Generating Basic Software Platforms

Reference Guide

UG1138 (v2018.2) July 16, 2018

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
July 16, 2018	2018.2	Updated copyright statements and logos. Some reformatting. No technical content changes.
April 6, 2016	2016.1	Updated MLB keywords section with OS_TYPE keyword.
September 30, 2015	2015.3	Updated Hardware Design File (.hdf) section.
April 1, 2015	2015.1	• Updated Pre-Synthesis Hardware Handoff Project Flow Images • Updated Post-Bitstream Tcl Projectless flow • Added name space to common Tcl commands. Existing scripts without namespaces in the commands work fine. However, using namespaces in all the commands is recommended to avoid any conflicts in future. • Updated Tcl Examples section with advanced configuration and multi- block design • Added appendix C for generating software outputs from within Vivado • Added Appendix G for customized template applications (MAD)
November 19, 2014	2014.4	First version of the document

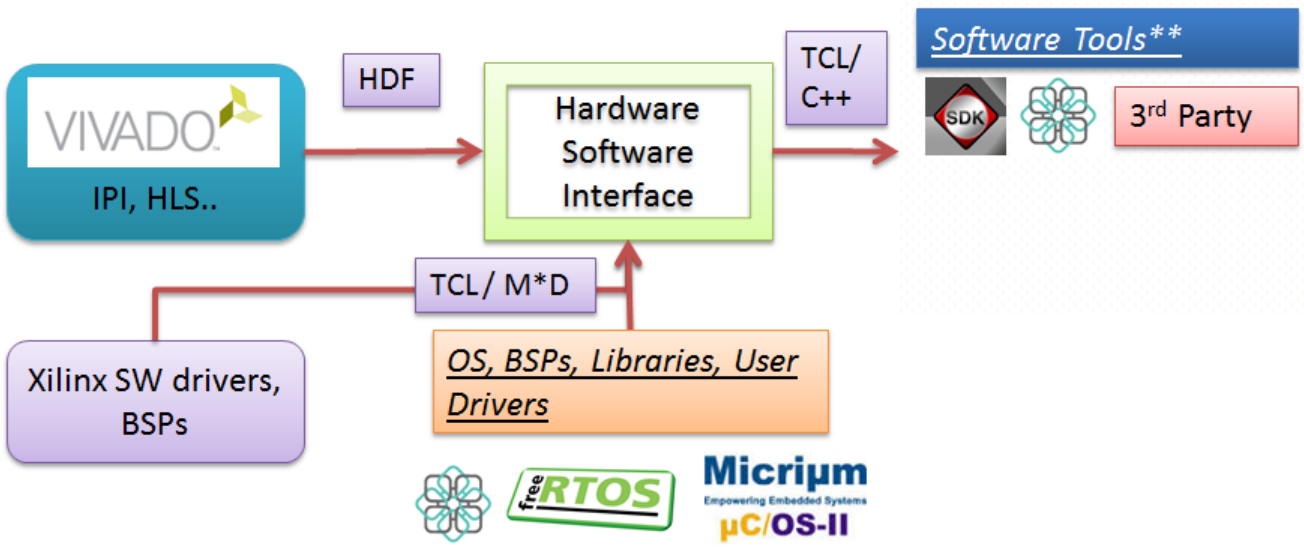
XILINX

Table of Contents

Introduction

Hardware Software Interface (HSI) is a scalable framework enabling embedded SW tool integration with Vivado. It enables third-party OS vendors and software providers to distribute their software for Xilinx FPGA Platforms. HSI consumes hardware design (.hdf) files and the software repository (Drivers, OS, board support packages (BSPs), Libs, Apps, and DTG). It provides a rich set of Tcl APIs to access hardware information and to generate BSPs, Device Tree, and template applications.

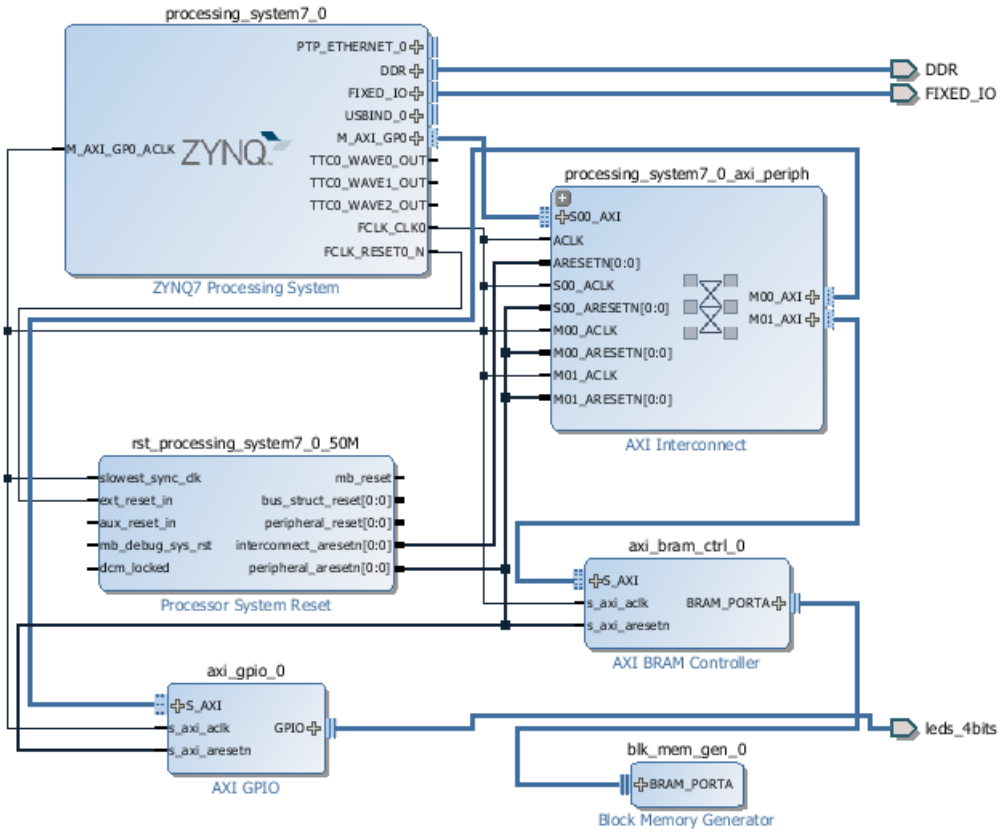
Figure 1: **Design Flow Using HSI**



Hardware Handoff

This chapter describes the Vivado hardware handoff flow for the pre-synthesis and post-bitstream designs. The figure below shows the IP integrator Zynq ZC702 example design.

Figure 2: **Example Zynq Design and Bus Functional Simulation**



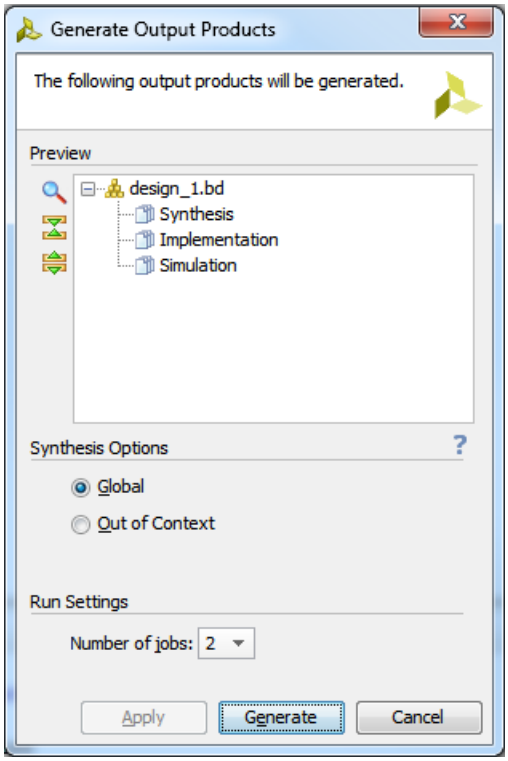
CAUTION!: *Vivado hardware handoff flow supports only single Block Diagram and independent multi Block Diagram designs. It does not support RTL, Reference Block Diagram, and dependent multi-Block Diagram designs.*

Pre-Synthesis Hardware Handoff

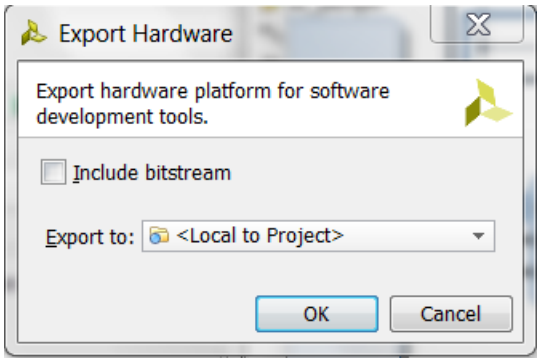
GUI Project Flow

Do the following to perform pre-synthesis hardware handoff in the Vivado interface.

1. Generate the block design. To do this:
   a. In the **Flow Navigator** under **IP Integrator**, click **Generate Block Design**. The Generate Output Products dialog box opens.



   b. Click **Generate** to generate the block design.
2. Export the hardware design. To do this:
   a. Select **File > Export > Export Hardware**. The Export Hardware dialog box opens.



b. Leave the **Include bitstream** check box unchecked.

c. Click **OK** to export the hardware design.

Tcl Projectless Flow

Use the following Tcl commands to perform pre-synthesis hardware handoff using Tcl commands outside of the Vivado project.

1. `create_project -in_memory -part xc7z020clg484-1`
2. `set_property board_part xilinx.com:zc702:part0:1.0 [current_project]`
3. `read_bd base_zynq_design.bd`
4. `read_vhd base_zynq_design_wrapper.vhd`
5. `generate_target all [get_files base_zynq_design.bd]`
6. `write_hwdef -file base_zynq_design_wrapper.hdf`

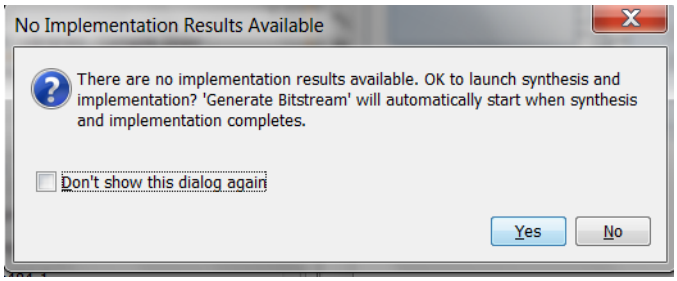
Post-Bitstream Hardware Handoff

GUI Project Flow

Do the following to perform post-bitstream hardware handoff in the Vivado interface.

1. Generate the block design. To do this:

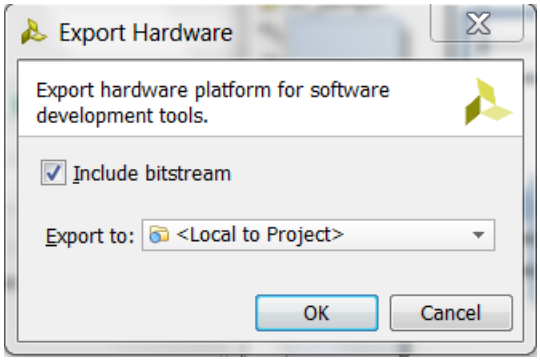
   a. In the **Flow Navigator** under **Program and Debug**, click **Generate Bitstream**. The following dialog box opens.



b. Click **Yes** to launch synthesis and implementation.When synthesis and implementation complete, the Generate Bitstream process automatically runs.

2. Export the hardware design. To do this:

   a. Select **File > Export > Export Hardware**. The Export Hardware dialog box opens.



   b. Check the **Include bitstream** check box.

   c. Click **OK** to export the hardware design and include the bitstream.

Tcl Projectless Flow

Use the following Tcl commands to perform post-bitstream hardware handoff using Tcl commands outside of the Vivado project.

1. `create_project -in_memory -part xc7z020clg484-1`
2. `set_property board_part xilinx.com:zc702:part0:1.0[current_project]`
3. `read_bd base_zynq_design.bd`
4. `read_vhd base_zynq_design_wrapper.vhd`
5. `generate_target all [get_files base_zynq_design.bd]`
6. `synth_design -top base_zynq_design_wrapper`
7. `opt_design`
8. `place_design`
9. `write_hwdef -file base_zynq_design_wrapper.hwdef`

10. `route_design`
11. `write_bitstream base_zynq_design_wrapper.bit`
12. `write_sysdef -hwdef base_zynq_design_wrapper.hwdef -bitfile base_zynq_design_wrapper.bit -file base_zynq_design_wrapper.hdf`

Tcl Capabilities Overview

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

First Class Tcl Object Types and Relationships

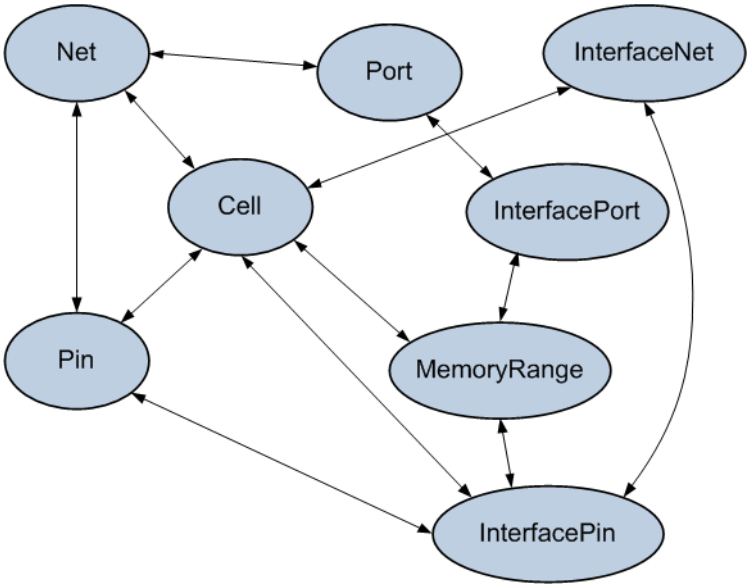
Hardware

- **HardwareDb**: Represents hardware design loaded in memory. SDK, PetaLinux, and third-party tools can have multiple HardwareDb objects.
- **Port**: 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**: 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**: A wire or list of wires that eventually can be physically connected directly together. Nets can be hierarchical or flat, but always sort a list of pins together.
- **InterfaceNet**: A list of wires that eventually can be physically connected directly together. Interface nets can be hierarchical or flat, but always sort a list of interface pins together.

- **Pin**: 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**: 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 the M_AXI_DP interface of MicroBlaze processors, and the S_AXI interface of gpio.
- **Cell**: The instantiation of IP in the hardware design. Examples of cells include instantiation of microblaze, gpio, and axi_dma, as well as hierarchical instances which are wrappers for other groups of cells.
- **MemoryRange**: Represents the memory range associated with the peripherals in the memory map of the processor.
- **SupportingDesignFile**: Represents files associated with the hardware design. For example, .bit or .mmi.

Hardware Object Relationships

Figure 3: **Hardware Object Relationships Diagram**



Software

- **SoftwareDb**: Represents one software design or microprocessor software specification (MSS). SDK, PetaLinux, and other third-party tools can have multiple SoftwareDb objects.
- **SwCore**: Represents driver/library/OS present in the software repositories. For example: cpu, gpio, standalone, xilffs.
- **SwProcessor**: The driver mapped to the processor instance in the hardware design. For example: cpu driver mapped for MicroBlaze processor cell.

- **SwDriver**: The driver mapped to the peripheral instance in the hardware design. For example: gpio driver mapped for axi_gpio cell.
- **SwLibrary**: The library added in the software design. For example: xilflash, xilffs.
- **SwOS**: The OS in the software design. For example: standalone, xilkernel.
- **SwInterface**: The interface of the library/driver. It describes the interface functions and header files used by the library/driver. For example: stdin, stdout of uart driver.
- **SwArray**: The array defined in driver/library/os. It contains any number of PARAMs and PROPERTYs which describe the size and description of the array and default values of elements in the array. It represents software array of the driver/library/os. For example: mem_table, shm_table of xilkernel bsp.
- **SwDTNode**: Represents node in Device Tree (DTS) file.
- **SwParam**: Represents parameters of node.

Tcl Commands Listed by Category

Categories

- DeviceTree
- Hardware
- Object
- Project
- PropertyAndParameter
- Report
- Software
- Tools

DeviceTree

- hsi::create_dt_node
- hsi::create_dt_tree
- hsi::current_dt_tree
- hsi::get_dt_nodes
- hsi::get_dt_trees

Hardware

- hsi::close_hw_design
- hsi::current_hw_design
- hsi::get_cells
- hsi::get_hw_designs
- hsi::get_hw_files
- hsi::get_intf_nets
- hsi::get_intf_pins
- hsi::get_intf_ports
- hsi::get_mem_ranges
- hsi::get_nets
- hsi::get_pins
- hsi::get_ports
- hsi::open_hw_design

Object

- common::get_property
- common::list_property
- common::list_property_value
- common::report_property
- common::reset_property
- common::set_property

Project

- common::help

PropertyAndParameter

- common::create_property
- common::get_param
- common::get_property
- common::list_param
- common::list_property
- common::list_property_value

- common::report_param
- common::report_property
- common::reset_param
- common::reset_property
- common::set_param
- common::set_property

Report

- common::report_environment
- common::report_param
- common::report_property
- common::reset_msg_config
- common::reset_msg_count
- common::set_msg_config
- common::version

Software

- hsi::add_library
- hsi::close_sw_design
- hsi::create_comp_param
- hsi::create_node
- hsi::create_sw_design
- hsi::current_sw_design
- hsi::delete_objs
- hsi::generate_app
- hsi::generate_bsp
- hsi::generate_target
- hsi::get_arrays
- hsi::get_comp_params
- hsi::get_drivers
- hsi::get_fields
- hsi::get_libs

- hsi::get_nodes
- hsi::get_os
- hsi::get_sw_cores
- hsi::get_sw_designs
- hsi::get_sw_interfaces
- hsi::get_sw_processor
- hsi::open_sw_design
- hsi::set_repo_path

Tools

- common::load_features

Tcl Examples

This chapter demonstrates how to load a `.hdf` file, access the hardware information, and generate BSPs, applications, and the Device Tree.

Accessing Hardware Design Data

Opening the hardware design

```
hsi::open_hw_design base_zynq_design_wrapper.hdf
                    base_zynq_design_imp
```

List loaded hardware designs

```
hsi::get_hw_designs
                    base_zynq_design_imp
```

Switch to current hardware design

```
hsi::current_hw_design
                    base_zynq_design_imp
```

Report properties of the current hardware design

```
common::report_property [hsi::current_hw_design]
```

Property	Type	Read-only	Visible	Value
ADDRESS_TAG	string*	true	true	base_zynq_design_i/ ps7_cortexa9_0:base_zynq_design_ibase_zynq_design_i/ ps7_cortexa9_1:base_zynq_design_i
BOARD	string	true	true	xilinx.com:zc702:part0:1.1
CLASS	string	true	true	hw_design

Property	Type	Read-only	Visible	Value
DEVICE	string	true	true	7x020
FAMILY	string	true	true	zynq
NAME	string	true	true	base_zynq_design_imp
PACKAGE	string	true	true	clg484
PATH	string	true	true	/scratch/demo//base_zynq_design.hwh
SPEEDGRADE	string	true	true	???1
SW_REPOSITORIES	string*	true	true	
TIMESTAMP	string	true	true	<current date and time>
VIVADO_VERSION	string	true	true	2014.3

List the hdf files in the container

```
hsi::get_hw_files
                base_zynq_design.hwh ps7_init.c ps7_init.h ps7_init_gpl.c
ps7_init_gpl.h ps7_init.tcl ps7_init.html
                base_zynq_design_wrapper.mmi base_zynq_design_bd.tcl
```

Filter the .bit files

```
hsi::get_hw_files -filter {TYPE==bit}
                base_zynq_design_wrapper.bit
```

List of external ports in the design

```
hsi::get_ports
                DDR_cas_n DDR_cke DDR_ck_n DDR_ck_p DDR_cs_n DDR_reset_n
DDR_odt DDR_ras_n
                DDR_we_n DDR_ba DDR_addr DDR_dm DDR_dq DDR_dqs_n DDR_dqs_p
FIXED_IO_mio
                FIXED_IO_ddr_vrn FIXED_IO_ddr_vrp FIXED_IO_ps_srstb
FIXED_IO_ps_clk
                FIXED_IO_ps_porb leds_4bits_tri_o
```

Reports properties of an external port

```
common::report_property [hsi::get_ports leds_4bits_tri_o]
```

Property	Type	Read-only	Visible	Value
CLASS	string	true	true	port

Property	Type	Read-only	Visible	Value
CLK_FREQ	string	true	true	
DIRECTION	string	true	true	O
INTERFACE	bool	true	true	0
IS_CONNECTED	bool	true	true	0
LEFT	string	true	true	3
NAME	string	true	true	leds_4bits_tri_o
RIGHT	string	true	true	0
SENSITIVITY	enum	true	true	
TYPE	enum	true	true	undef

List of IP instances in the design

```
hsi::get_cells
                axi_bram_ctrl_0 axi_gpio_0 blk_mem_gen_0
processing_system7_0_axi_periph_m00_couplers_auto_pc
                processing_system7_0_axi_periph_s00_couplers_auto_pc
processing_system7_0_axi_periph_xbar
                rst_processing_system7_0_50M ps7_clockc_0 ps7_uart_1
ps7_pl310_0 ps7_pmu_0 ps7_qspi_0
                ps7_qspi_linear_0 ps7_axi_interconnect_0 ps7_cortexa9_0
ps7_cortexa9_1 ps7_ddr_0
                ps7_ethernet_0 ps7_usb_0 ps7_sd_0 ps7_i2c_0 ps7_can_0
ps7_ttc_0 ps7_gpio_0
                ps7_ddrc_0 ps7_dev_cfg_0 ps7_xadc_0 ps7_ocmc_0
ps7_coresight_comp_0 ps7_gpv_0 ps7_scuc_0
                ps7_globaltimer_0 ps7_intc_dist_0 ps7_l2cachec_0 ps7_dma_s
ps7_iop_bus_config_0 ps7_ram_0
                ps7_ram_1 ps7_scugic_0 ps7_scutimer_0 ps7_scuwdt_0
ps7_slcr_0 ps7_dma_ns ps7_afi_0 ps7_afi_1
                ps7_afi_2 ps7_afi_3 ps7_m_axi_gp0
```

#List of processors in the design

```
hsi::get_cells -filter {IP_TYPE==PROCESSOR}
                ps7_cortexa9_0 ps7_cortexa9_1
```

Properties of IP instance

```
common::report_property [hsi::get_cells axi_gpio_0]
```

Property	Type	Read-only	Visible	Value
CLASS	string	true	true	cell

Property	Type	Read-only	Visible	Value
CONFIG.C_ALL_INPUTS	string	true	true	0
CONFIG.C_ALL_INPUTS_2	string	true	true	0
CONFIG.C_ALL_OUTPUTS	string	true	true	1
CONFIG.C_ALL_OUTPUTS_2	string	true	true	0
CONFIG.C_BASEADDR	string	true	true	0x41200000
CONFIG.C_DOUT_DEFAULT	string	true	true	0x00000000
CONFIG.C_DOUT_DEFAULT_2	string	true	true	0x00000000
CONFIG.C_FAMILY	string	true	true	zynq
CONFIG.C_GPIO2_WIDTH	string	true	true	32
CONFIG.C_GPIO_WIDTH	string	true	true	4
CONFIG.C_HIGHADDR	string	true	true	0x4120FFFF
CONFIG.C_INTERRUPT_PRESENT	string	true	true	0
CONFIG.C_IS_DUAL	string	true	true	0
CONFIG.C_S_AXI_ADDR_WIDTH	string	true	true	9
CONFIG.C_S_AXI_DATA_WIDTH	string	true	true	32
CONFIG.C_TRI_DEFAULT	string	true	true	0xFFFFFFFF
CONFIG.C_TRI_DEFAULT_2	string	true	true	0xFFFFFFFF
CONFIG.Component_Name	string	true	true	base_zynq_design_axi_gpio_0_0
CONFIG.EDK_IPTYPE	string	true	true	PERIPHERAL
CONFIG.GPIO2_BOARD_INTERFACE	string	true	true	Custom
CONFIG.GPIO_BOARD_INTERFACE	string	true	true	leds_4bits
CONFIG.USE_BOARD_FLOW	string	true	true	true
CONFIGURABLE	bool	true	true	0
IP_NAME	string	true	true	axi_gpio
IP_TYPE	enum	true	true	PERIPHERAL
NAME	string	true	true	axi_gpio_0
PRODUCT_GUIDE	string	true	true	LogiCORE IP AXI GPIO v2.0 Product Guide
SLAVES	string*	true	true	
VLNV	string	true	true	xilinx.com:ip:axi_gpio:2.0

Memory range of the Slave IPs

```
common::report_property [lindex [hsi::get_mem_ranges -of_objects
                [hsi::get_cells -filter {IP_TYPE==PROCESSOR}]] 39]
```

Property	Type	Read-only	Visible	Value
BASE_NAME	string	true	true	C_BASEADDR

Property	Type	Read-only	Visible	Value
BASE_VALUE	string	true	true	0x41200000
CLASS	string	true	true	mem_range
HIGH_NAME	string	true	true	C_HIGHADDR
HIGH_VALUE	string	true	true	0x4120FFFF
INSTANCE	cell	true	true	axi_gpio_0
IS_DATA	bool	true	true	1
IS_INSTRUCTION	bool	true	true	0
MEM_TYPE	enum	true	true	REGISTER
NAME	string	true	true	axi_gpio_0

Creating Standalone Software Design and Accessing Software Information

List of the drivers in the software repository

```
hsi::get_sw_cores *uart*
                  uartlite_v2_01_a uartlite_v3_0 uartns550_v2_01_a
uartns550_v2_02_a uartns550_v3_0
                  uartns550_v3_1 uartps_v1_04_a uartps_v1_05_a uartps_v2_0
uartps_v2_1 uartps_v2_2
```

Creates software design

```
hsi::create_sw_design swdesign -proc ps7_cortexa9_0 -os standalone
                  swdesign
```

To switch to active software design

```
hsi::current_sw_design
                  swdesign
```

Properties of the current software design

```
common::report_property [hsi::current_sw_design ]
```

Property	Type	Read-only	Visible	Value
APP_COMPILER	string	false	true	arm-xilinx-eabi-gcc
APP_COMPILER_FLAGS	string	false	true	
APP_LINKER_FLAGS	string	false	true	

Property	Type	Read-only	Visible	Value
BSS_MEMORY	string	false	true	
CLASS	string	true	true	sw_design
CODE_MEMORY	string	false	true	
DATA_MEMORY	string	false	true	
NAME	string	true	true	swdesign

The drivers associated to current hardware design

```
hsi::get_drivers
                axi_bram_ctrl_0 axi_gpio_0 ps7_afi_0 ps7_afi_1 ps7_afi_2
ps7_afi_3 ps7_can_0
                ps7_coresight_comp_0 ps7_ddr_0 ps7_ddrc_0 ps7_dev_cfg_0
ps7_dma_ns ps7_dma_s
                ps7_ethernet_0 ps7_globaltimer_0 ps7_gpio_0 ps7_gpv_0
ps7_i2c_0 ps7_intc_dist_0
                ps7_iop_bus_config_0 ps7_l2cachec_0 ps7_ocmc_0 ps7_pl310_0
ps7_pmu_0 ps7_qspi_0
                ps7_qspi_linear_0 ps7_ram_0 ps7_ram_1 ps7_scuc_0
ps7_scugic_0 ps7_scutimer_0
                ps7_scuwdt_0 ps7_sd_0 ps7_slcr_0 ps7_ttc_0 ps7_uart_1
ps7_usb_0 ps7_xadc_0
                hsi% get_osstandalone
```

Properties of the OS object

```
common::report_property[hsi::get_os]
```

Property	Type	Read-only	Visible	Value
CLASS	string	true	true	os
CONFIG.enable_sw_intrusive_profiling	string	false	true	false
CONFIG.microblaze_exceptions	string	false	true	false
CONFIG.predecode_fpu_exceptions	string	false	true	false
CONFIG.profile_timer	string	false	true	none
CONFIG.stdin	string	false	true	ps7_uart_1
CONFIG.stdout	string	false	true	ps7_uart_1
NAME	string	false	true	standalone
VERSION	string	false	true	4.2

Properties of the processor object

```
common::report_property [hsi::get_sw_processor ]
```

Property	Type	Read-only	Visible	Value
CLASS	string	true	true	sw_proc
CONFIG.archiver	string	false	true	arm-xilinx-eabi-ar
CONFIG.compiler	string	false	true	arm-xilinx-eabi-gcc
CONFIG.compiler_flags	string	false	true	-O2 -c
CONFIG.extra_compiler_flags	string	false	true	-g
HW_INSTANCE	string	true	true	ps7_cortexa9_0
NAME	string	false	true	cpu_cortexa9
VERSION	string	false	true	2.1

Generate BSP. BSP source code will be dumped to the output directory.

```
hsi::generate_bsp -dir bsp_out
```

List of available apps in the repository

```
hsi::generate_app -lapp
peripheral_tests dhrystone empty_application hello_world lwip_echo_server
memory_tests rsa_auth_app srec_bootloader xilkernel_thread_demo
zynq_dram_test
zynq_fsbl linux_empty_app linux_hello_world opencv_hello_world
```

Generate template application

```
hsi::generate_app -app hello_world -proc ps7_cortexa9_0 -dir app_out
```

Generate Device Tree. Clone device tree repo from GIT to /device_tree_repository/device-tree-generator-master directory.

load the hardware design

```
hsi::open_hw_design zynq_1_wrapper.hdf
```

Cloned GIT repo path

```
hsi::set_repo_path ./device_tree_repository/device-tree-generator-master
```

create sw design

```
hsi::create_sw_design sw1 -proc ps7_cortexa9_0 -os device_tree
```

generate device tree

```
hsi::generate_target {dts} -dir dtg_out
```

Creating Standalone Software Design and Accessing Software Information

List of the drivers in the software repository

```
hsi::get_sw_cores *uart*
                uartlite_v2_01_a uartlite_v3_0 uartns550_v2_01_a
uartns550_v2_02_a uartns550_v3_0
                uartns550_v3_1 uartps_v1_04_a uartps_v1_05_a uartps_v2_0
uartps_v2_1 uartps_v2_2
```

Creates software design

```
hsi::create_sw_design swdesign -proc ps7_cortexa9_0 -os standalone
                swdesign
```

To switch to active software design

```
hsi::current_sw_design
                swdesign
```

Properties of the current software design

```
common::report_property [hsi::current_sw_design ]
```

Property	Type	Read-only	Visible	Value
APP_COMPILER	string	false	true	arm-xilinx-eabi-gcc
APP_COMPILER_FLAGS	string	false	true	
APP_LINKER_FLAGS	string	false	true	
BSS_MEMORY	string	false	true	
CLASS	string	true	true	sw_design
CODE_MEMORY	string	false	true	
DATA_MEMORY	string	false	true	
NAME	string	true	true	swdesign

The drivers associated to current hardware design

```
hsi::get_drivers
                axi_bram_ctrl_0 axi_gpio_0 ps7_afi_0 ps7_afi_1 ps7_afi_2
ps7_afi_3 ps7_can_0
                ps7_coresight_comp_0 ps7_ddr_0 ps7_ddrc_0 ps7_dev_cfg_0
ps7_dma_ns ps7_dma_s
                ps7_ethernet_0 ps7_globaltimer_0 ps7_gpio_0 ps7_gpv_0
ps7_i2c_0 ps7_intc_dist_0
                ps7_iop_bus_config_0 ps7_l2cachec_0 ps7_ocmc_0 ps7_pl310_0
ps7_pmu_0 ps7_qspi_0
                ps7_qspi_linear_0 ps7_ram_0 ps7_ram_1 ps7_scuc_0
ps7_scugic_0 ps7_scutimer_0
                ps7_scuwdt_0 ps7_sd_0 ps7_slcr_0 ps7_ttc_0 ps7_uart_1
ps7_usb_0 ps7_xadc_0
                hsi% get_osstandalone
```

Properties of the OS object

```
common::report_property[hsi::get_os]
```

Property	Type	Read-only	Visible	Value
CLASS	string	true	true	os
CONFIG.enable_sw_intrusive_profiling	string	false	true	false
CONFIG.microblaze_exceptions	string	false	true	false
CONFIG.predecode_fpu_exceptions	string	false	true	false
CONFIG.profile_timer	string	false	true	none
CONFIG.stdin	string	false	true	ps7_uart_1
CONFIG.stdout	string	false	true	ps7_uart_1
NAME	string	false	true	standalone
VERSION	string	false	true	4.2

Properties of the processor object

```
common::report_property [hsi::get_sw_processor ]
```

Property	Type	Read-only	Visible	Value
CLASS	string	true	true	sw_proc
CONFIG.archiver	string	false	true	arm-xilinx-eabi-ar
CONFIG.compiler	string	false	true	arm-xilinx-eabi-gcc
CONFIG.compiler_flags	string	false	true	-O2 -c
CONFIG.extra_compiler_flags	string	false	true	-g
HW_INSTANCE	string	true	true	ps7_cortexa9_0
NAME	string	false	true	cpu_cortexa9
VERSION	string	false	true	2.1

Generate BSP. BSP source code will be dumped to the output directory.

```
hsi::generate_bsp -dir bsp_out
```

List of available apps in the repository

```
hsi::generate_app -lapp
                  peripheral_tests dhrystone empty_application hello_world
lwip_echo_server
                  memory_tests rsa_auth_app srec_bootloader
xilkernel_thread_demo zynq_dram_test
                  zynq_fsbl linux_empty_app linux_hello_world
opencv_hello_world
```

Generate template application

```
                  hsi::generate_app -app hello_world -proc ps7_cortexa9_0 -
dir app_out
```

Generate Device Tree. Clone device tree repo from GIT to /device_tree_repository/device-tree-generator-master directory.

load the hardware design

```
hsi::open_hw_design zynq_1_wrapper.hdf
```

Cloned GIT repo path

```
hsi::set_repo_path ./device_tree_repository/device-tree-generator-master
```

create sw design

```
hsi::create_sw_design sw1 -proc ps7_cortexa9_0 -os device_tree
```

generate device tree

```
hsi::generate_target {dts} -dir dtg_out
```

Generating and Compiling Application with compiler settings and memory sections of choice

Generating and Compiling Application with compiler settings and memory sections of choice

#Create a software design for the template application with default compiler flags and memory section settings

```
set sw_system_1 [hsi::create_sw_design system_1 -proc microblaze_1 -os
xilkernel -app hello_world]
```

#Change compiler and its flags of the software design

```
common::set_property APP_COMPILER "mb-gcc" $sw_system_1
common::set_property -name APP_COMPILER_FLAGS -value "-DRSA_SUPPORT -
DFSBL_DEBUG_INFO"
-objects $sw_system_1
common::set_property -name APP_LINKER_FLAGS -value "-Wl,--start-group,-
lxil,-lgcc,-lc,--end-group"
-objects $sw_system_1
```

#Change memory sections

```
common::set_property CODE_MEMORY axi_bram_ctrl_1 $sw_system_1
common::set_property BSS_MEMORY axi_bram_ctrl_1 $sw_system_1
common::set_property DATA_MEMORY axi_bram_ctrl_2 $sw_system_1
```

#Genereate application for the above customized software design to Zynq_Fsbl directory

```
hsi::generate_app -dir hw_output -compile
```

Generating and Compiling BSP with advanced driver/library/os/processor configuration

Generating and Compiling BSP with advanced driver/library/os/processor configuration

#Create a software design for the template application with default compiler flags and memory section settings

```
 set sw_system_1 [hsi::create_sw_design system_1 -proc microblaze_1 -os
xilkernel ]
```

#Get the old driver object

```
 set old_driver [hsi::get_drivers myip1]
```

#Set repository path to find the custom drivers and libraries

```
hsi::set_repo_path ./my_local_sw_repository
```

#Set the new driver name and version to old driver object

```
common::set_property NAME myip1_custom_driver $old_driver
common::set_property VERSION 1.0 $old_driver
```

#Change default OS configuration to desired one

```
set OS [hsi::get_os]
common::set_property CONFIG.systmr_dev axi_timer_0 $OS
common::set_property CONFIG.stdin axi_uartlite_0 $OS
common::set_property CONFIG.stdout axi_uartlite_0 $OS
```

#Add custom library to software design

```
hsi::add_library xilflash
```

#Get all the properties of the library, only read_only = false properties can be changed

```
common::report_property [hsi::get_libs xilflash]
```

#Change the default configuration of the library

```
set lib [hsi::get_libs xilflash]
common::set_property CONFIG.enable_amd true $lib
common::set_property CONFIG.enable_intel false $lib
```

#Generate the BSP with the above configuration

```
hsi::generate_bsp -dir advanced_bsp -compile
```

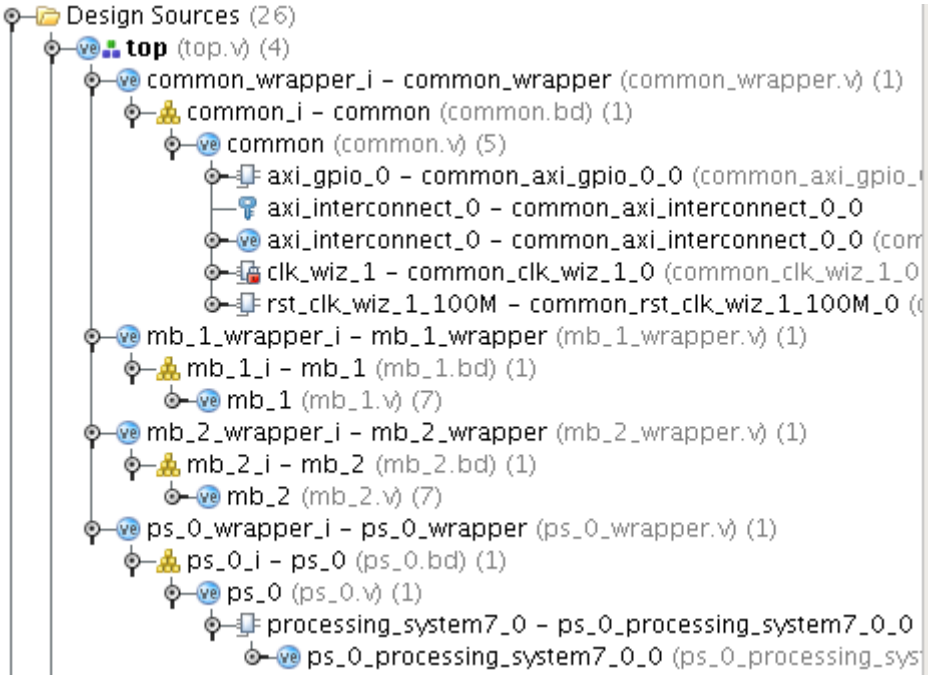
#Delete the library added to software design

```
 hsi::delete_objs $lib
```

Generating and Compiling BSP for a multi-block design

Generating and Compiling BSP for a multi block design

Figure 4: **Example Design with multiple block design instances in the active top design**



#Open hardware design with multiple block design instances

```
hsi% hsi::open_hw_design system_wrapper.hdf
design_1_wrapper
```

#Get the hardware cell instances

#NOTE: cell instances from all the block designs in the top are shown andtheir names are prefixed with their hierarchy

```
hsi% join [get_cells ] \n
ps_0_wrapper_i_ps_0_i_processing_system7_0
ps7_uart_1
ps7_qspi_0
ps7_cortexa9_0
ps7_cortexa9_1
ps7_ddr_0
ps7_ethernet_0
....
mb_2_wrapper_i_mb_2_i_axi_gpio_0
mb_2_wrapper_i_mb_2_i_mdm_1
mb_2_wrapper_i_mb_2_i_microblaze_0
mb_2_wrapper_i_mb_2_i_microblaze_0_axi_periph
mb_2_wrapper_i_mb_2_i_microblaze_0_local_memory_dlmb_bram_if_cntlr
mb_2_wrapper_i_mb_2_i_microblaze_0_local_memory_dlmb_v10
mb_2_wrapper_i_mb_2_i_microblaze_0_local_memory_ilmb_bram_if_cntlr
mb_2_wrapper_i_mb_2_i_microblaze_0_local_memory_ilmb_v10
mb_2_wrapper_i_mb_2_i_microblaze_0_local_memory_lmb_bram
mb_2_wrapper_i_mb_2_i_rst_clk_wiz_1_100M
mb_1_wrapper_i_mb_1_i_axi_gpio_0
mb_1_wrapper_i_mb_1_i_mdm_1
mb_1_wrapper_i_mb_1_i_microblaze_0
mb_1_wrapper_i_mb_1_i_microblaze_0_axi_periph
mb_1_wrapper_i_mb_1_i_microblaze_0_local_memory_dlmb_bram_if_cntlr
 mb_1_wrapper_i_mb_1_i_microblaze_0_local_memory_dlmb_v10
mb_1_wrapper_i_mb_1_i_microblaze_0_local_memory_ilmb_bram_if_cntlr
mb_1_wrapper_i_mb_1_i_microblaze_0_local_memory_ilmb_v10
mb_1_wrapper_i_mb_1_i_microblaze_0_local_memory_lmb_bram
mb_1_wrapper_i_mb_1_i_rst_clk_wiz_1_100M
common_wrapper_i_common_i_axi_gpio_0
common_wrapper_i_common_i_axi_interconnect_0
common_wrapper_i_common_i_clk_wiz_1
common_wrapper_i_common_i_rst_clk_wiz_1_100M
```

#Generate BSP for a processor in bsp_out directory and compile the bsp sources

```
hsi::generate_bsp -proc mb_2_wrapper_i_mb_2_i_microblaze_0 -dir bsp_out -
compile
ls ./bsp_out/mb_2_wrapper_i_mb_2_i_microblaze_0
code
indent
lib
libsrc
```

Input and Output Files

Input Files

HDF

Hardware Design File (.hdf) is a Xilinx proprietary file format and only Xilinx software tools understand it. Third-party software tools can communicate to the HSI Tcl interface to extract data from the .hdf file.

***Note*:** Xilinx does not recommend manually editing the HDF file or altering its contents.

HDF is a container and contains:

- One or more .hwh files
  - Vivado tool version, part, and board tag information
  - IP - instance, name, VLNV, and parameters
  - Memory Map information of the processors
  - Internal Connectivity information (including interrupts, clocks, etc.) and external ports information
- BMM/MMI and BIT files
- User and HLS driver files
- Other meta-data files

Software Repository

Default Repositories

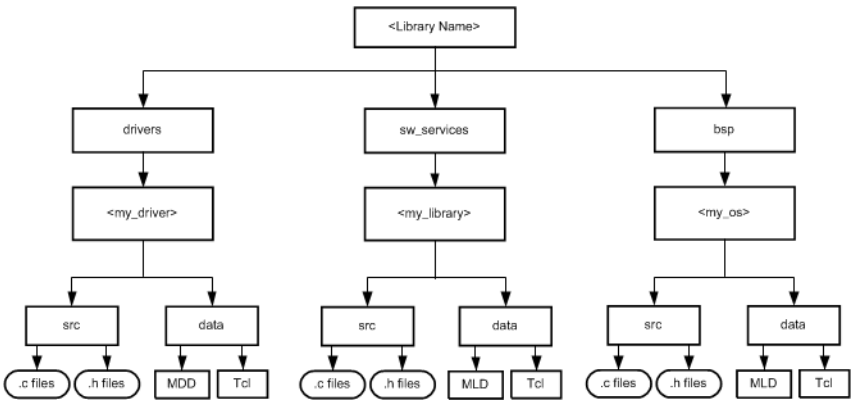
By default, the tool scans the following repositories for software components:

- `<install>/data/embeddedsw/lib/XilinxProcessorIPLib`
- `<install>/data/embeddedsw/lib`

- `<install>/data/embeddedsw/ThirdParty`

The following figure shows the repository directory structure.

Figure 5: **Example Directory Structure**



GIT Repositories

The Device Tree repository can be cloned from Xilinx GIT. Use the `set_repo_path` Tcl command to specify the cloned GIT repository.

User Repositories

You can create drivers, BSPs, and Apps in an example directory structure format, as illustrated in the figure above. Use the `set_repo_path` Tcl command to specify the user repository.

Search Priority Mechanism

The tool uses a search priority mechanism to locate drivers and libraries, as follows:

1. Search the repositories under the library path directory specified using the `set_repo_path` Tcl command.
2. Search the default repositories described above.

Output Files

The tool generates directories, files, and the software design file (MSS) in the `<your_project>` directory. For every processor instance in the MSS file, the tool generates a directory with the name of the processor instance. Within each processor instance directory the tool generates the following directories and files.

- **The include Directory:**

The include directory contains C header files needed by drivers. The include file `xparameters.h` is also created using the tool in this directory. This file defines base addresses of the peripherals in the system, #defines needed by drivers, OSs, libraries, and user programs, as well as function prototypes.

- The Microprocessor Driver Definition (MDD) file for each driver specifies the definitions that must be customized for each peripheral that uses the driver. See Microprocessor Driver Definition (MDD) Overview.
- The Microprocessor Library Definition (MLD) file for each OS and library specifies the definitions that you must customize. See Microprocessor Library Definition (MLD) Overview.

- **The lib Directory:**

  The `lib` directory contains `libc.a`, `libm.a`, and `libxil.a` libraries. The `libxil` library contains driver functions that the particular processor can access. For more information about the libraries, refer to the introductory section of the *OS and Libraries Document Collection (UG643)*.

- **The libsrc Directory:**

  The libsrc directory contains intermediate files and make files needed to compile the OSs, libraries, and drivers. The directory contains peripheral-specific driver files, BSP files for the OS, and library files that are copied from install, as well as your driver, OS, and library directories. Refer to Drivers, OS Block, and Libraries for more information.

- **The code Directory:**

  The code directory is a repository for tool executables. The tool creates an `xmdstub.elf` file (for the MicroBlaze™ processor on-board debug) in this directory.

***Note*:** The tool removes these directories every time you run the it. You must put your sources, executables, and any other files in an area that you create.

Generating Libraries and Drivers

This section provides an overview of generating libraries and drivers. The hardware specification file and the MSS files define a system. For each processor in the system, the tool finds the list of addressable peripherals. For each processor, a unique list of drivers and libraries are built. The tool does the following for each processor:

- Builds the directory structure, as defined in Output Files.
- Copies the necessary source files for the drivers, OSs, and libraries into the processor instance specific area: `OUTPUT_DIR/processor_instance_name/libsrc`.

- Calls the Design Rule Check (DRC) procedure, which is defined as an option in the MDD or MLD file, for each of the drivers, OSs, and libraries visible to the processor.
- Calls the `generate` Tcl procedure (if defined in the Tcl file associated with an MDD or MLD file) for each of the drivers, OSs, and libraries visible to the processor. This generates the necessary configuration files for each of the drivers, OSs, and libraries in the include directory of the processor.
- Calls the `post_generate` Tcl procedure (if defined in the Tcl file associated with an MDD or MLD file) for each of the drivers, OSs, and libraries visible to the processor.
- Runs `make` (with targets include and libs) for the OSs, drivers, and libraries specific to the processor. On the Linux platform, the `gmake` utility is used, while on NT platforms, `make` is used for compilation.
- Calls the `execs_generate` Tcl procedure (if defined in the Tcl file associated with an MDD or MLD file) for each of the drivers, OSs, and libraries visible to the processor.

MDD, MLD, and Tcl

A driver or library has two associated data files:

- **Data Definition File (MDD or MLD file)**: This file defines the configurable parameters for the driver, OS, or library.
- **Data Generation File (Tcl)**: This file uses the parameters configured in the MSS file for a driver, OS, or library to generate data. Data generated includes but is not limited to generation of header files, C files, running DRCs for the driver, OS, or library, and generating executables.

  The Tcl file includes procedures that tool calls at various stages of its execution. Various procedures in a Tcl file include:

  . **DRC**: The name of DRC given in the MDD or MLD file.

  . **generate**: A tool-defined procedure that is called after files are copied.

  . **post_generate**: A tool-defined procedure that is called after generate has been called on all drivers, OSs, and libraries.

  . **execs_generate**: A tool-defined procedure that is called after the BSPs, libraries, and drivers have been generated.

***Note*:** The data generation (Tcl) file is not necessary for a driver, OS, or library.

For more information about the Tcl procedures and MDD/MLD related parameters, refer to Microprocessor Driver Definition (MDD) Overview and Microprocessor Library Definition (MLD) Overview.

MSS Parameters

For a complete description of the MSS format and all the parameters that MSS supports, refer to MSS Overview

Drivers

Most peripherals require software drivers. The peripherals are shipped with associated drivers, libraries and BSPs. Refer to the *Device Driver Programmer Guide* for more information on driver functions. This guide is located in your SDK installation, in `< <install_directory>\SDK \<version> \data\embeddedsw\doc\xilinx_drivers_guide.pdf`.

The MSS file includes a driver block for each peripheral instance. The block contains a reference to the driver by name (DRIVER_NAME parameter) and the driver version (DRIVER_VER). There is no default value for these parameters.

A driver has an associated MDD file and a Tcl file.

- The driver MDD file is the data definition file and specifies all configurable parameters for the drivers.
- Each MDD file has a corresponding Tcl file which generates data that includes generation of header files, generation of C files, running DRCs for the driver, and generating executables.

You can write your own drivers. These drivers must be in a specific directory under `/` or `/drivers`, as shown in the figure in Software Repository.

- The DRIVER_NAME attribute allows you to specify any name for your drivers, which is also the name of the driver directory.
- The source files and make file for the driver must be in the `/src` subdirectory under the `/` directory.
- The make file must have the targets `/include` and `/libs`.
- Each driver must also contain an MDD file and a Tcl file in the `/data` subdirectory.

Open the existing driver files to get an understanding of the required structure.

Refer to Microprocessor Driver Definition (MDD) Overview for details on how to write an MDD and its corresponding Tcl file.

Libraries

The MSS file includes a library block for each library. The library block contains a reference to the library name (LIBRARY_NAME parameter) and the library version (LIBRARY_VER). There is no default value for these parameters. Each library is associated with a processor instance specified using the PROCESSOR_INSTANCE parameter. The library directory contains C source and header files and a make file for the library.

The MLD file for each library specifies all configurable options for the libraries and each MLD file has a corresponding Tcl file.

You can write your own libraries. These libraries must be in a specific directory under `/sw_services` as shown in the figure in Software Repository.

- The LIBRARY_NAME attribute lets you specify any name for your libraries, which is also the name of the library directory.
- The source files and make file for the library must be in the `/src` subdirectory under the `/` directory.
- The make file must have the targets `/include` and `/libs`.
- Each library must also contain an MLD file and a Tcl file in the `/data` subdirectory.

Refer to the existing libraries for more information about the structure of the libraries.

Refer to Microprocessor Library Definition (MLD) Overview for details on how to write an MLD and its corresponding Tcl file.

OS Block

The MSS file includes an OS block for each processor instance. The OS block contains a reference to the OS name (OS_NAME parameter), and the OS version (OS_VER). There is no default value for these parameters. The bsp directory contains C source and header files and a make file for the OS.

The MLD file for each OS specifies all configurable options for the OS. Each MLD file has a corresponding Tcl file associated with it. Refer to Microprocessor Library Definition (MLD) Overview and MSS Overview.

You can write your own OSs. These OSs must be in a specific directory under /bsp or /bsp as shown in the figure in Software Repository.

- The OS_NAME attribute allows you to specify any name for your OS, which is also the name of the OS directory.
- The source files and make file for the OS must be in the src subdirectory under the `/` directory.
- The make file should have the targets `/include` and `/libs`.
- Each OS must contain an MLD file and a Tcl file in the `/data` subdirectory.

Look at the existing OSs to understand the structures. See Microprocessor Library Definition (MLD) Overview for details on how to write an MLD and its corresponding Tcl file, refer to the *Device Driver Programmer Guide*. This guide is located in your SDK installation, in `<install_directory> \SDK\<version>\data\embeddedsw\doc \xilinx_drivers_guide.pdf`.

Appendix A

Obsolete Commands

The table below shows the equivalent HSI Tcl commands that are equivalent to obsolete Libgen commands.

Libgen TCL commands	HSI TCL commands
`xget_hw_busif_handle <handle> <busif_name>`	`hsi::get_bus_intfs <busif_name> -of_objects <handle>`
`xget_hw_busif_value <handle> <busif_name>`	`common::get_property BUS_NAME [hsi::get_bus_intfs <busif_name> -of_objects <handle>]`
`xget_hw_ipinst_handle <handle> <ipinst_name>`	`hsi::get_cells <ipinst_name> -of_objects <handle>`
`xget_hw_name <handle>`	`common::get_property NAME <handle>`
`xget_hw_value <inhandle>`	`set class [common::get_property class $inhandle] if { $class == "hsm_cell" } { return [common::get_property ip_name $inhandle] } elseif { $class == "hsm_port" } { return [common::get_property net_name $inhandle] } elseif { $class == "hsm_bus_intf" } { return [common::get_property bus_name $inhandle] } else { #throw error, parameter and others are not handled return "" }`
`xget_hw_proc_slave_periphs <merged_proc_handle>`	`common::get_property slaves <proc_handle>`
`xget_hw_port_handle <handle> <port_name>`	`hsi::get_ports <port_name> -of_objects <port_handle>`
`xget_hw_port_value <handle> <port_name>`	`common::get_property SIG_NAME [hsi::get_ports <port_name> -of_objects <port_handle>]`
`xget_hw_connected_busifs_handle <merged_mhs_handle> <businst_name> <busif_type>`	`hsi::get_bus_intfs -conn_name <businst_name> -of_objects <hw_db_handle> -filter "TYPE == <busif_type>"`

Libgen TCL commands	HSI TCL commands
`xget_hw_connected_ports_handle <merged_mhs_handle> <connector_name> <port_type>`	`hsi::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 HSI.
`xget_hw_parameter_value <handle> <parameter_name>`	`common::get_property CONFIG.<parameter_name> <cell_handle>`
`xget_hw_bus_slave_addrpairs <merged_bus_handle>`	There is no equivalent command in HSI, because there is no way to get address range values from BUSIF handle.
`xget_hw_subproperty_value <property_handle> <subprop_name>`	`common::get_property <subprop_name> <property_handle>`
`xget_hwhandle <ip_name>`	`hsi::get_cells <ip_name>`

Deprecated Commands

The table below shows the old Tcl commands that are deprecated, and the new Tcl commands to replace them.

Category	Old Tcl Proc	New Tcl Proc	Description
Hardware	`xget_connected_intf`	`::hsi::utils::get_connected_intf <periph_name> <intf_name>`	Returns the connected interface.
Hardware	`xget_hw_port_value`	`::hsi::utils::get_net_name <periph_name> <pin_name>`	Returns the connected net name to an IP pin.
Hardware	`xget_hw_busif_value`	`::hsi::utils::get_intfnet_name <periph_name> <intf_name>`	Returns the connected interface name.
Hardware	`xget_hw_proc_slave_periphs`	`::hsi::utils::get_proc_slave_periphs <processor_cell_object>`	Returns all the peripheral objects which are connected to the processor.
Hardware	`xget_ip_clk_pin_freq`	`::hsi::utils::get_clk_pin_freq <periph_object> <pin_name>`	Returns the clock frequency value of the IP clock pin.
Hardware	`is_external_pin`	`::hsi::utils::is_external_pin <pin_object>`	Returns `true` if pin is connected to external port. Otherwise it will return `false`.
Hardware	`xget_port_width`	`::hsi::utils::get_port_width <port_object>`	Returns the width of `port`.
Hardware	`xget_interrupt_sources`	`::hsi::utils::get_interrupt_sources <periph_object>`	Returns the handles for all ports driving the interrupt pin of a peripheral.
Hardware	`xget_source_pins`	`::hsi::utils::get_source_pins <pin_object>`	Returns the source pins of a peripheral pin object.
Hardware	`xget_sink_pins`	`::hsi::utils::get_sink_pins <pin_object>`	Returns the sink pins of a peripheral pin object.
Hardware	`xget_connected_pin_count`	`::hsi::utils::get_connected_pin_count <pin_object>`	Returns the count of pins that are connected to peripheral pin.
Hardware	`xget_param_value`	`::hsi::utils::get_param_value <periph_object> <param_name>`	Returns the parameter value of a peripheral.
Hardware	`xget_p2p_name`	`::hsi::utils::get_p2p_name <periph_object> <arg>`	Returns the name of the point2point (p2p) peripheral if `arg` is present.

Category	Old Tcl Proc	New Tcl Proc	Description
Hardware	xget_procs	::hsi::utils::get_procs	Returns all the processor instance object in the design.
Hardware	xget_port_interrupt_id	::hsi::utils::get_port_intr_id <periph_obj> <interrupt_port_name>	Returns the interrupt ID of a peripheral interrupt port.
Hardware	is_interrupt_controller	::hsi::utils::is_intr_cntrl <periph_name>	Returns true if peripheral is interrupt controller.
Hardware	get_connected_interrupt_controller	::hsi::utils::get_connected_intr_cntrl <periph_name> <pin_name>	Returns the connected interrupt controller.
Hardware	get_ip_sub_type	::hsi::utils::get_ip_sub_type <periph_object>	Returns the IP subtype. (EDK_SPECIAL)
Common	xget_swverandbld	::hsi::utils::get_sw_build_version	Returns the software version.
Common	xget_copyrightstr	::hsi::utils::get_copyright_msg	Returns the copyright message along with software version.
Common	xprint_generated_header	::hsi::utils::write_c_header <file_handle> <description>	Writes the standard Xilinx Header for .h/.c files.
Common	xprint_generated_header_tcl	::hsi::utils::write_tcl_header <file_handle> <description>	Writes the standard Xilinx Header for Tcl files.
Common	xformat_addr_string	::hsi::utils::format_addr_string <value> <param_name>	get the special format for the address parameters that have special string in parameter value.
Common	xformat_address_string	::hsi::utils::format_address_string <value>	get the hex format string of input value.
Common	xconvert_binary_to_hex	::hsi::utils::convert_binary_to_hex <value>	Converts a binary number to a hex value.
Common	xconvert_binary_to_decimal	::hsi::utils::convert_binary_to_decimal <value>	Converts a binary number to decimal value.
Common	xconvert_num_to_binary	::hsi::utils::convert_num_to_binary <value> <length>	Converts a number (hex or decimal formt) to binary.
Common	compare_unsigned_addr_strings	::hsi::utils::compare_unsigned_addresses <base_addr> <base_param> <high_addr> <high_param>	return 1 if $base_addr > $high_addr.
			return 0 if $base_addr == $high_addr.
			return -1 if $base_addr < $high_addr.

Category	Old Tcl Proc	New Tcl Proc	Description
Common	compare_unsigned_int_values	::hsi::utils::compare_unsigned_int_values <int_base> <int_high>	return 1 if $int_base > $int_high.
			return 0 if $int_base == $int_high.
			return -1 if $int_base < $int_high.
Common	xformat_tohex	::hsi::utils::format_to_hex <value> <bitwidth> <direction>	Writes the given hex number in the format specified by bitwidth. Padding or truncating bits as necessary in direction specified by direction.
Common	xformat_tobin	::hsi::utils::format_to_bin <value> <bitwidth> <direction>	Writes the given binary number in the format specified by bitwidth, padding or truncating bits as necessary in direction specified by direction.
Common	xget_nameofexecutable	::hsi::utils::get_nameofexecutable	Returns the executable name.
Common	xget_hostos_platform	::hsi::utils::get_hostos_platform	Returns the host os platform. Possible return values are lnx, lnx64, win, win64
Common	xget_hostos_exec_suffix	::hsi::utils::get_hostos_exec_suffix	Returns the executable suffix .exe for windows; empty for linux.
Common	xget_hostos_sharedlib_suffix	::hsi::utils::get_hostos_sharedlib_suffix	Returns the shared library suffix.
Common	xfind_file_in_dirs	::hsi::utils::find_file_in_dirs <dir_list> <related_filepath>	Finds a file within a list of given directory.
Common	xfind_file_in_xilinx_install	::hsi::utils::find_file_in_xilinx_install <relative_filepath>	Finds a specific file within the Xilinx tool install.
Common	xload_xilinx_library	::hsi::utils::load_xilinx_library <libname>	Dynamically loads a DLL into TCL interpreter. This procedure searches for DLLs in $MYXILINX and $XILINX.
Software	xopen_include_file	::hsi::utils::open_include_file <file_name>	Opens file in the include directory.
Software	xget_name	::hsi::utils::get_ip_param_name <periph_name> <param>	Creates a parameter name based on the format of Xilinx device drivers. Uses peripheral name to form the parameter name.

Category	Old Tcl Proc	New Tcl Proc	Description
Software	xget_dname	::hsi::utils::get_driver_param_name <driver_name> <param>	Creates a parameter name based on the format of Xilinx Device Drivers. Uses driver name to form the parameter name.
Software	xdefine_include_file	::hsi::utils::define_include_file <driver_handler> <file_name> <drv_string> <args>	Given a list of arguments, define them all in an include file.
Software	xdefine_zynq_include_file	::hsi::utils::define_zynq_include_file <driver_handler> <file_name> <drv_string> <args>	Given a list of arguments, define them all in an include file.
Software	xdefine_if_all	::hsi::utils::define_if_all <driver_handler> <file_name> <driver_string> <args>	Given a list of arguments, define parameter only if all peripherals have this parameter defined.
Software	xdefine_max	::hsi::utils::define_max <driver_handle>r <file_name> <define_name> <arg>	Define parameter as the maximum value for all connected peripherals.
Software	xdefine_config_file	::hsi::utils::define_config_file <driver_handle> <file_name> <driver_string> <args>	Creates Configuration C file as required by Xilinx Drivers.
Software	xdefine_zynq_config_file	::hsi::utils::define_zynq_config_file <driver_handle> <file_name> <driver_string> <args>	Creates Configuration C file as required by Xilinx Zynq®-7000 SoC Drivers.
Software	xdefine_with_names	::hsi::utils::define_with_names <driver_handle> <periph>_handlr <file_name> <args>	Add definitions in an include file. Args must be name value pairs.
Software	xdefine_include_file_membank	::hsi::utils::define_include_file_membank <drv_handle> <file_name> <args>	Given a list of memory bank arguments, define them all in an include file. The args is a base, high address pairs of the memory banks.
Software	xdefine_membank	::hsi::utils::define_membank <periph>_object <file_name> <args>	Generates the definition for a memory bank.
Software	xfind_addr_params	::hsi::utils::find_addr_params <periph>	Find all possible address params for the given peripheral periph.
Software	xdefine_addr_params	::hsi::utils::define_addr_params <drv_handle> <file_name>	Defines all possible address params in the filename for all periphs that use this driver.
Software	xdefine_all_params	::hsi::utils::define_all_params <drv_handle> <file_name>	Defines all params in the filename for all periphs that use this driver.

Category	Old Tcl Proc	New Tcl Proc	Description
Software	xdefine_canonical_xpars	::hsi::utils::define_canonical_xpars <drv_handle> <file_name> <driver_string> <args>	Defines canonical for a driver. Given a list of arguments, define each as a canonical constant name, using the driver name, in an include file.
Software	xdefine_zynq_canonical_xpars	::hsi::utils::define_zynq_canonical_xpars <drv_handle> <file_name> <driver_string> <args>	Defines canonical for a driver. Given a list of arguments, define each as a canonical constant name, using the driver name, in an include file.
Software	xdefine_processor_params	::hsi::utils::define_processor_params <drv_handle> <file_name>	Define processor params using IP Type.
Software	xget_ip_mem_ranges	::hsi::utils::get_ip_mem_ranges <periph>	Get the memory ranges of IP for current processor.
Software	handle_stdin	::hsi::utils::handle_stdin <drv_handle>	Handle the stdin parameter of a processor.
Software	handle_stdout	::hsi::utils::handle_stdout <drv_handle>	Handle the stdout parameter of a processor.
Software	xget_sw_iplist_for_driver	::hsi::utils::get_common_driver_ips <drv_handle>	Returns list of IP cell objects which have a common driver.
Software	is_interrupting_current_processor	::hsi::utils::is_pin_interrupting_current_proc <periph_name> <intr_pin>	Returns true if it is interrupting the current proecessor.
Software	get_current_processor_interrupt_controller	::hsi::utils::get_current_proc_intr_cntrl	Returns the interrupt controller that belongs to current processor of the sw_design.
Software	is_ip_interrupting_current_processor	::hsi::utils::is_ip_interrupting_current_proc <periph_name>	Returns true if at least one interrupt port of IP is interrupting the current processor

Appendix C

BSP, DTS, and Application Generation in Vivado

This chapter demonstrates how to load a .hdf file, access the hardware and software information, and generate BSPs, Applications, and the Device Tree from within Vivado.

1. Run the Vivado hardware handoff flow either in Pre-Synth or Post-Bitstream mode. See the Hardware Handoff section for more information on hardware handoff .
2. Load the hsi feature in Vivado to access all hsi and its util commands as shown below. The hsi:: namespace should be used to access hsi commands.

Load hsi feature in vivado. After loading the hsi feature, all the hsi commands can be accessed through hsi namespace:

```
Vivado% common::load_feature hsi
```

Set the software repository path so that driver/bsp/library/applications are available to hsi. Refer to Software Repository section for more information on default and git repositories. If software repository path is not set then only hardware information can be accessed.

```
Vivado% hsi::set_repo_path <software repository path>
```

Open the hardware design

```
Vivado% hsi::open_hw_design base_zynq_design_wrapper.hdf
```

Get the processor instances in the design

```
Vivado% hsi::get_cells -filter {IP_TYPE==PROCESSOR}
ps7_cortexa9_0 ps7_cortexa9_1
```

Get the top level ports in the hardware design

```
Vivado% hsi::get_ports
DDR_cas_n DDR_cke DDR_ck_n DDR_ck_p DDR_cs_n DDR_reset_n DDR_odt DDR_ras_n
DDR_we_n
DDR_ba DDR_addr DDR_dm DDR_dq DDR_dqs_n DDR_dqs_p FIXED_IO_mio
FIXED_IO_ddr_vrn
FIXED_IO_ddr_vrp FIXED_IO_ps_srstb FIXED_IO_ps_clk FIXED_IO_ps_porb
leds_4bits_tri_o
```

Get the list of BSPs available in software repository specified above

```
Vivado% hsi::get_sw_cores -filter {TYPE==OS}
freertos820_xilinx_v1_0 standalone_v3_10_a standalone_v3_11_a
standalone_v3_12_a
standalone_v4_0 xilkernel_v5_01_a xilkernel_v5_02_a xilkernel_v6_0
standalone_v4_1
xilkernel_v6_1 xilkernel_v6_2 standalone_v5_0 standalone_v4_2 device-tree
```

Create a software design

```
Vivado% hsi::create_sw_design swdesign -proc ps7_cortexa9_0 -os
standalone
swdesign
```

Generate BSP. BSP source code will be dumped to the bsp_out directory

```
Vivado% hsi::generate_bsp -dir bsp_out
hsi::generate_bsp: Time (s): cpu = 00:00:00.29 ; elapsed = 00:00:27 .
Memory (MB): peak = 975.637 ;
gain = 0.000 ; free physical = 14087 ; free virtual = 135939
Vivado% ls ./bsp_out/ Makefile ps7_cortexa9_0 swdesign.mss

Vivado% ls ./bsp_out/ps7_cortexa9_0/
 code include lib libsrc
```

List of available apps in the repository

```
Vivado% hsi::generate_app -lapp
peripheral_tests dhrystone empty_application hello_world lwip_echo_server
memory_tests srec_bootloader …
```

#Generate a template application. Template application will be dumped to app_out directory

```
Vivado% hsi::generate_app -app peripheral_tests -proc ps7_cortexa9_0 -os
standalone -dir app_out
hsi::generate_app: Time (s): cpu = 00:00:00.53 ; elapsed = 00:01:16 .
Memory (MB): peak = 975.637 ;
gain = 0.000 ; free physical = 14846 ; free virtual = 136743>
Vivado% ls ./app_out/
canps_header.h scugic_header.h xemacps_example_intr_dma.c
devcfg_header.h scutimer_header.h xemacps_example_util.c
```

```
dmaps_header.h scuwdt_header.h xgpio_tapp_example.c
emacps_header.h testperiph.c xiicps_selftest_example.c
gpio_header.h ttcps_header.h xqspips_selftest_example.c
iicps_header.h xcanps_intr_example.c xscugic_tapp_example.c
lscript.ld xcanps_polled_example.c xscutimer_intr_example.c
Makefile xdevcfg_selftest_example.c xscutimer_polled_example.c
peripheral_tests_bsp xdmaps_example_w_intr.c xscuwdt_intr_example.c
qspips_header.h xemacps_example.h xttcps_tapp_example.c
```

#Generate Processing System core initialization files. Generated ps_init* are dumped to psinit_out directory

```
Vivado% hsi::generate_target {psinit} [hsi::get_cells -filter
{CONFIGURABLE==1}] -dir ./psinit_out
hsi::generate_target: Time (s): cpu = 00:00:08 ; elapsed = 00:00:10 .
Memory (MB): peak = 997.637 ;
gain = 22.000 ; free physical = 14785 ; free virtual = 136900
Vivado% ls ./psinit_out/
ps7_init.c ps7_init_gpl.h ps7_init.html ps7_parameters.xml
ps7_init_gpl.c ps7_init.h ps7_init.tcl
# Generate Device Tree. Clone device tree repo from GIT
# to ./device_tree_repository/device-tree-generator-master directory.
# Set cloned GIT repo path
Vivado% hsi::set_repo_path ./device_tree_repository/device-tree-generator-
master
# create sw design for device tree
Vivado% set proc_name [common::get_property NAME [hsi::get_cells
*ps7_cortexa9_0*
-filter {IP_TYPE == PROCESSOR}]]
Vivado% hsi::create_sw_design sw_dsgn_device_tree -proc $proc_name -os
device_tree
sw_dsgn_device_tree
# generate device tree. Device tree files are dumped to dtg_out directory
Vivado% hsi::generate_target -dir ./device_tree_out
hsi::generate_target: Time (s): cpu = 00:00:04 ; elapsed = 00:00:05 .
Memory (MB): peak = 997.637 ;
gain = 0.000 ; free physical = 14731 ; free virtual = 136955
Vivado% ls ./device_tree_out/
pl.dtsi skeleton.dtsi sw_dsgn_device_tree.mss system.dts zynq-7000.dtsi
```

XILINX
ALL PROGRAMMABLE™

Appendix D

Microprocessor Software Specification (MSS)

MSS 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 filesys 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 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

OS, Driver, Library, and Processor Block Parameters

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

- PROC_INSTANCE
- HW_INSTANCE
- OS_NAME
- OS_VER
- DRIVER_NAME
- DRIVER_VER
- LIBRARY_NAME
- LIBRARY_VER

PROC_INSTANCE

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

```
parameter PROC_INSTANCE = <instance_name>
```

All operating systems require processor instances to be associated with them. The instance name that is given must match the name specified in the hardware database.

HW_INSTANCE

This option is required for drivers associated with peripheral instances specified in the hardware database 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 as `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 Mircoprocessor 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 as `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 Mircoprocessor 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 as `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. Refer to Microprocessor Library Defintion (MLD) for more information.

MLD/MDD 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 Mircoprocessor Library Definition (MLD) and Mircoprocessor Driver Definition (MDD) for more information.

OS—Specific Specific Parameters

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 Specific Parameters

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

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 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:

```

```

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
parameter COMPILER = dcc
```

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)

Microprocessor Library Definition (MLD) Overview

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

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 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.

MLD 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 Descriptions.

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, header files, C files, DRCs for the OS or library, and 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 the following:

- `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
- `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 configurable parameters in an OS/library. The format used to describe this section is discussed in MLD Parameter Descriptions.

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.

MLD Design Rule Check 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 hardware (HDF) and software (MSS) database 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.

MLD Format Examples

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

Example: MLD File for a Library

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 DESC = "Xilinx Memory File System" ;
OPTION drc = mfs_drc ;
option copyfiles = all;
OPTION REQUIRES_OS = (standalone xilkernel freertos_zynq);
OPTION VERSION = 2.0;
OPTION NAME = xilmfs;
```

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

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]
   ::hsi::utils::write_c_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 [common::get_property CONFIG.need_utils $lib_handle]
    if {$need_utils} {
        # tell libgen or xps that the hardware platform needs to provide
stdio functions
        # inbyte and outbyte to support utils
```

```
    puts $conffile "#include <stdio.h>"
     }
     puts $conffile "#include <xilmfs.h>"
     set value  [common::get_property CONFIG.numbytes $lib_handle]
     puts  $conffile "#define MFS_NUMBYTES  $value"
     set value  [common::get_property CONFIG.base_address $lib_handle]
     puts  $conffile "#define MFS_BASE_ADDRESS $value"
     set value  [common::get_property CONFIG.init_type $lib_handle]
     puts  $conffile "#define MFS_INIT_TYPE  $value"
     puts $conffile "#endif"
     close $conffile
}
```

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 Descriptions

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 Keywords

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

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 preceeding 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.

OS_TYPE

Specifies the type of OS. This value is matched with SUPPORTED_OS_TYPES of the driver MDD file for assigning the driver. Default is standalone.

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 the 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` 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.

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
```

MLD Design Rule Check 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 hardware (HDF) and software (MSS) database 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.

MLD 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.

XILINX
ALL PROGRAMMABLE™

Appendix F

Microprocessor Driver Definition (MDD)

Microprocessor Driver Definition (MDD) Overview

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

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 Tool Command Language (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, see 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.

MDD 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.

- **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 MSS file for the driver to generate data. Data generated includes but is 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.

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.

MDD Format Examples

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

Example: MDD File

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.

```
  OPTION supported_peripherals = (mdm axi_uartlite);
  OPTION driver_state = ACTIVE;
  OPTION copyfiles = all;
  OPTION VERSION = 3.0;
  OPTION NAME = uartlite;
```

The NAME option indicates the name of the driver. The VERSION option indicates the version of the driver. The COPYFILES option indicates the files to be copied for a "level" 0 uartlite driver.

```
    BEGIN INTERFACE stdin
```

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

```
      PROPERTY header = xuartlite_l.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_l.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_l.h.

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

```
    BEGIN INTERFACE stdio
      PROPERTY header = xuartlite_l.h;
      FUNCTION name = inbyte, value = XUartLite_RecvByte;
      FUNCTION name = outbyte, value = XUartLite_SendByte;
    END INTERFACE
```

`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 generate {drv_handle} {
    ::hsi::utils::define_include_file $drv_handle "xparameters.h"
"XUartLite" "NUM_INSTANCES" "C_BASEADDR"
"C_HIGHADDR" "DEVICE_ID" "C_BAUDRATE" "C_USE_PARITY" "C_ODD_PARITY"
"C_DATA_BITS"
    ::hsi::utils::define_config_file $drv_handle "xuartlite_g.c"
"XUartLite"  "DEVICE_ID" "C_BASEADDR"
"C_BAUDRATE" "C_USE_PARITY" "C_ODD_PARITY"  "C_DATA_BITS"

    ::hsi::utils::define_canonical_xpars $drv_handle "xparameters.h"
"UartLite" "DEVICE_ID" "C_BASEADDR"
"C_HIGHADDR" "C_BAUDRATE" "C_USE_PARITY" "C_ODD_PARITY" "C_DATA_BITS"
}
```

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 Keywords

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

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 that 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`.

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_uartlte_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 described below.

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. For 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 for more details about 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 can not 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.

MDD 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 hardware (HDF) and software (MSS) database 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 then 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 just return without any value.

MDD 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.

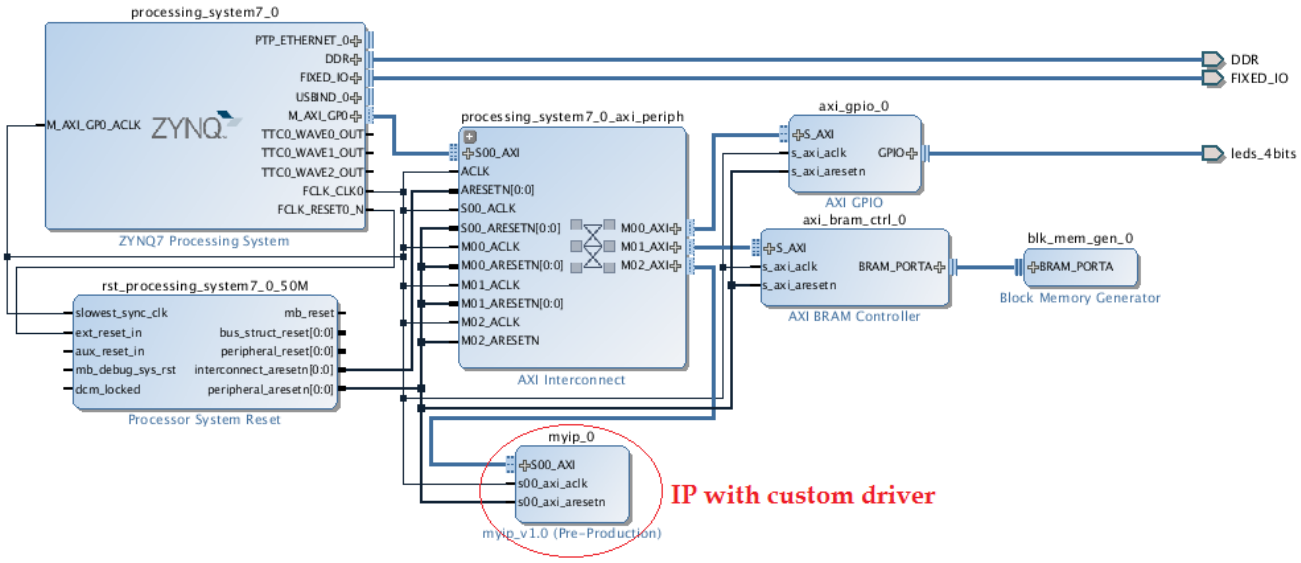
The `handle` is a handle to the current driver in the database.

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.

Custom Driver

This section demonstrates how to handoff a custom driver associated with an IP(driver files are specified in IPXACT file of the IP component) and access the driver information in HSI as well as associate the driver with IP during BSP generation. For more information on packaging IP with custom driver, refer to *Vivado Design Suite User Guide: Creating and Packaging Custom IP* (UG1118).

Figure 6: **Example Design with an IP with custom driver**



The figure above shows an example design of an IP with custom driver specified in its IPXACT definition.

Figure 7: **Custom driver specified in IPXACT specification of an IP**

```
<spirit:fileSet>
  <spirit:name>xilinx_softwaredriver_view_fileset</spirit:name>
  <spirit:file>
    <spirit:name>drivers/myip_v1_0/data/myip.mdd</spirit:name>
    <spirit:userFileType>mdd</spirit:userFileType>
    <spirit:userFileType>driver_mdd</spirit:userFileType>
  </spirit:file>
  <spirit:file>
    <spirit:name>drivers/myip_v1_0/data/myip.tcl</spirit:name>
    <spirit:fileType>tclSource</spirit:fileType>
    <spirit:userFileType>driver_tcl</spirit:userFileType>
  </spirit:file>
  <spirit:file>
    <spirit:name>drivers/myip_v1_0/src/Makefile</spirit:name>
    <spirit:userFileType>unknown</spirit:userFileType>
    <spirit:userFileType>driver_src</spirit:userFileType>
  </spirit:file>
  <spirit:file>
    <spirit:name>drivers/myip_v1_0/src/myip.h</spirit:name>
    <spirit:fileType>cSource</spirit:fileType>
    <spirit:userFileType>driver_src</spirit:userFileType>
  </spirit:file>
  <spirit:file>
    <spirit:name>drivers/myip_v1_0/src/myip.c</spirit:name>
    <spirit:fileType>cSource</spirit:fileType>
    <spirit:userFileType>driver_src</spirit:userFileType>
  </spirit:file>
  <spirit:file>
    <spirit:name>drivers/myip_v1_0/src/myip_selftest.c</spirit:name>
    <spirit:fileType>cSource</spirit:fileType>
    <spirit:userFileType>driver_src</spirit:userFileType>
  </spirit:file>
</spirit:fileSet>
```

Run Vivado hardware handoff flow either in Pre-Synth or Post-Bitstream mode. Refer to Hardware Handoff section for more information. Custom driver for each IP is packaged in HDF.

Open the hardware design with custom drivers.

```
hsi::open_hw_design ./base_zynq_design_wrapper.hdf
base_zynq_design_wrapper
```

Create a software design

```
hsi::create_sw_design swdesign -proc ps7_cortexa9_0 -os standalone
Swdesign
```

Check if the custom drivers are assign to respective IP cores or not

```
join [hsi::get_drivers ] \n
axi_bram_ctrl_0
axi_gpio_0
myip_0
```

Check the custom driver properties

```
common::report_property [ hsi::get_drivers myip*]
```

Property	Type	Read-only	Visible	Value
CLASS	string	true	true	driver
HW_INSTANCE	string	true	true	myip_0
NAME	string	false	true	myip
VERSION	string	false	true	1.0

Generate BSP. BSP source code including custom driver sources will be dumped to the bsp_out #directory

```
hsi::generate_bsp -dir bsp_out
base_zynq_design_wrapper
ls ./bsp_out/ps7_cortexa9_0/libsrc/
. . .
myip_v1_0
. . .
```

XILINX
ALL PROGRAMMABLE™

Appendix G

Microprocessor Application Definition (MAD)

Microprocessor Application Definition (MAD) Overview

A MAD file contains directives for customizing software application. This section describes the Microprocessor Application Definition (MAD) format, Platform Specification Format 2.1.0. and the parameters that can be used to customize applications.

Requirements

Each application has an MAD file and a Tool Command Language (Tcl) file associated with it.

The MAD file is used by Hsi to recognize it as an application and to consider its configuration while generating the application sources. The MAD file for each application must be located in its data directory.

Microprocessor Application Definition Files

Application Definition involves defining a Microprocessor Application Definition file (MAD) and a Data Generation file (Tcl file).

Application Definition File

The MAD file (`<application_name>.mad`) contains the name, description, and other configurable parameters. A detailed description of the various parameters and the MAD format is described in MAD Format Specification.

Data Generation File

The second file (`<application_name>.tcl`, with the filename being the same as the MAD filename) uses the parameters in the MAD file for the application to generate data..

Data generated includes, but is not limited to, generation of header files, C files, running DRCs for the application, 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 following:

- `DRC` (swapp_is_supported_hw, swapp_is_supported_sw)
- `swapp_generate` (tool defined procedure) called after application source files are copied

MAD Format Specification

The MAD format specification involves the MAD file format specification and the Tcl file format specification.

MAD File Format Specification

The MAD file format specification describes the parameters using a sample MAD file and its corresponding Tcl file.

The following example shows a MAD file for a sample application called my_application.

```
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.

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

```
BEGIN APPLICATION my_application
```

The `BEGIN APPLICATION` construct defines the start of an application named `my_application`.

```
OPTION NAME = myapplication
OPTION DESCRIPTION = "My custom application"
END APPLICATION
```

***Note*:** The application NAME should match the return value of the Tcl process swapp_get_name in the application Tcl file described above.

Tcl File Format Specification

Each application has a Tcl file associated with the MAD file. This Tcl file has the following sections:

- **DRC Section**: This section contains Tcl routines that validate your hardware and software instances and their configuration needed for the application.
- **Generation Section**: This section contains Tcl routines that generate the application header and C files based on the hardware and software configuration.

MAD Format Example

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

Example: MAD File

The following is an example of an MAD file for a sample application called my_application.

```
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.

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

```
BEGIN APPLICATION my_application
```

The `BEGIN APPLICATION` construct defines the start of an application named `my_application`.

```
OPTION NAME = myapplication
OPTION DESCRIPTION = "My custom application"
END APPLICATION
```

***Note*:** Applicaiton NAME should match the return value of Tcl proc swapp_get_name in application Tcl file described above.

Tcl Commands Listed Alphabetically

This section contains all Hardware Software Interface Tcl commands, arranged alphabetically.

common::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 namefor 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` - (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` - (Optional) The type of property to create. 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

Create a property called PURPOSE for cell objects:

```
common::create_property PURPOSE cell
```

***Note*:** Because the type was not specified, the value will default to "strings".

Create a pin property called COUNT which holds an Integer value:

```
common::create_property -type int COUNT pin
```

See Also

- common::get_property
- common::list_property
- common::list_property_value
- common::report_property
- common::reset_property
- common::set_property

common::get_msg_config

Returns the current message count, limit, or the message configuration rules previously defined by the set_msg_config command.

Syntax

```
get_msg_config [-id <arg>] [-severity <arg>] [-rules] [-limit] [-count] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-id]	The message id to match. Should be used in conjunction with -limit or -count Default: empty
[-severity]	The message severity to match. Should be used in conjunction with -limit or -count Default: empty
[-rules]	Show a table displaying all message control rules for the current project
[-limit]	Show the limit for the number of messages matching either -id or -severity that will be displayed
[-count]	Show the number of messages matching either -id or -severity that have been displayed
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

Report

common::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

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

Get the current value of the messaging parameter used for enabling the description:

```
common::get_param messaging.enableDescription
```

See Also

- common::list_param
- common::report_param
- common::reset_param
- common::set_param

common::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 named property, 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 named property, 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.

<object> - (Required) One or more objects to examine for the specified property.

Examples

Get the NAME property from the specified cell:

```
common::get_property NAME [lindex [get_cells] 0]
```

Get the BOARD property from the current hardware design:

```
common::get_property BOARD [current_hw_design]
```

See Also

- common::create_property
- hsi::get_cells
- hsi::get_ports
- common::report_property
- common::reset_property
- common::set_property

common::help

Display help for one or more topics.

Syntax

```
 help [-category <arg>] [-args] [-syntax] [-long] [-prop <arg>] [-class
<arg>] [-message <arg>] [-quiet] [-verbose] [<pattern_or_object>]
```

Returns

Nothing

Usage

Name	Description
[-category]	Search for topics in the specified category
[-args]	Display arguments description
[-syntax]	Display syntax description
[-long]	Display long help description
[-prop]	Display property help for matching property names Default: *
[-class]	Display object type help
[-message]	Display information about the message with the given message. Every message delivered by the tool has a unique global message ID that consists of an application sub-system code and a message identifier. Example: -message {Common 17-8}.
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<pattern_or_object>]	Display help for topics that match the specified pattern Default: *

Categories

Project

common::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

Get a list of all user-definable parameters:

```
common::list_param
```

See Also

- common::get_param
- common::report_param
- common::reset_param
- common::set_param

common::list_property

List properties of object.

Syntax

```
 list_property [-class <arg>] [-regexp] [-quiet] [-verbose] [<object>]
[<pattern>]
```

Returns

List of property names

Usage

Name	Description
[-class]	Object type to query for properties. Ignored if object is specified.
[-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

Description

Gets a list of all properties on a specified object or class.

***Note*:** report_property returns a list of properties on an object or class of objects and 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>

<-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 CELL object:

```
common::list_property [get_cells microblaze_0]
```

See Also

- common::create_property
- hsi::get_cells
- common::list_property_value
- common::report_property
- common::reset_property
- common::set_property

common::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

common::load_features

Load Tcl commands for a specified feature.

Syntax

```
load_features [-quiet] [-verbose] [<features>...]
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<features>]	Feature(s) to load, use list_features for a list of available features.

Categories

Tools

common::register_proc

Register a Tcl proc with Vivado.

Syntax

```
register_proc [-quiet] [-verbose] <proc> [<tasknm>]
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<proc>	Name of proc to register. Proc must be known to Tcl
[<tasknm>]	Name of Tcl task that wraps the proc. Default: Register the proc using the root name proc (no namespaces).

Categories

Tools

common::report_environment

Report system information.

Syntax

```
 report_environment [-file <arg>] [-format <arg>] [-append] [-
return_string] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-file]	Write system information to specified file.
[-format]	Specifies how to format the report. Default is 'text', another option is 'xml'. Only applies if -file is used. If xml output is used, -append is not allowed. Default: text
[-append]	Append report to existing file
[-return_string]	Return report content as a string value
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

Report

common::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:

```
common::report_param
```

The following example returns the name, value, and description of user-definable parameters that match the specified search pattern:

```
common::report_param *coll*
```

See Also

- common::get_param
- common::list_param
- common::reset_param
- common::set_param

common::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
[-class]	Object type to query for properties. Not valid with <object>
[-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 -classoption 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:

```
common::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:

```
common::report_property -all [current_hw_design]
```

```
common::report_property -all [current_sw_design]
```

See Also

- common::create_property
- hsi::get_cells
- common::get_property
- common::list_property
- common::list_property_value
- common::reset_property
- common::set_property

common::reset_msg_config

Resets or removes a message control rule previously defined by the set_msg_config command.

Syntax

```
reset_msg_config [-string <args>] [-id <arg>] [-severity <arg>] [-limit]
[-suppress] [-count] [-default_severity] [-regexp] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-string]	A qualifier, apply the selected operation only to messages that contain the given strings Default: empty
[-id]	A qualifier, the message id to match. If not specified, all message ids will be matched
[-severity]	A qualifier, apply the selected operation only to messages at the given severity level
[-limit]	reset the limit values for message controls that match the given qualifiers for the current project
[-suppress]	stop suppressing messages that match the given qualifiers for the current project
[-count]	reset the count of messages for all message controls that match the given qualifiers for the current project. This will prevent messages from being suppressed by a -limit control until the message count once again exceeds the specified limit.
[-default_severity]	reset the message severity of all messages controls for the current project that match the given qualifiers to their default value
[-regexp]	The values used for -string are full regular expressions
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

Report

common::reset_msg_count

Reset message count.

Syntax

```
reset_msg_count [-quiet] [-verbose] <id>
```

Returns

New message count

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<id>	Unique message Id to be reset, e.g. "Common 17-99". "reset_msg_count -id *" reset all counters

Categories

Report

common::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:

```
common::reset_param tcl.statsThreshold
```

See Also

- common::get_param
- common::list_param
- common::report_param
- common::set_param

common::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:

```
common::set_property CONFIG.archiver armar [get_sw_processor]
```

```
common::reset_property CONFIG.archiver armar [get_sw_processor]
```

See Also

- common::create_property
- hsi::get_cells
- common::get_property
- common::list_property
- common::list_property_value
- common::report_property
- common::set_property

common::set_msg_config

Configure how the Vivado tool will display and manage specific messages based on message ID, string, or severity.

Syntax

```
set_msg_config [-id <arg>] [-string <args>] [-severity <arg>] [-limit
<arg>] [-new_severity <arg>] [-suppress] [-regexp] [-quiet] [-verbose]
```

Returns

Nothing

Usage

Name	Description
[-id]	A qualifier, apply the selected operation only to messages that match given message id. Example: '-id {Common 17-35}'. Default: match any id
[-string]	A qualifier, apply the selected operation only to messages that contain the given list of strings. Default: none
[-severity]	A qualifier, apply the selected operation only to messages at the given severity level. Example: '-severity INFO' Default: match any severity
[-limit]	for the messages that match the qualifiers, limit the number of messages displayed to the given integer value. Can only be used in conjunction with one of -id or -severity.
[-new_severity]	for the messages that match the qualifiers, change the severity to the given value for the current project
[-suppress]	for the messages that match the qualifiers, suppress (do not display) any messages for the current project
[-regexp]	The values used for -string are full regular expressions
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

Report

common::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

common::set_param messaging.defaultLimit 1000

See Also

- common::get_param
- common::list_param
- common::report_param
- common::reset_param

common::set_property

Set property on object(s).

Syntax

```
set_property [-dict <args>] [-quiet] [-verbose] <name> <value> <objects>..
.
```

Returns

Nothing

Usage

Name	Description
[-dict]	list of name/value pairs of properties to set
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<name>	Name of property to set. Not valid with -dict option
<value>	Value of property to set. Not valid with -dict option
<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:

```
common::create_property -type bool truth cell
common::set_property truth false [lindex [get_cells] 1]
```

The following example sets the compiler and archiver property value for the specified software processor:

```
common::set_property CONFIG.archiver armar [get_sw_processor]
common::set_property CONFIG.compiler armcc [get_sw_processor]
```

See Also

- common::create_property
- hsi::get_cells
- common::get_property
- common::list_property
- common::list_property_value
- common::report_property
- common::reset_property

common::unregister_proc

Unregister a previously registered Tcl proc.

Syntax

```
unregister_proc [-quiet] [-verbose] <tasknm>
```

Returns

Nothing

Usage

Name	Description
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
<tasknm>	Name of Tcl task to unregister. The task must be wrapping a proc.

Categories

Tools

common::version

Returns the build for hsi and the build date.

Syntax

```
version [-short] [-quiet] [-verbose]
```

Returns

Hsi version

Usage

Name	Description
[-short]	Return only the numeric version number
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

Report

hsi::add_library

Add software library to software design.

Syntax

```
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 the specified Library to the current software design:

```
hsi::add_library xilffs
```

adds version of the library

```
hsi::add_library xilrsa 1.0
```

See Also

- hsi::current_sw_design

hsi::close_hw_design

Close a hardware design.

Syntax

```
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 HSM 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.

Examples

Close the current hardware design object:

```
hsi::close_hw_design [current_hw_design]
```

Close the specified hardware design object:

```
hsi::close_hw_design design_1_imp
```

See Also

- hsi::current_hw_design
- hsi::get_hw_designs
- hsi::open_hw_design

hsi::close_sw_design

Close a software design.

Syntax

```
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 Hardware Software Interface session.

IMPORTANT!: *If the design has been modified, you will not be prompted to save the design prior to closing. In the current release the 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.

Examples

Close the current software design in the current session:

```
hsi::close_sw_design [current_sw_design]
```

```
hsi::close_sw_design
```

Close the specified software design in the current session:

```
hsi::close_sw_design [current_sw_design]
```

See Also

- hsi::create_sw_design
- hsi::current_sw_design
- hsi::get_sw_designs
- hsi::open_sw_design

hsi::create_comp_param

Add Parameter.

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).

If successful, this command returns the name of the param created. Otherwise it returns an error.

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.

Examples

The following example creates a new param called p1 to specified driver:

```
hsi::create_comp_param p1 [get_drivers ps7_uart_1]
```

The following example creates a new param called p2 to all drivers.

```
hsi::create_comp_param p2 [get_drivers]
```

See Also

- hsi::create_node
- hsi::create_sw_design

hsi::create_dt_node

Create a DT Node.

Syntax

```
 create_dt_node -name <arg> [-unit_addr <arg>] [-label <arg>] [-objects
<args>] [-quiet] [-verbose]
```

Returns

DT Node object. Returns nothing if the command fails

Usage

Name	Description
-name	child DT node name
[-unit_addr]	unit address of node
[-label]	label of node
[-objects]	List of nodes
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

DeviceTree

Description

Create a new DT node and add to the current DT tree.

If successful, this command returns the name of the DT node created where name is represented as <node_label>+<node_name>+@<unit_address>. Otherwise it returns an error.

Arguments

-name - The name of the node to be created.

-label - The label of the node to represent in generated dtsi file.

--unit_addr - The unit address of the node to represent in generated dtsi file.

`-objects` - The list of node objects where the newly created node will be a child to all specified nodes.

`-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

Create a new DT node amba with lable axi_interconnect and unit_addr 0x000 in the current DT tree:

```
hsi::create_dt_node -name amba -label axi_interconnect -unit_addr 0x0000
```

```
hsi::create_dt_node -name amba -label axi_interconnect -unit_addr 0x0000 -
objects [get_dt_nodes -of_objects\ [current_dt_tree]
```

See Also

- hsi::get_dt_nodes
- hsi::current_dt_tree

hsi::create_dt_tree

Create a DT tree.

Syntax

```
create_dt_tree -dts_file <arg> [-dts_version <arg>] [-quiet] [-verbose]
```

Returns

Tree object. Returns nothing if the command fails

Usage

Name	Description
`-dts_file`	dts file name
[`-dts_version`]	dts version
[`-quiet`]	Ignore command errors
[`-verbose`]	Suspend message limits during command execution

Categories

DeviceTree

Description

Create a new DT tree add to the current HSI session.

If successful, this command returns the name of the DT tree created. Otherwise it returns an error.

Arguments

`-dts_file` - The DT tree name or file name targeted for the output DTSI file.

`-dts_version` - The DTS version of the DTSI file.

`-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

Create a new DT tree pl.dtsi and add the tree to the current session:

```
hsi::create_dt_tree -dts_file pl.dtsi -dts_version /dts-v1/
```

```
hsi::create_dt_tree -dts_file system.dts -dts_version /dts-v3/ -header
"include pl.dtsi, include ps.dtsi"
```

```
hsi::create_dt_tree -dts_file ps.dtsi -dts_version /dts-v3/ -header "PS
system info"
```

See Also

- hsi::current_dt_tree
- hsi::get_dt_trees

hsi::create_node

Add 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).

If successful, this command returns the name of the node created. Otherwise it returns an error.

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.

Examples

The following example creates a new node called n1 to the specified driver:

```
hsi::create_node n1 [get_drivers ps7_uart_1]
```

The following example creates a new node called n2 to all drivers.

```
hsi::create_node n2 [get_drivers]
```

See Also

- hsi::create_comp_param
- hsi::create_sw_design

hsi::create_sw_design

Create a software design.

Syntax

```
 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.

If successful, this command returns the name of the software design created. Otherwise it returns an error.

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 is standalone.

`-os_ver` - (Optional) The OS version targeted for the software design. Default value is the 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.

Examples

Create a new software design module called sw_design_1 and add the module to the current session:

```
hsi::create_sw_design sw_design_1 -proc microblaze_0 -os xilkernel
```

```
hsi::create_sw_design sw_design_1 -proc microblaze_0 -os xilkernel -os_ver
6.0
```

```
hsi::create_sw_design sw_design_1 -proc ps7_cortexa9_0
```

See Also

- hsi::close_sw_design
- hsi::current_sw_design
- hsi::get_sw_designs
- hsi::open_sw_design

hsi::current_dt_tree

Set or get current tree.

Syntax

```
current_dt_tree [-quiet] [-verbose] [<tree>]
```

Returns

Tree object, "" if failed

Usage

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

Categories

DeviceTree

Description

Defines the current DT tree for use with the Hardware Software Interface, or returns the name of the current DT tree in the active session.

The current DT tree is the target of the Hardware Software Interface - DeviceTree Tcl commands.

You can use the get_dt_trees command to get a list of created DT trees 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.

<dt_tree> - (Optional) The name of a DT tree to set as the current DT tree in the session. If a dt_tree is not specified, the command returns the current DT tree of the active session.

Examples

Get the current DT tree object:

```
hsi::current_dt_tree
```

OR

Set the specified dt_tree as the current session:

```
hsi::current_dt_tree pl.dtsi
```

See Also

- hsi::create_dt_tree
- hsi::get_dt_trees

hsi::current_hw_design

Set or get current hardware design.

Syntax

```
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 Hardware Software Interface, or returns the name of the current design in the active project.

The current hardware design is the target of the Hardware Software Interface 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 a hardware design to set as the current design in the session. If a design is not specified, the command returns the current hardware design of the active session.

Examples

Get the current hardware design object:

```
hsi::current_hw_design
```

OR

Set the specified hardware design object as the current design:

```
hsi::current_hw_design hw_design_1
```

See Also

- hsi::close_hw_design
- hsi::get_hw_designs
- hsi::open_hw_design

hsi::current_hw_instance

Set or reset current hardware instance.

Syntax

```
current_hw_instance [-quiet] [-verbose] [<instance>]
```

Returns

Current cell instance object, "" if failed

Usage

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

Categories

Hardware

Description

Sets or resets the current hardware instance to the given hierarchial instance, for scoping the hardware data access commands in a hierarchial design. Default is top design.

Arguments

`-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.

<instance> - (Optional) Hardware cell instance (hierarchial instance) to be set.

Examples

The following example sets the current hardware instance to axi_ethernet_0:

```
hsi::current_hw_instance [get_cells axi_ethernet_0]
```

```
hsi::get_cells
```

Returns the hardware cell instances inside current hierarchial instance `axi_ethernet_0`.

See Also

- hsi::get_cells
- hsi::get_nets
- hsi::get_intf_pins
- hsi::get_intf_nets

hsi::current_sw_design

Set or get current software design.

Syntax

```
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

Sets the current software design for use with the Hardware Software Interface, or returns the name of the current design in the active project.

The current software design is the target of the Hardware Software Interface software Tcl commands.

You can use the get_sw_designs command to get a list of open software 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 a software design to set as the current design in the Hardware Software Interface. If a design is not specified, the command returns the current software design of the active project.

Examples

Get the current software design object:

```
hsi::current_sw_design
```

OR

Set the specified software design object as the current design:

```
hsi::current_sw_design sw_design_1
```

See Also

- hsi::close_sw_design
- hsi::create_sw_design
- hsi::get_sw_designs
- hsi::open_sw_design

hsi::delete_objs

Delete specified objects.

Syntax

```
 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.

Examples

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

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

hsi::generate_app

Generates Template Application.

Syntax

```
 generate_app [-dir <arg>] [-hw <arg>] [-hw_xml <arg>] [-sw <arg>] [-proc
<arg>] [-os <arg>] [-os_ver <arg>] [-app <arg>] [-lapp] [-sapp] [-compile]
[-quiet] [-verbose]
```

Returns

Returns nothing

Usage

Name	Description
[-dir]	Output directory where App needs to be generated
[-hw]	Hardware Design Name
[-hw_xml]	Hardware Design XML File Path
[-sw]	Software Design Name
[-proc]	Instance Name of the Processor to which App needs to be generated
[-os]	Name of Operating System for App
[-os_ver]	Version of Operating System for App
[-app]	Name of the Application
[-lapp]	List all the Applications in Repositories
[-sapp]	List all the Supported Applications for Hardware and Software Designs
[-compile]	Compile the generated source files
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

Software

Description

Generates a template application for a processor on an operating system

If successful, this command returns nothing. Otherwise it returns an error.

Arguments

`-dir` - (Optional) The output where the application needs to be generated.

`-hw` - (Optional) Hardware design name.

`-hw_xml` - (Optional) XML file path of the hw design.

`-sw` - (Optional) Software design name.

`-proc` - Instance Name of the Processor to which App needs to be generated.

`-os` - (Optional) Name of Operating System for App.

`-os_ver` - (Optional) Version of Operating System for App.

`-app` - Name of the application.

`-lapp` - (Optional) List all the Applications in Repositories.

`-sapp` - (Optional) List all the Supported Applications for Hardware and Software Designs.

`-compile` - (Optional) Compile the generated source files.

`-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

List all applications in the repositories:

```
hsi::generate_app -lapp
```

The following example gets a list of all supported applications for processor ps7_cortexa9_0 and stanalone operating system:

```
hsi::generate_app -sapp -proc ps7_cortexa9_0
```

The following example generates a hello_world application for processor microblaze_0 for xilkernel OS:

```
hsi::generate_app -app hello_world -proc microblaze_0 -os xilkernal
```

See Also

- hsi::generate_bsp
- hsi::generate_target

hsi::generate_bsp

Generates Board Support Package.

Syntax

```
 generate_bsp [-dir <arg>] [-hw <arg>] [-hw_xml <arg>] [-sw <arg>] [-
sw_mss <arg>] [-proc <arg>] [-os <arg>] [-os_ver <arg>] [-compile] [-
quiet] [-verbose]
```

Returns

Returns nothing

Usage

Name	Description
[-dir]	Output directory where BSP needs to be generated
[-hw]	Hardware Design Name
[-hw_xml]	Hardware Design XML File Path
[-sw]	Software Design Name
[-sw_mss]	Software Design MSS File Path
[-proc]	Instance Name of the Processor to which BSP needs to be generated
[-os]	Name of Operating System for BSP
[-os_ver]	Version of Operating System for BSP
[-compile]	Compile the generated source files
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution

Categories

Software

Description

Generates a template application for a processor on an operating system

If successful, this command returns nothing. Otherwise it returns an error.

Arguments

`-dir` - (Optional) Output directory where BSP needs to be generated.

`-hw` - (Optional) Hardware design name.

`-hw_xml` - (Optional) XML file path of the hw design.

`-sw` - (Optional) Software design name.

`-sw_mss` - (Optional) Software Design MSS File Path.

`-proc` - Instance Name of the Processor to which BSP needs to be generated.

`-os` - (Optional) Name of Operating System for BSP.

`-os_ver` - (Optional) Version of Operating System for BSP.

`-compile` - (Optional) Compile the generated source files.

`-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

generates a BSP for the processor ps7_cortexa9_0:

```
hsi::generate_bsp -proc ps7_cortexa9_0
```

The following example generates BSP for the processor ps7_cortexa9_0 for a MSS file sw_app.mss:

```
hsi::generate_bsp -sw_mss sw_app.mss
```

See Also

- hsi::generate_app
- hsi::generate_target

hsi::generate_target

Generates Target.

Syntax

```
 generate_target [-dir <arg>] [-compile] [-quiet] [-verbose] [<name>]
[<objects>]
```

Returns

Returns nothing

Usage

Name	Description
[-dir]	Output directory where target needs to be generated
[-compile]	Compile the generated source files
[-quiet]	Ignore command errors
[-verbose]	Suspend message limits during command execution
[<name>]	List of targets to be generated, or 'all' to generate all supported targets
[<objects>]	The objects for which target needs to be generate

Categories

Software

hsi::get_arrays

Get a list of software Arrays.

Syntax

```
 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 cell 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 based on property values on the objects. You can find the properties on an object with the report_property or list_property commands.

You should quote the filter search pattern 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 ||).

For array objects, "NAME", and "other config parameters" are some of the properties that can be used to filter results.

`-of_objects`<arg> - (Optional) Get the arrays that are available in OS, Drivers, Libraries, Processor, and Core, as returned by the get_os, get_drivers, get_libs, get_sw_processor, or get_sw_cores commands.

***Note*:** The -of_objects option requires objects to be specified using one of the 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 arrays. More than one pattern can be specified to find multiple arrays 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 arrays present in all software cores(drivers/libs/os)

```
hsi::get_arrays
```

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

```
hsi::get_arrays mem_table
```

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

```
hsi::get_arrays -of_objects [get_os]
```

See Also

- hsi::get_sw_interfaces

hsi::get_cells

Get a list of cells.

Syntax

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

Returns

Cell objects. Returns nothing if the command fails

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-hierarchical]	Get cells from all levels of hierarchical cells
[-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 design that match a specified search pattern. The default command returns a list of all IP instances in the 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 based on property values on the objects. You can find the properties on an object with the report_property or list_property commands.

You should quote the filter search pattern 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 ||).

For cell objects, "IP_TYPE", and "IP_NAME" are some of the properties you can use to filter results. The following gets cells with an IP_TYPE of "PROCESSOR" and with names containing "ps7":

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

`-hierarchical` - (Optional) Get cells from all levels of hierarchical cells.

`-of_objects`<arg> - (Optional) Get the cells connected to the specified pins, timing paths, nets, bels, clock regions, sites, or DRC violation 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:

```
hsi::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:

```
common::report_property [lindex [get_cells] 1]
```

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

See Also

- hsi::get_nets
- hsi::get_pins
- hsi::get_ports
- hsi::get_intf_nets
- hsi::get_intf_pins
- hsi::get_intf_ports
- common::list_property
- common::report_property

hsi::get_comp_params

Get a list of parmas.

Syntax

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

Returns

Property objects. Returns nothing if the command fails

Usage

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

Categories

Software

Description

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 based on property values on the objects. You can find the properties on an object with the report_property or list_property commands.

You should quote the filter search pattern 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 match NAME and VALUE within their name:

```
get_comp_params -filter {NAME == clock-names && VALUE == "ref_clk
aper_clk"}
```

`-of_objects`<arg> - (Optional) Get 'node' objects of these types: 'sw_driver', 'sw_os', 'sw_proc', 'sw_node'.

***Note*:** The -of_objects option requires objects to be specified using the get_* commands, such as get_nodes, 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 sotfware 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 the software design:

```
hsi::get_comp_params -of_objects [get_drivers ps7_uart_0]
```

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

```
hsi::get_comp_params -of_objects [get_os]
```

See Also

- hsi::get_nodes
- hsi::get_drivers

hsi::get_drivers

Get a list of software driver instances.

Syntax

```
 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 based on property values on the objects. You can find the properties on an object with the report_property or list_property commands.

You should quote the filter search pattern 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 ||).

NAME and HW_INSTANCE are some of the properties you can use to filter results for drivers. The following gets drivers named gpio with a HW_INSTANCE of axi_gpio_0:

```
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 sotfware 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:

```
hsi::get_drivers
```

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

```
hsi::get_drivers * -filter {NAME==gpio]
```

See Also

- hsi::get_libs
- hsi::get_os
- hsi::get_sw_processor

hsi::get_dt_nodes

Get a list of DT node objects.

Syntax

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

Returns

Node objects. Returns nothing if the command fails

Usage

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

Categories

DeviceTree

Description

Gets a list of DT nodes created under a DT tree in the current HSI session that match a specified search pattern. The default command gets a list of all root DT nodes in the current DT tree.

Arguments

-of_objects<arg> - (Optional) Gets all nodes of DTSNode and DTSTree

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

`-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 based on property values on the objects. You can find the properties on an object with the report_property or list_property commands.

`-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 nodes against the specified patterns. The default pattern is the wildcard `*` which gets a list of all root nodes in the current DT tree. More than one pattern can be specified to find multiple nodes 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 root nodes attached to the specified DT tree:

```
hsi::get_dt_nodes -of_objects [lindex [get_dt_trees] 1]
```

Note: If there are no nodes matching the pattern, the tool will return empty.

The following example gets a list of all nodes in the current DT tree:

```
hsi::get_dt_nodes -hier
```

Note: If there are no nodes matching the pattern, the tool will return empty.

The following example gets a list of nodes created under a root node:

```
hsi::get_dt_nodes -of_objects [current_dt_tree]
```

***Note*:** If there are no nodes matching the pattern, the tool will return empty.

See Also

- hsi::current_dt_tree
- hsi::create_dt_node

hsi::get_dt_trees

Get a list of dts trees created.

Syntax

```
get_dt_trees [-regexp] [-filter <arg>] [-quiet] [-verbose] [<patterns>...]
```

Returns

DTS tree 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 tree names against patterns Default: *

Categories

DeviceTree

Description

Gets a list of DT trees created in the current HSI session that match a specified search pattern. The default command gets a list of all open DT trees in the HSI 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 based on property values on the objects. You can find the properties on an object with the report_property or list_property commands.

You should quote the filter search pattern 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 ||).

For the "DT tree" object you can use the"DTS_FILE_NAME" property to filter results. The following gets dt trees that do NOT contain the "pl.dtsi" substring within their name:

```
get_dt_trees * -filter {NAME !~ "*pl.dtsi*"}
```

`-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 DT trees against the specified patterns. The default pattern is the wildcard `*` which gets all DT trees. More than one pattern can be specified to find multiple trees based on different search criteria.

Examples

Get all created DT trees in the current session:

```
hsi::get_dt_trees
```

See Also

- hsi::current_dt_tree
- hsi::create_dt_tree

hsi::get_fields

Get a list of fields of a register.

Syntax

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

Returns

Register objects. Returns nothing if the command fails

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-of_objects]	Get 'field' objects of these types: 'register'.
[-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 fields of a register. By default, the command returns the fields of all registers, of all the cells.

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 based on property values on the objects. You can find the properties on an object with the report_property or list_property commands.

You should quote the filter search pattern 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 ||).

For array objects, "NAME", and "other config parameters" are some of the properties that can be used to filter results.

`-of_objects`<arg> - (Optional) Get 'field' objects of these types: 'register'.

***Note*:** The -of_objects option requires objects to be specified using the get_registers command, 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 field names against the specified patterns. The default pattern is the wildcard `*` which gets a list of all fields. More than one pattern can be specified to find multiple fields 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 fields present in all registers of all the software driver cores:

```
hsi::get_fields
```

hsi::get_hw_designs

Get a list of hardware designs opened.

Syntax

```
 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 HSM 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 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 based on property values on the objects. You can find the properties on an object with the report_property or list_property commands.

You should quote the filter search pattern 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 ||).

For the "Hardware design" object you can use the"NAME" property to filter results. The following gets hardware designs that do NOT contain the "design" 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

Get all open hardware designs in the current session:

```
hsi::get_hw_designs
```

See Also

- hsi::close_hw_design
- hsi::current_hw_design

- hsi::open_hw_design

hsi::get_hw_files

Get a list of hardware design supporting files.

Syntax

```
 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 based on property values on the objects. You can find the properties on an object with the report_property or list_property commands.

You should quote the filter search pattern 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 ||).

For "file" objects you can use the "TYPE" property to filter results.

`-of_objects`<args> - (Optional) Get the files that are associated with the specified fileset objects. The default is to search all filesets. When -compile_order and -used_in are specified, the -of_objects switch will only accept a single fileset, or a single sub-design such as an IP core, Block Design, or DSP design. A sub-design is also known as a composite file.

***Note*:** The -of_objects option requires objects to be specified using one of 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:

```
hsi::get_hw_files -filter {TYPE == bit}
```

hsi::get_intf_nets

Get a list of interface Nets.

Syntax

```
 get_intf_nets [-regexp] [-filter <arg>] [-boundary_type <arg>] [-
hierarchical] [-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
[-boundary_type]	Used when source object is on a hierarchical block's pin. Valid values are 'upper', 'lower', or 'both'. If 'lower' boundary, searches from the lower level of hierarchy onwards. This option is only valid for connected_to relations. Default: upper
[-hierarchical]	Get interface nets from all levels of hierarchical cells
[-of_objects]	Get 'intf_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 based on property values on the objects. You can find the properties on an object with the report_property or list_property commands.

You should quote the filter search pattern 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 ||).

For hardware design nets you can use the "NAME" property to filter results.

`-hierarchical` - (Optional) Get interface nets from all levels of hierarchical cells.

`-boundary_type` - (Optional) Used when source object is on a hierarchical block's pin. Valid values are 'upper', 'lower', or 'both'. If 'lower' boundary, searches from the lower level of hierarchy onwards. This option is only valid for connected_to relations.

`-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 one of 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:

```
hsi::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

- hsi::get_cells
- hsi::get_nets
- hsi::get_pins
- hsi::get_ports
- hsi::get_intf_pins
- hsi::get_intf_ports
- common::list_property
- common::report_property

hsi::get_intf_pins

Get a list of interface Pins.

Syntax

```
 get_intf_pins [-regexp] [-filter <arg>] [-hierarchical] [-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
[-hierarchical]	Get interface pins from all levels of hierarchical cells
[-of_objects]	Get 'bus_intf' objects of these types: 'hw_design cell port 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 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 based on property values on the objects. You can find the properties on an object with the report_property or list_property commands.

You should quote the filter search pattern 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 ||).

"NAME" and "TYPE" are some of the properties you can use to filter results for interface pins. The following gets slave interface pins that do NOT contain the "S_AXI" substring within their name:

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

`-hierarchical` - (Optional) Get interface pins from all levels of hierarchical cells.

`-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 one of 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>

<-match_style [sdc | ucf]> - (Optional) Indicates that the search pattern matches UCF constraints or SDC constraints. The default is SDC.

`-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:

```
hsi::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

- hsi::get_cells
- hsi::get_nets
- hsi::get_pins
- hsi::get_ports
- hsi::get_intf_nets
- hsi::get_intf_ports
- common::list_property
- common::report_property

hsi::get_intf_ports

Get a list of interface Ports.

Syntax

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

Returns

Interface Port 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 port 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 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 an IP subsystem design are ports, or interface ports. The external connections in an IP Integrator cell or hierarchical module 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 based on property values on the objects. You can find the properties on an object with the report_property or list_property commands.

You should quote the filter search pattern 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 ||).

For IP subsystem interface ports, "DIRECTION", and "NAME" are some of the properties you can use to ilter results.

`-of_objects`<arg> - (Optional) Get the interface ports connected to the specified IP subsystem 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:

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

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

See Also

- hsi::get_cells
- hsi::get_nets
- hsi::get_pins
- hsi::get_ports
- hsi::get_intf_nets
- hsi::get_intf_pins
- common::list_property
- common::report_property

hsi::get_libs

Get a list of software libraries.

Syntax

```
 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.

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

You should quote the filter search pattern 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 ||).

"NAME", "VERSION", and "other config parameters" are some of the properties you can use to filter results for libraries. The following gets software libraries that which are named xilrsa and are version 1.0:

```
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_pins, or by the get_intf_pins commands.

***Note*:** The -of_objects option requires objects to be specified using one of 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:

```
hsi::get_libs
```

See Also

- hsi::get_drivers
- hsi::get_os
- hsi::get_sw_processor
- common::report_property

hsi::get_mem_ranges

Get a list of memory ranges.

Syntax

```
 get_mem_ranges [-regexp] [-filter <arg>] [-hierarchical] [-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
[-hierarchical]	Get memory ranges from all levels of hierarchical cells
[-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 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 argument filters the list of objects returned based on property values on the objects. You can find the properties on an object with the report_property or list_property commands.

You should quote the filter search pattern 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 ||).

`-hierarchical` - (Optional) Get memory ranges from all levels of hierarchical cells.

`-of_objects`<arg> - (Optional) Get the slaves of the specified 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:

```
hsi::get_mem_ranges
```

```
hsi::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.

hsi::get_nets

Get a list of nets.

Syntax

```
 get_nets [-regexp] [-filter <arg>] [-boundary_type <arg>] [-hierarchical]
[-of_objects <args>] [-quiet] [-verbose] [<patterns>...]
```

Returns

Net objects. Returns nothing if the command fails

Usage

Name	Description
[-regexp]	Patterns are full regular expressions
[-filter]	Filter list with expression
[-boundary_type]	Used when source object is on a hierarchical block's pin. Valid values are 'upper', 'lower', or 'both'. If 'lower' boundary, searches from the lower level of hierarchy onwards. This option is only valid for connected_to relations. Default: upper
[-hierarchical]	Get nets from all levels of hierarchical cells
[-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

`-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 based on property values on the objects. You can find the properties on an object with the report_property or list_property commands.

You should quote the filter search pattern 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 ||).

For the "hardware design " object you can use the "NAME" property to filter results.

<-boundary_type> - (Optional) Used when source object is on a hierarchical block's pin. Valid values are 'upper', 'lower', or 'both'. If 'lower' boundary, searches from the lower level of hierarchy onwards. This option is only valid for connected_to relations. Default: upper.

<-hierarchical> - (Optional) Get nets from all levels of hierarchical cells.

<-of_objects> - (Optional) Get 'net' objects of these types: 'hw_design cell port'.

`-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:

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

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

See Also

- hsi::get_cells
- hsi::get_pins
- hsi::get_ports
- hsi::get_intf_nets
- hsi::get_intf_pins
- hsi::get_intf_ports
- common::list_property
- common::report_property

hsi::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 based on property values on the objects. You can find the properties on an object with the report_property or list_property commands.

You should quote the filter search pattern 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'.

Note: The -of_objects option requires objects to be specified using the get_* commands, such as get_nodes, 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 sotfware 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 the software design:

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

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

```
hsi::get_nodes -of_objects [get_os]
```

See Also

- hsi::get_comp_params
- hsi::get_drivers

hsi::get_os

Get OS in the software design.

Syntax

```
 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.

Examples

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

```
hsi::get_os
```

See Also

- hsi::get_drivers
- hsi::get_libs
- hsi::get_sw_processor

hsi::get_pins

Get a list of pins.

Syntax

```
 get_pins [-regexp] [-filter <arg>] [-hierarchical] [-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
[-hierarchical]	Get pins from all levels of hierarchical cells
[-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 based on property values on the objects. You can find the properties on an object with the report_property or list_property commands.

You should quote the filter search pattern 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 ||).

"DIR" and "TYPE" are some of the properties you can use to filter results for pins. The following gets input pins that do NOT contain the "RESET" substring within their name:

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

`-hierarchical` - (Optional) Get pins from all levels of hierarchical cells.

`-of_objects`<arg> - (Optional) Get the pins connected to the specified IP subsystem cell or net.

Note: The -of_objects option requires objects to be specified using one of 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:

```
hsi::get_pins -of [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 subsystem net:

```
hsi::get_pins -of [get_nets ps7_axi_interconnect_0_M_AXI_BRESP]
```

See Also

- hsi::get_cells
- hsi::get_nets
- hsi::get_ports
- hsi::get_intf_pins
- hsi::get_intf_nets
- hsi::get_intf_ports
- common::list_property
- common::report_property

hsi::get_ports

Get a list of external ports.

Syntax

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

Returns

Port 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 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 external connections in an IP Integrator cell or hierarchical module 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 based on property values on the objects. You can find the properties on an object with the report_property or list_property commands.

You should quote the filter search pattern 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 ||).

For IP subsystem ports, "DIRECTION", "TYPE", and "SENSITIVITY" are some of the properties you can use to filter results.

`-of_objects`<arg> - (Optional) Get the ports connected to the specified IP subsystem nets returned by get_nets.

***Note*:** The -of_objects option requires objects to be specified using one of 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:

```
hsi::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

- hsi::get_cells
- hsi::get_nets
- hsi::get_pins
- hsi::get_intf_nets
- hsi::get_intf_pins
- hsi::get_intf_ports
- common::list_property
- common::report_property

hsi::get_sw_cores

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

Syntax

```
get_sw_cores [-regexp] [-filter <arg>] [-quiet] [-verbose] [<patterns>...]
```

Returns

Software core 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 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

`-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 based on property values on the objects. You can find the properties on an object with the report_property or list_property commands.

You should quote the filter search pattern 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 ||).

"NAME", "CORE_STATE", and "TYPE" are some of the properties you can use to filter results for sw_cores.

`-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"}
```

hsi::get_sw_designs

Get a list of software designs opened.

Syntax

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

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 based on property values on the objects. You can find the properties on an object with the report_property or list_property commands.

You should quote the filter search pattern 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 ||).

For software designs you can use "NAME" to filter results.

`-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

- hsi::close_sw_design
- hsi::create_sw_design
- hsi::current_sw_design
- hsi::open_sw_design

hsi::get_sw_interfaces

Get a list of software Interfaces.

Syntax

```
 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 cell 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 based on property values on the objects. You can find the properties on an object with the report_property or list_property commands.

You should quote the filter search pattern 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 ||).

sw_interface, "NAME", and "other config parameters" are some of the properties you can use to filter results.

`-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 one of 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 software interfaces against the specified patterns. The default pattern is the wildcard `*` which gets a list of all software interfaces. More than one pattern can be specified to find multiple software interfaces 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 interfaces present in all software cores(drivers/libs/os)

```
get_sw_interfaces
```

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

```
get_sw_interfaces stdout
```

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

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

See Also

- hsi::get_arrays

hsi::get_sw_processor

Get processor of the software design.

Syntax

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

Returns

Processor object. Returns nothing if the command fails

Usage

Name	Description
[-regexp]	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 or 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.

Examples

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

```
get_sw_processor
```

See Also

- hsi::get_drivers
- hsi::get_libs
- hsi::get_os

hsi::open_hw_design

Open a hardware design from disk file.

Syntax

```
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

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

If successful, this command returns a hardware design object representing the opened Hardware design. Otherwise it returns an error.

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 HSM. The name must include the file extension.

Examples

Open 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

- hsi::close_hw_design
- hsi::current_hw_design
- hsi::get_hw_designs

hsi::open_sw_design

Open a software design from disk file.

Syntax

```
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 a software design in the Hardware Software Interface.

If successful, this command returns software design object representing the opened Software design. Otherwise it returns an error.

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.

Examples

Open the specified software design in the current session:

```
open_sw_design sw_design_1
```

See Also

- hsi::close_sw_design
- hsi::create_sw_design
- hsi::current_sw_design
- hsi::get_sw_designs

hsi::set_repo_path

Set a list of software repository paths.

Syntax

```
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.

Examples

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

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

See Also

- hsi::get_sw_cores

XILINX
ALL PROGRAMMABLE™

Appendix I

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.

Please Read: Important Legal Notices