.hdf
```

See Also:

- [hsm::close hw design](#)
- [hsm::current hw design](#)

hsm::close_hw_design

Close a hardware design.

Syntax:

```
hsm::close_hw_design [-quiet] [-verbose] <name>
```

Returns:

Returns nothing, error message if failed.

Usage:

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<name>	Name of design to close

Categories:

Hardware

Description:

Closes the hardware design in the tool active session. Design modification is not allowed in the current release, otherwise it will prompt to save the design prior to closing.

Arguments:

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the set_msg_config command.

<name>

The name of the hardware design object to close.

Example:

The following example closes the current Hardware design object in the current session:

```
close_hw_design [current_hw_design]  
close_hw_design design_1_imp
```

See Also:

- [hsm::current_hw_design](#)
- [hsm::open_hw_design](#)

hsm::current_hw_design

Set or get current hardware design.

Syntax:

```
hsm::current_hw_design [-quiet] [-verbose] [<design>]
```

Returns:

Current hardware design object, "" if failed.

Usage:

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<design>]	design to be set

Categories:

Hardware

Description:

Defines the current Hardware design for use with the tool, or returns the name of the current design in the active project.

The current hardware design is the target of the tool hardware Tcl commands.

You can use the `get_hw_designs` command to get a list of open hardware designs in the active project.

Arguments:

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns `TCL_OK` regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

<design>

(Optional) The name of an hardware design to set as the current design in the tool. If a <design> is not specified, the command returns the current hardware design of the active project.

Examples:

The following example sets the hardware design as the current design:

```
current_hw_design design_1
current_hw_design
```

See Also:

- [hsm::open hw design](#)
- [hsm::close hw design](#)

hsm::get_hw_designs

Get a list of hardware designs opened

Syntax:

```
hsm::get_hw_designs [-regexp] [-filter <arg>] [-quiet] [-verbose]
                    [<patterns>...]
```

Returns:

Hardware design objects. Returns nothing if the command fails.

Usage:

Name	Description
[-regexp]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<patterns>]	Match design names against patterns Default: *

Categories:

Hardware

Description:

Gets a list of hardware designs open in the current tool session that match a specified search pattern. The default command gets a list of all open hardware designs in the session.

Arguments:

-regexp

(Optional) Specifies that the search <patterns> are written as regular expressions. Both search <patterns> and -filter expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of or to the end of the search string. You can add "." to the beginning or end of a search string to widen the search to include a substring. See http://www.tcl.tk/man/tcl8.5/TclCmd/re_syntax.htm for help with regular expression syntax.

Note: The Tcl built-in command regexp is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.5/TclCmd/regexp.htm>.

-filter <args>

(Optional) Filter the results list with the specified expression. The `-filter` argument filters the list of objects returned by `get_hw_designs` based on conditional expression of property values on the designs. You can find the properties on an object with the `report_property` or `list_property` commands. In the case of the "Hardware design" object, "NAME", is the property that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, port, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

Note: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of ""

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_hw_designs * -filter {NAME !~ "*design*"}
```

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns `TCL_OK` regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

<patterns>

(Optional) Match designs against the specified patterns. The default pattern is the wildcard "*" which gets all hardware designs. More than one pattern can be specified to find multiple designs based on different search criteria.

Examples:

The following example gets all open hardware designs in the current session:

```
get_hw_designs
```

See Also:

- [hsm::current_hw_design](#)
- [hsm::open_hw_design](#)
- [report_property](#)

hsm::get_hw_files

Get a list of hardware design supporting files

Syntax:

```
hsm::get_hw_files [-regexp] [-filter <arg>] [-of_objects <args>] [-quiet]
                  [-verbose] [<patterns>...]
```

Returns:

Hardware design supporting file objects. Returns nothing if the command fails.

Usage:

Name	Description
[-regexp]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-of_objects]	Get 'hw_file' objects of these types: 'hw_design'.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<patterns>]	Match cell names against patterns Default: *

Categories:

Hardware

Description:

Gets a list of hardware handoff files in the current hardware session.

Arguments:

-regexp

(Optional) Specifies that the search <patterns> are written as regular expressions. Both search <patterns> and -filter expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search to include a substring. See http://www.tcl.tk/man/tcl8.5/TclCmd/re_syntax.htm for help with regular expression syntax.

Note: The Tcl built-in command regexp is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.5/TclCmd/regexp.htm>.

-filter <args>

(Optional) Filter the results list with the specified expression. The `-filter` argument filters the list of objects returned by `get_files` based on property values on the files. You can find the properties on an object with the `report_property` or `list_property` commands.

Any property/value pair can be used as a filter. In the case of the "file" object, "TYPE" is the property that can be used to filter results.

-of_objects <args>

(Optional) Get the files that are associated with the specified hardware design objects. The default is to search all opened hardware.

Note: The `-of_objects` option requires objects to be specified using the `get_*` commands, such as `get_cells` or `get_pins`, rather than specifying objects by name. In addition, `-of_objects` cannot be used with a `search <pattern>`

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns `TCL_OK` regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

<patterns>

(Optional) Match files against the specified patterns. The default pattern is the wildcard ``*`` which gets all files in the project or `of_objects`. More than one pattern can be specified to find multiple files based on different search criteria.

Examples:

The following example returns the bit files in the design that are used for programming FPGA :

```
get_files -filter {TYPE == bit}
```

hsm::get_mem_ranges

Get a list of memory ranges

Syntax:

```
hsm::get_mem_ranges [-regexp] [-filter <arg>] [-of_objects <args>]
[-quiet] [-verbose] [<patterns>...]
```

Returns:

Memory range objects. Returns nothing if the command fails.

Usage:

Name	Description
[-regexp]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-of_objects]	Get 'mem_range' objects of these types: 'cell'.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<patterns>]	Match cell names against patterns Default: *

Categories:

Hardware

Description:

Get a list of slaves of the processor/master in the current hardware design.

Arguments:

-regexp

(Optional) Specifies that the search <patterns> are written as regular expressions. Both search <patterns> and -filter expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add ".*" to the beginning or end of a search string to widen the search to include a substring. See http://www.tcl.tk/man/tcl8.5/TclCmd/re_syntax.htm for help with regular expression syntax.

Note: The Tcl built-in command regexp is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.5/TclCmd/regexp.htm>.

-filter <args>

(Optional) Filter the results list with the specified expression. The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

Note: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of ""

-of_objects <arg>

(Optional) Get the slaves of the specified processor/master cell object.

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the set_msg_config command.

<patterns>

(Optional) Match address segments against the specified patterns. The default pattern is the wildcard "*" which gets a list of all address segments in the current IP subsystem design. More than one pattern can be specified to find multiple address segments based on different search criteria.

Note: You must enclose multiple search patterns in braces {} to present the list as a single element.

Examples:

The following example gets the slaves of the processor:

```
get_mem_ranges
get_mem_ranges -of_objects [lindex [get_cells -filter
{IP_TYPE==PROCESSOR} ] 0]
```

Note: If there are no objects matching the pattern you will get a warning.

hsm::get_cells

Get a list of cells

Syntax:

```
hsm::get_cells [-regex] [-filter <arg>] [-of_objects <args>] [-quiet]
               [-verbose] [<patterns>...]
```

Returns:

Cell objects. Returns nothing if the command fails.

Usage:

Name	Description
[-regex]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-of_objects]	Get 'cell' objects of these types: 'hw_design port bus_intf net intf_net'.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<patterns>]	Match cell names against patterns Default: *

Categories:

Hardware

Description:

Gets a list of IP instance objects in the current hardware design that match a specified search pattern. The default command returns a list of all IP instances in the current hardware design.

Note: To improve memory and performance, the get_* commands return a container list of a single type of objects (e.g. cells, nets, or ports). You can add new objects to the list (using lappend for instance), but you can only add the same type of object that is currently in the list.

Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments:

-regexp

(Optional) Specifies that the search <patterns> are written as regular expressions. Both search <patterns> and -filter expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add ".*" to the beginning or end of a search string to widen the search to include a substring. See http://www.tcl.tk/man/tcl8.5/TclCmd/re_syntax.htm for help with regular expression syntax.

Note: The Tcl built-in command regexp is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.5/TclCmd/regexp.htm>.

-filter <args>

(Optional) Filter the results list with the specified expression. The -filter argument filters the list of objects returned by get_cells based on property values on the cells. You can find the properties on an object with the report_property or list_property commands.

In the case of the "cell" object, "IP_TYPE", and "IP_NAME" are some of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

Note: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of ""

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_cells * -filter {IP_TYPE == PROCESSOR && NAME !~ "*ps7*"}

```

-of_objects <arg>

(Optional) Get the cells connected to the specified pins, nets, intf_pins, intf_nets and mem_range objects.

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

<patterns>

(Optional) Match cells against the specified patterns. The default pattern is the wildcard ``*`` which gets a list of all cells in the project. More than one pattern can be specified to find multiple cells based on different search criteria.

Note: You must enclose multiple search patterns in braces, `{}`, or quotes, `""`, to present the list as a single element.

Examples:

The following example returns list of processor instances :

```
get_cells -filter { IP_TYPE == "PROCESSOR" }
```

This example gets a list of properties and property values attached to the second object of the list returned by `get_cells`:

```
report_property [lindex [get_cells] 1]
```

Note: If there are no cells matching the pattern you will get a warning.

See Also:

- [hsm::get_nets](#)
- [hsm::get_pins](#)
- [list_property](#)
- [report_property](#)



hsm::get_ports

Get a list of external ports

Syntax:

```
hsm::get_ports [-regex] [-filter <arg>] [-of_objects <args>] [-quiet]
               [-verbose] [<patterns>...]
```

Returns:

Port objects. Returns nothing if the command fails.

Usage:

Name	Description
[-regex]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-of_objects]	Get 'port' objects of these types: 'hw_design bus_intf net'.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<patterns>]	Match cell names against patterns Default: *

Categories:

Hardware

Description:

Gets a list of port objects in the current hardware design that match a specified search pattern. The default command gets a list of all ports in the hardware design.

The external connections in an hardware design are ports, or interface ports. The internal connections in an IP instance cell are pins and interface pins. Use the `get_pins` and `get_intf_pins` commands to select the pin objects.

Arguments:

-regexp

(Optional) Specifies that the search <patterns> are written as regular expressions. Both search <patterns> and -filter expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search to include a substring. See http://www.tcl.tk/man/tcl8.5/TclCmd/re_syntax.htm for help with regular expression syntax.

Note: The Tcl built-in command regexp is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.5/TclCmd/regexp.htm>.

-filter <args>

(Optional) Filter the results list with the specified expression. The -filter argument filters the list of objects returned by get_ports based on property values on the ports. You can find the properties on an object with the report_property or list_property commands. In the case of the IP subsystem port object, "DIRECTION", "TYPE", and "SENSITIVITY" are some of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

Note: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object.

In the case of the "*" wildcard character, this will match a property with a defined value of ""

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_ports * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
```

-of_objects <arg>

(Optional) Get the ports connected to the specified nets returned by get_nets or get_intf_nets.

Note: The -of_objects option requires objects to be specified using the get_* commands, such as get_cells or get_pins, rather than specifying objects by name. In addition, -of_objects cannot be used with a search <pattern>

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

<patterns>

(Optional) Match ports against the specified patterns. The default pattern is the wildcard ``*`` which gets a list of all ports in the subsystem design. More than one pattern can be specified to find multiple ports based on different search criteria.

Note: You must enclose multiple search patterns in braces `{}` to present the list as a single element.

Examples:

The following example gets the ports connected to the specified hardware subsystem net:

```
get_ports -of_objects [get_nets bridge_1_apb_m] -filter {DIRECTION==I}
```

Note: If there are no ports matching the pattern, the tool will return a warning.

See Also:

- [hsm::get_nets](#)
- [hsm::get_pins](#)

hsm::get_nets

Get a list of nets

Syntax:

```
hsm::get_nets [-regex] [-filter <arg>] [-of_objects <args>] [-quiet]
              [-verbose] [<patterns>...]
```

Returns:

Net objects. Returns nothing if the command fails.

Usage:

Name	Description
[-regex]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-of_objects]	Get 'net' objects of these types: 'hw_design cell port'.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<patterns>]	Match cell names against patterns Default: *

Categories:

Hardware

Description:

Gets a list of nets in the current hardware design that match a specified search pattern. The default command gets a list of all nets in the subsystem design.

Arguments:

-regex

(Optional) Specifies that the search <patterns> are written as regular expressions. Both search <patterns> and -filter expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search to include a substring. See http://www.tcl.tk/man/tcl8.5/TclCmd/re_syntax.htm for help with regular expression syntax.

Note: The Tcl built-in command regex is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.5/TclCmd/regex.htm>.

-filter <args>

(Optional) Filter the results list with the specified expression. The `-filter` argument filters the list of objects returned by `get_nets` based on property values on the nets. You can find the properties on an object with the `report_property` or `list_property` commands. In the case of the hardware design `nets` object, "NAME" property that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

Note: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object.

In the case of the "*" wildcard character, this will match a property with a defined value of ""

-of_objects <args>

(Optional) Get a list of the nets connected to the specified cell, pin, or port objects.

Note: The `-of_objects` option requires objects to be specified using the `get_*` commands, such as `get_cells` or `get_pins`, rather than specifying objects by name. In addition, `-of_objects` cannot be used with a `search <pattern>`

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns `TCL_OK` regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

<patterns>

(Optional) Match hardware design nets against the specified patterns. The default pattern is the wildcard `*`` which returns a list of all nets in the current IP Integrator subsystem design. More than one pattern can be specified to find multiple nets based on different search criteria.

Note: You must enclose multiple search patterns in braces `{}` to present the list as a single element.

Examples:

The following example gets the net attached to the specified pin of an hardware design module, and returns both the net:

```
get_nets -of_objects [get_pins aclk]
```

Note: If there are no nets matching the pattern you will get a warning.

See Also:

- [hsm::get_cells](#)
- [hsm::get_pins](#)
- [hsm::get_ports](#)
- [list_property](#)
- [report_property](#)

hsm::get_pins

Get a list of pins.

Syntax:

```
hsm::get_pins [-regexp] [-filter <arg>] [-of_objects <args>] [-quiet]
              [-verbose] [<patterns>...]
```

Returns:

Pin objects. Returns nothing if the command fails.

Usage:

Name	Description
[-regexp]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-of_objects]	Get 'port' objects of these types: 'hw_design cell bus_intf net'.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<patterns>]	Match cell names against patterns Default: *

Categories:

Hardware

Description:

Gets a list of pin objects on the current hardware design that match a specified search pattern. The default command gets a list of all pins in the subsystem design.

Arguments:

-regexp

(Optional) Specifies that the search <patterns> are written as regular expressions. Both search <patterns> and -filter expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search to include a substring. See http://www.tcl.tk/man/tcl8.5/TclCmd/re_syntax.htm for help with regular expression syntax.

Note: The Tcl built-in command regexp is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.5/TclCmd/regexp.htm>.

-filter <args>

(Optional) Filter the results list with the specified expression. The `-filter` argument filters the list of objects returned by `get_pins` based on property values on the pins. You can find the properties on an object with the `report_property` or `list_property` commands. In the case of the pins object, "DIR" and "TYPE" are some of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

Note: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of ""

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||).

The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == I && NAME !~ "*RESET*"}
```

-of_objects <arg>

(Optional) Get the pins connected to the specified cell, net or `intf_pin`.

Note: The `-of_objects` option requires objects to be specified using the `get_*` commands, such as `get_cells` or `get_pins`, rather than specifying objects by name. In addition, `-of_objects` cannot be used with a search `<pattern>`

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns `TCL_OK` regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

<patterns>

(Optional) Match hardware design pins against the specified patterns.

Note: More than one pattern can be specified to find multiple pins based on different search criteria. You must enclose multiple search patterns in braces {} to present the list as a single element

Examples:

The following example gets a list of pins attached to the specified cell:

```
get_pins -of_objects [get_cells axi_gpio_0]
```

Note: If there are no pins matching the pattern, the tool will return a warning.

The following example gets a list of pins attached to the specified net:

```
get_pins -of_objects [get_nets ps7_axi_interconnect_0_M_AXI_BRESP]
```

See Also:

- [hsm::get_intf_pins](#)
- [hsm::get_intf_nets](#)
- [hsm::get_nets](#)
- [hsm::get_pins](#)
- [list_property](#)
- [report_property](#)

hsm::get_intf_ports

Get a list of interface Ports

Syntax:

```
hsm::get_intf_ports [-regex] [-filter <arg>] [-of_objects <args>]
[-quiet] [-verbose] [<patterns>...]
```

Returns:

Interface Port objects. Returns nothing if the command fails.

Usage:

Name	Description
[-regex]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-of_objects]	Get 'bus_intf' objects of these types: 'hw_design port net'.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<patterns>]	Match cell names against patterns Default: *

Categories:

Hardware

Description:

Gets a list of interface port objects in the current hardware subsystem design that match a specified search pattern. The default command gets a list of all interface ports in the subsystem design.

The external connections in a hardware design are ports, or interface ports. The external connections in a hardware design are pins and interface pins. Use the `get_pins` and `get_intf_pins` commands to select the pin objects.

Note: To improve memory and performance, the `get_*` commands return a container list of a single type of objects (e.g. cells, nets, pins, or ports). You can add new objects to the list (using `lappend` for instance), but you can only add the same type of object that is currently in the list. Adding a different type of object, or string, to the list is not permitted and will result in a Tcl error.

Arguments:

-regexp

(Optional) Specifies that the search <patterns> are written as regular expressions. Both search <patterns> and -filter expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search to include a substring. See http://www.tcl.tk/man/tcl8.5/TclCmd/re_syntax.htm for help with regular expression syntax.

Note: The Tcl built-in command regexp is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.5/TclCmd/regexp.htm>.

-filter <args>

(Optional) Filter the results list with the specified expression. The -filter argument filters the list of objects returned by get_intf_ports based on property values on the interface ports. You can find the properties on an object with the report_property or list_property commands. In the case of the IP subsystem interface port object, "DIRECTION", and "NAME" are some of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

Note: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of ""

-of_objects <arg>

(Optional) Get the interface ports connected to the specified interface nets returned by get_intf_nets.

Note: The -of_objects option requires objects to be specified using the get_* commands, such as get_cells or get_pins, rather than specifying objects by name. In addition, -of_objects cannot be used with a search <pattern>

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

<patterns>

(Optional) Match interface ports against the specified patterns. The default pattern is the wildcard `*`` which gets a list of all interface ports in the subsystem design. More than one pattern can be specified to find multiple interface ports based on different search criteria.

Note: You must enclose multiple search patterns in braces `{}` to present the list as a single element.

Examples:

The following example gets the interface ports in the subsystem design that operate in Master mode:

```
get_intf_ports -filter {MODE=="master"}
```

Note: If there are no interface ports matching the pattern, the tool will return a warning.

See Also:

- [hsm::get_intf_nets](#)
- [hsm::get_intf_pins](#)
- [hsm::get_intf_ports](#)
- [hsm::get_nets](#)
- [hsm::get_pins](#)
- [list_property](#)
- [report_property](#)

hsm::get_intf_nets

Get a list of interface Nets

Syntax:

```
hsm::get_intf_nets [-regexp] [-filter <arg>] [-of_objects <args>]
[-quiet] [-verbose] [<patterns>...]
```

Returns:

Interface Net objects. Returns nothing if the command fails.

Usage:

Name	Description
[-regexp]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-of_objects]	Get 'net' objects of these types: 'hw_design cell bus_intf'.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<patterns>]	Match cell names against patterns Default: *

Categories:

Hardware

Description:

Gets a list of interface nets in the current hardware design that match a specified search pattern. The default command gets a list of all interface nets in the subsystem design.

Arguments:

-regexp

(Optional) Specifies that the search <patterns> are written as regular expressions. Both search <patterns> and -filter expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search to include a substring. See http://www.tcl.tk/man/tcl8.5/TclCmd/re_syntax.htm for help with regular expression syntax.

Note: The Tcl built-in command regexp is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.5/TclCmd/regexp.htm>.

-filter <args>

(Optional) Filter the results list with the specified expression. The `-filter` argument filters the list of objects returned by `get_intf_nets` based on property values on the nets. You can find the properties on an object with the `report_property` or `list_property` commands. In the case of the hardware design nets object, "NAME" property that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

Note: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of ""

-of_objects <args>

(Optional) Get a list of the nets connected to the specified IP Integrator subsystem cell, pin, or port objects.

Note: The `-of_objects` option requires objects to be specified using the `get_*` commands, such as `get_cells` or `get_pins`, rather than specifying objects by name. In addition, `-of_objects` cannot be used with a search `<pattern>`

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns `TCL_OK` regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

<patterns>

(Optional) Match hardware design interface nets against the specified patterns. The default pattern is the wildcard ``*`` which returns a list of all interface nets in the current IP Integrator subsystem design. More than one pattern can be specified to find multiple nets based on different search criteria.

Note: You must enclose multiple search patterns in braces `{}` to present the list as a single element.

Examples:

The following example gets the interface net attached to the specified pin of an hardware design, and returns the net:

```
get_intf_nets -of_objects [get_pins aclk]
```

Note: If there are no interface nets matching the pattern you will get a warning.

See Also:

- [hsm::get_cells](#)
- [hsm::get_pins](#)
- [hsm::get_ports](#)
- [list_property](#)
- [report_property](#)

hsm::get_intf_pins

Get a list of interface Pins

Syntax:

```
hsm::get_intf_pins [-regexp] [-filter <arg>] [-of_objects <args>]
[-quiet] [-verbose] [<patterns>...]
```

Returns:

Interface pin objects. Returns nothing if the command fails.

Usage:

Name	Description
[-regexp]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-of_objects]	Get 'bus_intf' objects of these types: 'hw_design cell port net'.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<patterns>]	Match cell names against patterns Default: *

Categories:

Hardware

Description:

Gets a list of pin objects in the current design that match a specified search pattern. The default command gets a list of all pins in the design.

Arguments:

-regexp

(Optional) Specifies that the search <patterns> are written as regular expressions. Both search <patterns> and -filter expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search to include a substring. See http://www.tcl.tk/man/tcl8.5/TclCmd/re_syntax.htm for help with regular expression syntax.

Note: The Tcl built-in command regexp is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.5/TclCmd/regexp.htm>.

-filter <args>

(Optional) Filter the results list with the specified expression. The `-filter` argument filters the list of objects returned by `get_pins` based on property values on the pins. You can find the properties on an object with the `report_property` or `list_property` commands. In the case of the pins object, "NAME" and "TYPE" are some of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

Note: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of ""

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||).

The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_intf_pins * -filter {TYPE == SLAVE && NAME !~ "*S_AXI*"}
```

-of_objects <arg>

(Optional) Get the pins connected to the specified cell, clock, timing path, or net; or pins associated with specified DRC violation objects.

Note: The `-of_objects` option requires objects to be specified using the `get_*` commands, such as `get_cells` or `get_pins`, rather than specifying objects by name. In addition, `-of_objects` cannot be used with a search <pattern>

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

<patterns>

(Optional) Match pins against the specified patterns. The default pattern is the wildcard `*` which gets a list of all pins in the project. More than one pattern can be specified to find multiple pins based on different search criteria.

Note: You must enclose multiple search patterns in braces, {}, or quotes, "", to present the list as a single element.

Examples:

The following example gets a list of pins attached to the specified cell:

```
get_intf_pins -of_objects [lindex [get_cells] 1]
```

Note: If there are no pins matching the pattern, the tool will return a warning.

See Also:

- [hsm::get_intf_nets](#)
- [hsm::get_intf_ports](#)
- [list_property](#)
- [report_property](#)

Software TCL Commands

hsm::create_sw_design

Create a software design.

Syntax:

```
hsm::create_sw_design -proc <arg> [-app <arg>] [-os <arg>]
[-os_ver <arg>] [-quiet] [-verbose] <name>...
```

Returns:

Software design object. Returns nothing if the command fails.

Usage:

Name	Description
-proc	processor name
[-app]	Application name
[-os]	os name
	Default: standalone
[-os_ver]	os version
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<name>	Software design name

Categories:

Software

Description:

Create a new software design module to add to the current session.

This command returns the name of the software design created if the command is successful. An error is returned if the command fails.

Arguments:

-proc

The processor instance name targeted for the software design.

-app

The template application name.

-os

(Optional) The OS name targeted for the software design. Default value : standalone.

-os_ver

(Optional) The OS version targeted for the software design. Default value : latest OS version.

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

<name>

The name of the software design module to create.

Example:

The following example creates a new software design module called `sw_design_1`, adds the module to the current session:

```
create_sw_design sw_design_1 -proc microblaze_0 -os xilkernel
create_sw_design sw_design_1 -proc microblaze_0 -os xilkernel -os_ver 6.0
create_sw_design sw_design_1 -proc ps7_cortexa9_0
```

See Also:

- [hsm::close sw design](#)
- [hsm::current sw design](#)

hsm::open_sw_design

Open a software design.

Syntax:

```
hsm::open_sw_design [-quiet] [-verbose] [<file>]
```

Returns:

Software design object. Returns nothing if the command fails.

Usage:

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<file>]	Software design file to open

Categories:

Software

Description:

Open an Software design in the tool.

This command returns a software design object of the opened Software design, or returns an error if the command fails.

Arguments:

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the set_msg_config command.

<name>

The name of the Software design to open in the tool.

Examples:

The following opens the specified software design in the current session:

```
open_sw_design sw_design_1
```

See Also:

- [hsm::close_sw_design](#)
- [hsm::current_sw_design](#)

hsm::close_sw_design

Close a software design.

Syntax:

```
hsm::close_sw_design [-quiet] [-verbose] <name>...
```

Returns:

Returns nothing, error message if failed.

Usage:

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<name>	Name of design to close

Categories:

Software

Description:

Closes the specified software design in the current session.

If the design has been modified, you will not be prompted to save the design prior to closing. In the current release persistence option is not available.

Arguments:

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the set_msg_config command.

<name>

The name of the software design object to close.

Example:

The following example closes the current software design in the current session:

```
close_sw_design [current_sw_design]
close_sw_design
close_sw_design sw_design_1
```

See Also:

- [hsm::create sw design](#)
- [hsm::current sw design](#)
- [hsm::get sw designs](#)
- [hsm::open sw design](#)

hsm::current_sw_design

Set or get current software design.

Syntax:

```
hsm::current_sw_design [-quiet] [-verbose] [<design>]
```

Returns:

Software design object, "" if failed.

Usage:

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<design>]	design to be set

Categories:

Software

Description:

Defines the current Software design for use with the current session, or returns the name of the current design in the active session.

You can use the `get_sw_designs` command to get a list of open software designs in the active session.

Arguments:

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns `TCL_OK` regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

<design>

(Optional) The name of an software design to set as the current design in the session. If a <design> is not specified, the command returns the current software design of the active session.

Examples:

The following example sets the software design as the current design:

```
current_sw_design sw_design_1
```

See Also:

- [hsm::get sw designs](#)
- [hsm::create sw design](#)
- [hsm::close sw design](#)

hsm::get_sw_designs

Get a list of software designs opened

Syntax:

```
hsm::get_sw_designs [-regexp] [-filter <arg>] [-quiet] [-verbose]
                    [<patterns>...]
```

Returns:

Software design objects. Returns nothing if the command fails.

Usage:

Name	Description
[-regexp]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<patterns>]	Match design names against patterns Default: *

Categories:

Software

Description:

Gets a list of software designs open in the current session that match a specified search pattern. The default command gets a list of all open software designs in the active session.

Arguments:

-regexp

(Optional) Specifies that the search <patterns> are written as regular expressions. Both search <patterns> and -filter expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add ".*" to the beginning or end of a search string to widen the search to include a substring. See http://www.tcl.tk/man/tcl8.5/TclCmd/re_syntax.htm for help with regular expression syntax.

Note: The Tcl built-in command regexp is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.5/TclCmd/regexp.htm>.

-filter <args>

(Optional) Filter the results list with the specified expression. The `-filter` argument filters the list of objects returned by `get_hw_designs` based on property values on the designs. You can find the properties on an object with the `report_property` or `list_property` commands. In the case of the "Software design" object, "NAME", is the property that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, port, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

Note: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of ""

For string comparison, the specific operators that can be used in filter expressions are "equal" (=), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||).

The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_sw_designs * -filter {&& NAME !~ "*sw*"}
```

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

<patterns>

(Optional) Match designs against the specified patterns. The default pattern is the wildcard '*' which gets all software designs. More than one pattern can be specified to find multiple designs based on different search criteria.

Examples:

The following example gets all open Software designs in the current session:

```
get_sw_designs
```

See Also:

- [hsm::current_sw_design](#)
- [hsm::open_sw_design](#)
- [report_property](#)

hsm::set_repo_path

Set a list of software repository paths

Syntax:

```
hsm::set_repo_path [-quiet] [-verbose] <paths>...
```

Returns:

Returns nothing

Usage:

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<paths>	List of software repository paths separated by spaces

Categories:

Software

Description:

Loads the software cores available in the repository path. Users can specify multiple repository paths.

Arguments:

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the set_msg_config command.

<paths>

List of software repository paths separated by spaces.

Example:

The following example loads the user software cores in the current session:

```
set_repo_path C:/users/user_driver_repo
```

See Also:

- [hsm::get_sw_cores](#)

hsm::get_sw_cores

Get a list of software cores like driver, library and os

Syntax:

```
hsm::get_sw_cores [-regex] [-filter <arg>] [-quiet] [-verbose]
                  [<patterns>...]
```

Returns:

Software core objects. Returns nothing if the command fails.

Usage:

Name	Description
[-regex]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<patterns>]	Match swcore name and versions against patterns Default: *

Categories:

Software

Description:

Get a list of SW cores defined in the SW repository of the current session, based on the specified search pattern. The default is to return all SW cores defined in the repo.

Arguments:

-regex

(Optional) Specifies that the search <patterns> are written as regular expressions. Both search <patterns> and -filter expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add ".*" to the beginning or end of a search string to widen the search to include a substring. See http://www.tcl.tk/man/tcl8.5/TclCmd/re_syntax.htm for help with regular expression syntax.

Note: The Tcl built-in command regex is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.5/TclCmd/regex.htm>.

-filter <args>

Filter the results list with the specified expression. The `-filter` argument filters the list of objects returned by `get_sw_cores` based on property values on the objects. You can find the properties on an object with the `report_property` or `list_property` commands. In the case of the "sw_cores" object, "NAME", "CORE_STATE" and "TYPE" are some of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

Note: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of ""

For string comparison, the specific operators that can be used in filter expressions are "equal" (=), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||). The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_sw_cores * -filter {CORE_STATE == "DEPRECATED" && TYPE == "DRIVER"}
```

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

<patterns>

(Optional) Match IP core definitions in the IP catalog against the specified search patterns. The default pattern is the wildcard "*" which gets a list of all IP cores in the catalog. More than one pattern can be specified to find multiple core definitions based on different search criteria.

Note: You must enclose multiple search patterns in braces, {}, or quotes, "", to present the list as a single element.

Examples:

The following example returns a list of all SW cores with TYPE property matching the specified pattern:

```
get_sw_cores -filter {TYPE == "driver"}
```

The following example returns a list of all SW cores with REPOSITORY property matching the specified pattern:

```
get_sw_cores -filter {REPOSITORY=="C:/user/repo"}
```

See Also:

- [hsm::set_repo_path](#)

hsm::add_library

Add software library to software design

Syntax:

```
hsm::add_library [-sw <arg>] [-quiet] [-verbose] <name> [<version>]
```

Returns:

The Software Library object. Returns nothing if the command fails.

Usage:

Name	Description
[-sw]	Software design name
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<name>	Software library name
[<version>]	Version of software Library

Categories:

Software

Description:

Adds library to the active software design. The software design must previously have been created using the create_sw_design command.

This command returns a message with the name of the library, or returns an error if the command fails.

Arguments:

-sw

(Optional) Name of the software design to which library to the added.

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

<name>

Name of the library.

<version>

(Optional) Version of the library name. Version less library will be picked if version is not specified.

Examples:

The following adds specified Library to the current software design:

```
add_library xilffs
add_library xilrsa 1.0
```

See Also:

- [hsm::current sw design](#)

hsm::delete_objs

Delete specified objects.

Syntax:

```
hsm::delete_objs [-quiet] [-verbose] <objects>...
```

Returns:

Pass if successful in deleting objects.

Usage:

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<objects>	The objects to be deleted

Categories:

Software

Description:

Delete specified objects from the current software design.

Objects must be passed directly to the delete_objs command, and not simply referenced by the object name.

This command returns nothing if it is successful, and returns an error if it fails.

Arguments:

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the set_msg_config command.

<objects>

A list of objects to delete from the current software design.

Example:

The following example deletes the various objects from the current software design:

```
delete_objs [get_libs xilffs] [get_drivers gpio]
```

hsm::get_drivers

Get a list of software driver instances

Syntax:

```
hsm::get_drivers [-regexp] [-filter <arg>] [-of_objects <args>] [-quiet]
[-verbose] [<patterns>...]
```

Returns:

Driver instance objects. Returns nothing if the command fails.

Usage:

Name	Description
[-regexp]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-of_objects]	Get 'driver' objects of these types: 'sw_design'.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<patterns>]	Match cell names against patterns Default: *

Categories:

Software

Description:

Get a list of driver instances in the current software design, Each instance is mapping to the IP instances in the loaded hardware design. Generic driver is assigned for the IPs which does not have drivers.

Arguments:

-regexp

(Optional) Specifies that the search <patterns> are written as regular expressions. Both search <patterns> and -filter expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search to include a substring. See http://www.tcl.tk/man/tcl8.5/TclCmd/re_syntax.htm for help with regular expression syntax.

Note: The Tcl built-in command regexp is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.5/TclCmd/regexp.htm>.

-filter <args>

(Optional) Filter the results list with the specified expression. The `-filter` argument filters the list of objects returned by `get_ells` based on property values on the cells. You can find the properties on an object with the `report_property` or `list_property` commands.

In the case of the driver object, "NAME", "VERSION" and "HW_INSTANCE" are some of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

Note: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of ""

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||).

The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_pins * -filter {DIRECTION == IN && NAME !~ "*RESET*"}
get_drivers * -filter {NAME==gpio && HW_INSTANCE == axi_gpio_0}
```

-of_objects <arg>

(Optional) Get 'driver' objects of these types: 'sw_design'.

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

<patterns>

(Optional) Match software design cells against the specified patterns. The default pattern is the wildcard ``*`` which gets a list of all cells in the current IP subsystem design. More than one pattern can be specified to find multiple cells based on different search criteria.

Note: You must enclose multiple search patterns in braces {} to present the list as a single element.

Examples:

The following example gets a list of drivers that include the specified Software design:

```
get_drivers
```

The following example gets a list of all driver instances of gpio driver:

```
get_drivers * -filter {NAME==gpio}
```

See Also:

- [hsm::get_libs](#)
- [hsm::get_os](#)

hsm::get_libs

Get a list of software libraries

Syntax:

```
hsm::get_libs [-regexp] [-filter <arg>] [-of_objects <args>] [-quiet]
              [-verbose] [<patterns>...]
```

Returns:

Library objects. Returns nothing if the command fails.

Usage:

Name	Description
[-regexp]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-of_objects]	Get 'lib' objects of these types: 'sw_design'.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<patterns>]	Match cell names against patterns Default: *

Categories:

Software

Description:

Get a list of libraries in the current software design.

Arguments:

-regexp

(Optional) Specifies that the search <patterns> are written as regular expressions. Both search <patterns> and -filter expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search to include a substring. See http://www.tcl.tk/man/tcl8.5/TclCmd/re_syntax.htm for help with regular expression syntax.

Note: The Tcl built-in command regexp is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.5/TclCmd/regexp.htm>.

- <args>

(Optional) Filter the results list with the specified expression. The `-filter` argument filters the list of objects returned by `get_cells` based on property values on the cells. You can find the properties on an object with the `report_property` or `list_property` commands. In the case of the library object, "NAME", "VERSION" and "other config parameters" are some of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

Note: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of ""

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||).

The following gets input pins that do NOT contain the "RESET" substring within their name:

```
get_libs * -filter {NAME == xilrsa && VERSION == "1.0"}
```

-of_objects <arg>

(Optional) Get the subsystem cells that are connected to the specified pin or net objects, as returned by the `get_nets` and `get_bd_pins`, or `get_bd_intf_pins` commands.

Note: The `-of_objects` option requires objects to be specified using the `get_*` commands, such as `get_cells` or `get_pins`, rather than specifying objects by name. In addition, `-of_objects` cannot be used with a search `<pattern>`

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

<patterns>

(Optional) Match subsystem cells against the specified patterns. The default pattern is the wildcard ``*`` which gets a list of all cells in the current IP subsystem design. More than one pattern can be specified to find multiple cells based on different search criteria.

Note: You must enclose multiple search patterns in braces {} to present the list as a single element.

Examples:

The following example gets a list of libraries:

```
get_libs
```

See Also:

- [hsm::get_drivers](#)
- [hsm::get_os](#)
- [report_property](#)

hsm::get_os

Get OS in the software design

Syntax:

```
hsm::get_os [-regexp] [-filter <arg>] [-of_objects <args>] [-quiet]
[-verbose] [<patterns>...]
```

Returns:

OS object. Returns nothing if the command fails.

Usage:

Name	Description
[-regexp]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-of_objects]	Get 'os' objects of these types: 'sw_design'.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<patterns>	Match design names against patterns Default: *

Categories:

Software

Description:

Returns OS object on success and nothing if the command fails.

Arguments:

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the set_msg_config command.

Example:

The following example returns OS object of the current software design:

```
get_os
```

See Also:

- [hsm::get_sw_processor](#)
- [hsm::get_drivers](#)
- [hsm::get_libs](#)

hsm::get_sw_processor

Get processor of the software design

Syntax:

```
hsm::get_sw_processor [-regex] [-filter <arg>] [-of_objects <args>]
[-quiet] [-verbose] [<patterns>...]
```

Returns:

Processor object. Returns nothing if the command fails.

Usage:

Name	Description
[-regex]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-of_objects]	Get 'sw_proc' objects of these types: 'sw_design'.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<patterns>]	Match design names against patterns Default: *

Categories:

Software

Description:

Returns the processor object of the active software design and nothing if the command fails.

Arguments:

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the set_msg_config command.

Example:

The following example returns the software processor of the current software design:

```
get_sw_processor
```

See Also:

- [hsm::get_os](#)
- [hsm::get_drivers](#)
- [hsm::get_libs](#)

hsm::get_arrays

Get a list of software Arrays

Syntax:

```
hsm::get_arrays [-regexp] [-filter <arg>] [-of_objects <args>] [-quiet]
                [-verbose] [<patterns>...]
```

Returns:

Array objects. Returns nothing if the command fails.

Usage:

Name	Description
[-regexp]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-of_objects]	Get 'array' objects of these types: 'sw_proc os driver lib sw_core'.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<patterns>]	Match array names against patterns Default: *

Categories:

Software

Description:

Arrays are defined in MDD/MLDs. It contains any number of PARAMs and PROPERTYs which describes size, description of array and default values of elements in array.

Arguments:

-regexp

(Optional) Specifies that the search <patterns> are written as regular expressions. Both search <patterns> and -filter expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search to include a substring. See http://www.tcl.tk/man/tcl8.5/TclCmd/re_syntax.htm for help with regular expression syntax.

Note: The Tcl built-in command regexp is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.5/TclCmd/regexp.htm>.

-filter <args>

(Optional) Filter the results list with the specified expression. The `-filter` argument filters the list of objects returned by `get_arrays` based on property values on the arrays. You can find the properties on an object with the `report_property` or `list_property` commands. In the case of the array object, "NAME", and "other config parameters" are some of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

Note: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of ""

For string comparison, the specific operators that can be used in filter expressions are "equal" (=), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||).

-of_objects <arg>

(Optional) Get the arrays that are available in OS, Drivers, Libraries, Processor, Core, as returned by the `get_os`, `get_drivers`, `get_libs`, `get_sw_processor`, `get_sw_cores` commands.

Note: The `-of_objects` option requires objects to be specified using the `get_*` commands, such as `get_os` or `get_libs`, rather than specifying objects by name. In addition, `-of_objects` cannot be used with a search `<pattern>`

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns `TCL_OK` regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

<patterns>

(Optional) Match arrays against the specified patterns. The default pattern is the wildcard "*" which gets a list of all cells in the current IP subsystem design. More than one pattern can be specified to find multiple cells based on different search criteria.

Note: You must enclose multiple search patterns in braces {} to present the list as a single element.

Example:

The following example gets a list of arrays present in all software cores(drivers/libs/os)

```
get_arrays
```

The following example gets a list of all arrays matching the name "mem_table"

```
get_arrays mem_table
```

The following example gets a list of arrays present in OS of current software design.

```
get_arrays -of_objects [get_os]
```

See Also

- [hsm::get_sw_interfaces](#)

hsm::get_sw_interfaces

Get a list of software Interfaces

Syntax:

```
hsm::get_sw_interfaces [-regexp] [-filter <arg>] [-of_objects <args>] [-quiet]
                        [-verbose] [<patterns>...]
```

Returns:

Software Interface objects. Returns nothing if the command fails.

Usage:

Name	Description
[-regexp]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-of_objects]	Get 'interface' objects of these types: 'sw_proc os driver lib sw_core'.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<patterns>]	Match interface names against patterns Default: *

Categories:

Software

Description:

Specifies the interfaces implemented by the library or driver and describes the interface functions and header files used by the library or driver.

Arguments:

-regexp

(Optional) Specifies that the search <patterns> are written as regular expressions. Both search <patterns> and -filter expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search to include a substring. See http://www.tcl.tk/man/tcl8.5/TclCmd/re_syntax.htm for help with regular expression syntax.

Note: The Tcl built-in command regexp is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.5/TclCmd/regexp.htm>.

-filter <args>

(Optional) Filter the results list with the specified expression. The `-filter` argument filters the list of objects returned by `get_sw_interfaces` based on property values on the interfaces. You can find the properties on an object with the `report_property` or `list_property` commands. In the case of the interface object, "NAME", and "other config parameters" are some of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

Note: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of ""

For string comparison, the specific operators that can be used in filter expressions are "equal" (=), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||).

-of_objects <arg>

(Optional) Get the software interfaces that are available in OS, Drivers, Libraries, Processor, Core, as returned by the `get_os`, `get_drivers`, `get_libs`, `get_sw_processor`, `get_sw_cores` commands.

Note: The `-of_objects` option requires objects to be specified using the `get_*` commands, such as `get_os` or `get_libs`, rather than specifying objects by name. In addition, `-of_objects` cannot be used with a search `<pattern>`

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns `TCL_OK` regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

<patterns>

(Optional) Match interfaces against the specified patterns. The default pattern is the wildcard ``*`` which gets a list of all cells in the current IP subsystem design. More than one pattern can be specified to find multiple cells based on different search criteria.

Note: You must enclose multiple search patterns in braces `{}` to present the list as a single element.

Example:

The following example gets a list of interfaces present in all software cores(drivers/libs/os)

```
get_sw_interfaces
```

The following example gets a list of all interfaces matching the name "stdout"

```
get_sw_interfaces stdout
```

The following example gets a list of interfaces present in OS of current software design.

```
get_sw_interfaces -of_objects [get_os]
```

See Also:

- [hsm::get_arrays](#)

hsm::get_nodes

Get a list of child nodes

Syntax:

```
get_nodes [-regexp] [-filter <arg>] [-of_objects <args>] [-quiet]
[-verbose] [<patterns>...]
```

Returns:

Node objects. Returns nothing if the command fails.

Usage:

Name	Description
[-regexp]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-of_objects]	Get 'node' objects of these types: 'driver sw_proc os node'.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<patterns>]	Match cell names against patterns Default: *

Categories:

Software

Description:

Get a list of nodes in drivers/os/nodes in the current software design.

A node can have child nodes in it.

Arguments:

-regexp

(Optional) Specifies that the search <patterns> are written as regular expressions. Both search <patterns> and -filter expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search to include a substring. See http://www.tcl.tk/man/tcl8.5/TclCmd/re_syntax.htm for help with regular expression syntax.

Note: The Tcl built-in command regexp is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.5/TclCmd/regexp.htm>.

filter <args>

(Optional) Filter the results list with the specified expression. The `-filter` argument filters the list of objects returned by `get_cells` based on property values on the cells. You can find the properties on an object with the `report_property` or `list_property` commands.

In the case of the node object, "NAME", "PARENT" are some of the properties that can be used to filter results.

The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

Note: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "" wildcard character, this will match a property with a defined value of ""

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||).

The following gets nodes that matches NAME and PARENT within their name:

```
get_nodes -filter {NAME==clkc && PARENT == ps7_slcr_0}
-of_objects <arg> - (Optional) Get 'node' objects of these types:
'sw_driver', 'sw_os', 'sw_proc', 'sw_node'.
```

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

<patterns>

(Optional) Match software design cells against the specified patterns. The default pattern is the wildcard "*" which gets a list of all cells in the current IP subsystem design. More than one pattern can be specified to find multiple cells based on different search criteria.

Note: You must enclose multiple search patterns in braces {} to present the list as a single element.

Examples:

The following example gets a list of nodes that include the specified driver in all software cores(drivers/libs/os)

Software design:

```
get_nodes -of_objects [get_drivers ps7_uart_0]
```

The following example gets a list of all nodes of OS:

```
get_nodes -of_objects [get_os]
```

See Also:

- [hsm::get_comp_params](#)
- [hsm::get_drivers](#)

hsm::get_comp_params

Get a list of params in drivers/os/nodes in the current software design.

Arguments:

-regexp

(Optional) Specifies that the search <patterns> are written as regular expressions. Both search <patterns> and -filter expressions must be written as regular expressions when this argument is used. Xilinx regular expression Tcl commands are always anchored to the start of the search string. You can add "." to the beginning or end of a search string to widen the search to include a substring. See http://www.tcl.tk/man/tcl8.5/TclCmd/re_syntax.htm for help with regular expression syntax.

Note: The Tcl built-in command regexp is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.5/TclCmd/regexp.htm>.

-filter <args>

(Optional) Filter the results list with the specified expression. The -filter argument filters the list of objects returned by get_cells based on property values on the cells. You can find the properties on an object with the report_property or list_property commands.

In the case of the param object, "NAME", "VALUE" are some of the properties that can be used to filter results. The filter search pattern should be quoted to avoid having to escape special characters that may be found in net, pin, or cell names, or other properties. String matching is case-sensitive and is always anchored to the start and to the end of the search string. The wildcard "*" character can be used at the beginning or at the end of a search string to widen the search to include a substring of the property value.

Note: The filter returns an object if a specified property exists on the object, and the specified pattern matches the property value on the object. In the case of the "*" wildcard character, this will match a property with a defined value of ""

For string comparison, the specific operators that can be used in filter expressions are "equal" (==), "not-equal" (!=), "match" (=~), and "not-match" (!~). Numeric comparison operators <, >, <=, and >= can also be used. Multiple filter expressions can be joined by AND and OR (&& and ||).

The following gets params that matches NAME and VALUE within their name:

```
get_comp_params -filter {NAME == clock-names && VALUE == "ref_clk
aper_clk"}
-of_objects <arg> - (Optional) Get 'param' objects of these types:
'sw_driver', 'sw_os', 'sw_proc', 'sw_node'.
```

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

<patterns>

(Optional) Match software design cells against the specified patterns. The default pattern is the wildcard `*`` which gets a list of all cells in the current IP subsystem design. More than one pattern can be specified to find multiple cells based on different search criteria.

Note: You must enclose multiple search patterns in braces `{}` to present the list as a single element.

Examples:

The following example gets a list of params that include the specified driver in Software design:

```
get_comp_params -of_objects [get_drivers ps7_uart_0]
```

The following example gets a list of all params of OS:

```
get_comp_params -of_objects [get_os]
```

See Also:

- [hsm::get_nodes](#)
- [hsm::get_drivers](#)

hsm::create_node

Create a new node to list of existing nodes (driver/os/prco/node).

Syntax:

```
create_node [-quiet] [-verbose] <name> <objects>
```

Returns:

Node object. Returns nothing if the command fails.

Usage:

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<name>	Child node name
<objects>	List of nodes

Categories:

Software

Description:

Create a new node to list of existing nodes(driver/os/prco/node).

This command returns the name of the node created if the command is successful. An error is returned if the command fails.

Arguments:

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the set_msg_config command.

<name>

The name of the node to create.

<objects>

List of nodes to which new node is created

Example:

The following example creates a new node called n1 to specified driver

```
create_node n1 [get_drivers ps7_uart_1]
```

The following example creates a new node called n2 to all drivers

```
create_node n2 [get_drivers]
```

See Also:

- [hsm::create_comp_param](#)
- [hsm::create_sw_design](#)

hsm::create_comp_param

Create a new param to list of nodes (driver/os/proc/node).

Syntax:

```
create_comp_param [-quiet] [-verbose] <name> <value> <objects>
```

Returns:

Parameter object. Returns nothing if the command fails.

Usage:

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<name>	Parameter name
<value>	Parameter value
<objects>	List of nodes

Categories:

Software

Description:

Create a new param to list of nodes(driver/os/proc/node).

This command returns the name of the param created if the command is successful. An error is returned if the command fails.

Arguments:

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the set_msg_config command.

<name>

The name of the param to create.

<value>

The value of the param to create. Default type of param is string.

<objects>

List of nodes to which new param is created

Example:

The following example creates a new param called p1 to specified driver

```
create_comp_param p1 [get_drivers ps7_uart_1]
```

The following example creates a new param called p2 to all drivers

```
create_comp_param p2 [get_drivers]
```

See Also:

- [hsm::create_node](#)
- [hsm::create_sw_design](#)

HSM Property and Parameter TCL Commands

create_property

Create property for class of objects(s)

Syntax:

```
create_property [-description <arg>] [-type <arg>] [-enum_values <args>]
               [-default_value <arg>] [-file_types <args>]
               [-display_text <arg>] [-quiet] [-verbose] <name> <class>
```

Returns:

The property that was created if success; "" if failure

Usage:

Name	Description
[-description]	Description of property
[-type]	Type of property to create; valid values are: string, int, long, double, bool, enum, file Default: string
[-enum_values]	Enumeration values
[-default_value]	Default value of type string
[-file_types]	File type extensions (without the dot)
[-display_text]	Text to display when selecting the file in file browser
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<name>	Name of property to create
<class>	Object type to create property for; valid values are: design, net, cell, pin, port, pblock, interface, fileset

Categories:

PropertyAndParameter

Description:

Creates a new property of the <type> specified with the user-defined <name> for the specified <class> of objects. The property that is created can be assigned to an object of the specified class with the set_property command, but is not automatically associated with all objects of that class.

The `report_property -all` command will not report the newly created property for an object of the specified class until the property has been assigned to that object.

Arguments:

-description <arg>

(Optional) Provide a description of the property being created. The description will be returned by the HSM help system when the property is queried.

-type <arg>

(Optional) The type of property to create. There allowed property types include:

string

Allows the new property to be defined with string values. This is the default value when -type is not specified.

int

Allows the new property to be defined with short four-byte signed integer values with a range of -2,147,483,648 to 2,147,483,647. If a floating point value is specified for an int property type, the HSM tool will return an error.

long

Specifies signed 64-bit integers with value range of -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. If a floating point value is specified for an long property type, the tool will return an error.

double

Allows the new property value to be defined with a floating point number.

bool

Allows the new property to be defined as a boolean with a true (1, or yes) or false (0, or no) value.

enum

An enumerated data type, with the valid enumerated values specified by the `-enum_values` option.

string_list

A Tcl list of string values.

int_list

A Tcl list of integer values.

double_list

A Tcl list of floating point values.

-enum_values <args>

(Optional) A list of enumerated values that the property can have. The list must be enclosed in braces, {}, with values separated by spaces. This option can only be used with -type enum.

-default_value <args>

(Optional) The default value to assign to the property. This option can be used for string, int, bool, and enum type properties.

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

<name>

(Required) The name of the property to be defined. The name is case sensitive.

<class>

(Required) The class of object to assign the new property to. All objects of the specified class will be assigned the newly defined property.

Valid classes are: design, net, cell, pin, port, Pblock, interface, and fileset.

Examples:

The following example defines a property called PURPOSE for cell objects:

```
create_property PURPOSE cell
```

Note: Because the -type was not specified, the value will default to strings.

The following example creates a pin property called COUNT which holds an Integer value:

```
create_property -type int COUNT pin
```

See Also:

- [get_property](#)
- [list_property](#)
- [list_property_value](#)
- [report_property](#)
- [reset_property](#)
- [set_property](#)

get_param

Get a parameter value

Syntax:

```
get_param [-quiet] [-verbose] <name>
```

Returns:

parameter value

Usage:

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<name>	Parameter name

Categories:

PropertyAndParameter

Description:

This command gets the currently defined value for a specified tool parameter. These parameters are user-definable configuration settings that control various behaviors within the tool. Refer to `report_param` for a description of what each parameter configures or controls.

Arguments:

<name>

(Required) The name of the parameter to get the value of. The list of user-definable parameters can be obtained with `list_param`. This command requires the full name of the desired parameter. It does not perform any pattern matching, and accepts only one parameter at a time.

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns `TCL_OK` regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

Examples:

The following example returns the current value of the Messaging parameter used for enabling the description:

```
get_param messaging.enableDescription
```

See Also:

- [list_param](#)
- [report_param](#)
- [reset_param](#)
- [set_param](#)

get_property

Get properties of object.

Syntax:

```
get_property [-min] [-max] [-quiet] [-verbose] <name> <object>
```

Returns:

property value

Usage:

Name	Description
[-min]	Return only the minimum value
[-max]	Return only the maximum value
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<name>	Name of property whose value is to be retrieved
<object>	Object to query for properties

Categories:

Object, PropertyAndParameter

Description:

Gets the current value of the named property from the specified object or objects. If multiple objects are specified, a list of values is returned.

If the property is not currently assigned to the object, or is assigned without a value, then the `get_property` command returns nothing, or the null string. If multiple objects are queried, the null string is added to the list of values returned.

This command returns a value, or list of values, or returns an error if it fails.

Arguments:

-min

(Optional) When multiple <objects> are specified, this option examines the values of the property specified by <name>, and returns the smallest value from the list of objects. Numeric properties are sorted by value. All other properties are sorted as strings.

-max

(Optional) When multiple <objects> are specified, this option examines the values of the property specified by <name>, and returns the largest value from the list of objects. Numeric properties are sorted by value. All other properties are sorted as strings.

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

<name>

(Required) The name of the property to be returned. The name is not case sensitive.

<objects>

(Required) One or more objects to examine for the specified property.

Examples:

The following example gets the NAME property from the specified cell:

```
get_property NAME [lindex [get_cells] 0]
```

The following example returns the BOARD property from the current hardware design:

```
get_property BOARD [current_hw_design]
```

See Also:

- [create_property](#)
- [hsm::get_cells](#)
- [hsm::get_ports](#)
- [list_property](#)
- [list_property_value](#)
- [report_property](#)
- [reset_property](#)
- [set_property](#)

list_param

Get all parameter names

Syntax:

```
list_param [-quiet] [--verbose]
```

Returns:

list

Usage:

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories:

PropertyAndParameter

Description:

Gets a list of user-definable configuration parameters. These parameters configure a variety of settings and behaviors of the tool. For more information on a specific parameter use the `report_param` command, which returns a description of the parameter as well as its current value.

Arguments:

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns `TCL_OK` regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

Examples:

The following example returns a list of all user-definable parameters:

```
list_param
```

See Also:

- [get_param](#)
- [report_param](#)
- [reset_param](#)
- [set_param](#)

list_property

List properties of object

Syntax:

```
list_property [-class <arg>] [-regex] [-quiet] [-verbose] [<object>]
              [<pattern>]
```

Returns:

list of property names

Usage:

Name	Description
[-class]	Object type to query for properties. Ignored if object is specified.
[-regex]	Pattern is treated as a regular expression
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<object>]	Object to query for properties
[<pattern>]	Pattern to match properties against Default: *

Categories:

PropertyAndParameter

Description:

Gets a list of all properties on a specified object or class.

Note: report_property also returns a list of properties on an object or class of objects, but also reports the property type and property value.

Arguments:

-class <arg>

(Optional) Return the properties of the specified class instead of a specific object. The class argument is case sensitive, and most class names are lower case.

Note: -class cannot be used together with an <object>

-regex

(Optional) Specifies that the search <pattern> is written as a regular expression.

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

<object>

(Optional) A single object on which to report properties.

Note: If you specify multiple objects you will get an error.

<pattern>

(Optional) Match the available properties on the `<object>` or `-class` against the specified search pattern. The `<pattern>` applies to the property name, and only properties matching the specified pattern will be reported. The default pattern is the wildcard ``*`` which returns a list of all properties on the specified object.

Note: The search pattern is case sensitive, and most properties are UPPER case.

Examples:

The following example returns all properties of the specified CELL object:

```
list_property [get_cells microblaze_0]
```

See Also:

- [create_property](#)
- [hsm::get_cells](#)
- [get_property](#)
- [list_property_value](#)
- [report_property](#)
- [reset_property](#)
- [set_property](#)

list_property_value

List legal property values of object

Syntax:

```
list_property_value [-default] [-class <arg>] [-quiet] [-verbose] <name>
                    [<object>]
```

Returns:

list of property values

Usage:

Name	Description
[-default]	Show only the default value.
[-class]	Object type to query for legal property values. Ignored if object is specified.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<name>	Name of property whose legal values is to be retrieved
[<object>]	Object to query for legal properties values

Categories:

Object, PropertyAndParameter

Description:

Gets a list of valid values for an enumerated type property of either a class of objects or a specific object.

Note: The command cannot be used to return valid values for properties other than Enum properties. The report_property command will return the type of property to help you identify Enum properties.

Arguments:

-default

(Optional) Return the default property value for the specified class of objects.

-class <arg>

(Optional) Return the property values of the specified class instead of a specific object. The class argument is case sensitive, and most class names are lower case.

Note: -class cannot be used together with an <object>

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

<name>

(Required) The name of the property to be queried. Only properties with an enumerated value, or a predefined value set, can be queried with this command. All valid values of the specified property will be returned.

<object>

(Optional) A single object on which to report properties.

Note: If you specify multiple objects you will get an error.

Examples:

The following example returns the same result, but uses an actual cell object in place of the general cell class:

```
list_property_value IP_TYPE [get_cells microblaze_0]
```

The following example returns the default value for the specified property by using the current design as a representative of the design class:

```
list_property_value -default PATH [current_hw_design]
```

See Also:

- [create_property](#)
- [hsm::current_hw_design](#)
- [hsm::get_cells](#)
- [get_property](#)
- [list_property](#)
- [report_property](#)
- [reset_property](#)
- [set_property](#)

report_param

Get information about all parameters

Syntax:

```
report_param [-file <arg>] [-append] [-non_default] [-return_string]
[-quiet] [-verbose] [<pattern>]
```

Returns:

param report

Usage:

Name	Description
[-file]	Filename to output results to. (send output to console if -file is not used)
[-append]	Append the results to file, don't overwrite the Results file
[-non_default]	Report only params that are set to a non default value
[-return_string]	Return report as string
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<pattern>]	Display params matching pattern Default: *

Categories:

PropertyAndParameter, Report

Description:

Gets a list of all user-definable parameters, the current value, and a description of what the parameter configures or controls.

Arguments:

-file <arg>

(Optional) Write the report into the specified file. The specified file will be overwritten if one already exists, unless -append is also specified.

Note: If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

-append

(Optional) Append the output of the command to the specified file rather than overwriting it.

Note: The -append option can only be used with the -file option

-return_string

(Optional) Directs the output to a Tcl string rather than to the standard output. The Tcl string can be captured by a variable definition and parsed or otherwise processed.

Note: This argument cannot be used with the -file option.

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the set_msg_config command.

<pattern>

(Optional) Match parameters against the specified pattern. The default pattern is the wildcard ``*`` which gets all user-definable parameters.

Examples:

The following example returns the name, value, and description of all user-definable parameters:

```
report_param
```

The following example returns the name, value, and description of user-definable parameters that match the specified search pattern:

```
report_param *coll*
```

See Also:

- [get_param](#)
- [list_param](#)
- [reset_param](#)
- [set_param](#)

report_property

Report properties of object

Syntax:

```
report_property [-all] [-class <arg>] [-return_string] [-file <arg>]
[-append] [-regexp] [-quiet] [-verbose] [<object>] [<pattern>]
```

Returns:

property report

Usage:

Name	Description
[-all]	Report all properties of object even if not set
[-detail]	HIDDEN. Report detailed help on properties
[-class]	Object type to query for properties. Not valid with <object>
[-list_owners]	HIDDEN. Report the owner of each property
[-return_string]	Set the result of running report_property in the Tcl interpreter's result variable
[-file]	Filename to output result to. Send output to console if -file is not used
[-append]	Append the results to file, don't overwrite the results file
[-regexp]	Pattern is treated as a regular expression
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<object>]	Object to query for properties
[<pattern>]	Pattern to match properties against Default: *

Categories:

Object, PropertyAndParameter, Report

Description:

Gets the property name, property type, and property value for all of the properties on a specified object, or class of objects.

Note: list_property also returns a list of all properties on an object, but does not include the property type or value.

You can specify objects for report_property using the get_* series of commands to get a specific object. You can use the lindex command to return a specific object from a list of objects:

```
report_property [lindex [get_cells] 0]
```

However, if you are looking for the properties on a class of objects, you should use the `-class` option instead of an actual object.

This command returns a report of properties on the object, or returns an error if it fails.

Arguments:

-all

(Optional) Return all of the properties for an object, even if the property value is not currently defined.

-class <arg>

(Optional) Return the properties of the specified class instead of a specific object. The class argument is case sensitive, and most class names are lower case.

Note: `-class` cannot be used together with an `<object>`

-return_string

(Optional) Directs the output to a Tcl string. The Tcl string can be captured by a variable definition and parsed or otherwise processed.

-file <arg>

(Optional) Write the report into the specified file. The specified file will be overwritten if one already exists, unless `-append` is also specified.

Note: If the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.

-append

(Optional) Append the output of the command to the specified file rather than overwriting it.

Note: The `-append` option can only be used with the `-file` option

-regexp

(Optional) Specifies that the search `<pattern>` is written as a regular expression.

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns `TCL_OK` regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

<object>

(Optional) A single object on which to report properties.

Note: If you specify multiple objects you will get an error.

<pattern>

(Optional) Match the available properties on the <object> or -class against the specified search pattern. The <pattern> applies to the property name, and only properties matching the specified pattern will be reported. The default pattern is the wildcard ``*` which returns a list of all properties on the specified object.

Note: The search pattern is case sensitive, and most properties are UPPER case.

Examples:

The following example returns all properties of the specified object:

```
report_property -all [get_cells microblaze_0]
```

To determine which properties are available for the different design objects supported by the tool, you can use multiple report_property commands in sequence. The following example returns all properties of the specified current objects:

```
report_property -all [current_hw_design]
report_property -all [current_sw_design]
```

See Also:

- [create_property](#)
- [hsm::get_cells](#)
- [get_property](#)
- [list_property](#)
- [list_property_value](#)
- [reset_property](#)
- [set_property](#)

reset_param

Reset a parameter

Syntax:

```
reset_param [-quiet] [-verbose] <name>
```

Returns:

original value

Usage:

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<name>	Parameter name

Categories:

PropertyAndParameter

Description:

Restores a user-definable configuration parameter that has been changed with the set_param command to its default value.

You can use the report_param command to see which parameters are currently defined.

Arguments:

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the set_msg_config command.

<name>

(Required) The name of a parameter to reset. You can only reset one parameter at a time.

Examples:

The following example restores the tcl.statsThreshold parameter to its default value:

```
reset_param tcl.statsThreshold
```

See Also:

- [get_param](#)
- [list_param](#)
- [report_param](#)
- [set_param](#)

reset_property

Reset property on object(s)

Syntax:

```
reset_property [-quiet] [-verbose] <property_name> <objects>...
```

Returns:

The value that was set if success, "" if failure

Usage:

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<property_name>	Name of property to reset
<objects>	Objects to set properties

Categories:

Object, PropertyAndParameter

Description:

Restores the specified property to its default value on the specified object or objects. If no default is defined for the property, the property is unassigned on the specified object.

Arguments:

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the set_msg_config command.

<property_name>

(Required) The name of the property to be reset.

<objects>

(Required) One or more objects on which the property will be restored to its default value.



Examples:

The following example sets the archiver property on the specified processor, and then resets the property:

```
set_property CONFIG.archiver armar [get_sw_processor]
reset_property CONFIG.archiver armar [get_sw_processor]
```

See Also:

- [create_property](#)
- [hsm::get_cells](#)
- [get_property](#)
- [list_property](#)
- [list_property_value](#)
- [report_property](#)
- [set_property](#)

set_param

Set a parameter value

Syntax:

```
set_param [-quiet] [-verbose] <name> <value>
```

Returns:

newly set parameter value

Usage:

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<name>	Parameter name
<value>	Parameter value

Categories:

PropertyAndParameter

Description:

Sets the value of a user-definable configuration parameter. These parameters configure and control various behaviors of the tool. Refer to report_param for a description of currently defined parameters.

You can use the reset_param command to restore any parameter that has been modified back to its default setting.

Note: Setting a specified parameter value to -1 will disable the feature.

Arguments:

<name>

(Required) The name of the parameter to set the value of. You can only set the value of one parameter at a time.

<value>

(Required) The value to set the specified parameter to.



-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

Examples:

```
set_param messaging.defaultLimit 1000
```

See Also:

- [get_param](#)
- [list_param](#)
- [report_param](#)
- [reset_param](#)

set_property

Set property on object(s)

Syntax:

```
set_property [-quiet] [-verbose] <name> <value> <objects>...
```

Usage:

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<name>	Name of property to set
<value>	Value of property to set
<objects>	Objects to set properties on

Categories:

Object, PropertyAndParameter

Description:

Assigns the defined property <name> and <value> to the specified <objects>.

This command can be used to define any property on an object in the design. Each object has a set of predefined properties that have expected values, or a range of values. The set_property command can be used to define the values for these properties. To determine the defined set of properties on an object, use report_property, list_property, or list_property_values.

You can also define custom properties for an object, by specifying a unique <name> and <value> pair for the object. If an object has custom properties, these will also be reported by the report_property and list_property commands.

This command returns nothing if successful, and an error if it fails.

Note: You can use the get_property command to validate any properties that have been set on an object

Arguments:

-dict

(Optional) Use this option to specify multiple properties (<name> <value> pairs) on an object with a single set_property command. Multiple <name> <value> pairs must be enclosed in braces, {}, or quotes, "".

-dict

```
"name1 value1 name2 value2 ... nameN valueN"
```

-quiet

(Optional) Execute the command quietly, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

Note: Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

-verbose

(Optional) Temporarily override any message limits and return all messages from this command.

Note: Message limits can be defined with the `set_msg_config` command.

<name>

(Required) Specifies the name of the property to be assigned to the object or objects. The `<name>` argument is case sensitive and should be specified appropriately.

<value>

(Required) Specifies the value to assign to the `<name>` on the specified object or objects. The value is checked against the property type to ensure that the value is valid. If the value is not appropriate for the property an error will be returned.

Note: In some cases the value of a property may include special characters, such as the dash character (`-`), which can cause the tool to interpret the value as a new argument to the command. In this case, you must use the explicit arguments (`-name`, `-value`, `-objects`) instead of the implied positional arguments (`name`, `value`, `objects`) as described here. This is shown in the Examples section below

<objects>

(Required) One or more objects to assign the property to.

Examples:

Create a user-defined boolean property, TRUTH, for cell objects, and set the property on a cell:

```
create_property -type bool truth cell
set_property truth false [lindex [get_cells] 1]
```

The following example sets the compiler and archiver property value for the specified software processor:

```
set_property CONFIG.archiver armar [get_sw_processor]
set_property CONFIG.compiler armcc [get_sw_processor]
```

See Also:

- [create_property](#)
- [hsm::get_cells](#)
- [get_property](#)
- [list_property](#)
- [list_property_value](#)
- [report_property](#)
- [reset_property](#)

Other HSM Equivalent Tcl Procedures for Libgen Driver Tcl

Current Libgen TCL Commands	HSM TCL Commands
xget_hw_busif_handle <handle> <busif name>	get_bus_intfs <busif_name> -of_objects <handle>
xget_hw_busif_value <handle> <busif_name>	get_property BUS_NAME [get_bus_intfs <busif_name> -of_objects <handle>]
xget_hw_ipinst_handle <handle> <ipinst name>	get_cells <ipinst_name> -of_bojects <handle>
xget_hw_name <handle>	get_property NAME <handle>
xget_hw_value <inhandle>	set class [get_property class \$inhandle] if { \$class == "hsm_cell" } { return [get_property ip_name \$inhandle] } elseif { \$class == "hsm_port" } { return [get_property net_name \$inhandle] } elseif { \$class == "hsm_bus_intf" } { return [get_property bus_name \$inhandle] } else { #throw error, parameter and others are not handled return "" }
xget_hw_proc_slave_periphs <merged_proc handle>	get_property slaves <proc_handle>
xget_hw_port_handle <handle> <port name>	get_ports <port_name> -of_objects <port handle>
xget_hw_port_value <handle> <port name>	get_property SIG_NAME [get_ports <port name> -of_objects <port handle>]
xget_hw_connected_busifs_handle <merged_mhs_handle> <businst_name> <busif_type>	get_bus_intfs -conn_name <businst_name> -of_objects <hw_db_handle> -filter "TYPE == <busif type>"
xget_hw_connected_ports_handle <merged_mhs_handle> <connector name> <port type>	get_ports -conn_name <connector_name> - of_objects <hw_db_handle> -filter "TYPE == <port type>"
xget_hw_parameter_handle <handle> <parameter_name>	#there is no equivalent command in HSM
xget_hw_parameter_value <handle> <parameter_name>	get_property CONFIG.<parameter_name> <cell handle>
xget_hw_bus_slave_addrpairs <merged_bus_handle>	#There is no equivalent command in HSM, as there is no way to get address range values from BUSIF handle
xget_hw_subproperty_value <property_handle> <subprop name>	get_property <subprop_name> <property_handle>
xget_hwhandle <ip name>	get_cells <ip name>