

# PlanAhead Tcl Command Reference Guide

UG789 (v 14.1) April 24, 2012



## Notice of Disclaimer

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

© Copyright 2002-2012 Xilinx Inc. All rights reserved. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. The PowerPC name and logo are registered trademarks of IBM Corp., and used under license. All other trademarks are the property of their respective owners.

# Introduction

---

## Overview of Tcl Capabilities in PlanAhead

The Tool Command Language (Tcl) is the scripting language integrated in the PlanAhead™ tool environment. Tcl is a standard language in the semiconductor industry for design constraints and Synopsys® Design Constraints (SDC).

SDC is the mechanism for communicating timing constraints for FPGA synthesis tools from Synopsys Synplify as well as other vendors, and is a timing constraint industry standard; consequently, the Tcl infrastructure is a “Best Practice” for scripting language.

Tcl lets you perform interactive queries to design tools in addition to executing automated scripts. Tcl offers the ability to “ask” questions interactively of design databases, particularly around tool and design settings and state. Examples are: querying specific timing analysis reporting commands live, applying incremental constraints, and performing queries immediately after to verify expected behavior without re-running any tool steps.

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

**Note** This chapter is not a comprehensive reference to Tcl commands. This chapter does provide references to Tcl resources, and describes the general capabilities of Tcl in the PlanAhead environment.

## Tcl Journal Files

When you invoke the PlanAhead tool, it writes the `PlanAhead.log` file to record the various command and operations performed during the design session. The PlanAhead tool also writes a file called `PlanAhead.jou` which is a journal of just the Tcl commands run during the session that can be used as a source to create new Tcl scripts.

**Note** A backup version of this file, called `planahead.jou_backup`, is written to save the details of the previous run.

Refer to Appendix A in the [PlanAhead User Guide \(UG632\)](#) for information regarding the location of these files.

## Tcl Help

The Tcl help command provides information related to the supported Tcl commands.

- `help` — Returns a list of Tcl command categories.

**help**

Command categories are groups of commands performing a specific function, like File I/O for instance.

- `help -category category` — Returns a list of commands found in the specified category.

**help -category object**

This example returns the list of Tcl commands for handling objects.

- `help pattern` — Returns a list of commands that match the specified search pattern. This form can be used to quickly locate a specific command from a group of commands.

**help get\_\***

This example returns the list of Tcl commands beginning with `get_`.

- `help command` — Provides detailed information related to the specified command.

**help get\_cells**

This example returns specific information of the `get_cells` command.

- `help -args command` — Provides an abbreviated help text for the specified command, including the command syntax and a brief description of each argument.

**help -args get\_cells**

- `help -syntax command` — Reports the command syntax for the specified command.

**help -syntax get\_cells**

## Starting the PlanAhead Tool

The PlanAhead tool provides three primary modes of operation:

- The GUI mode (default)
- Starting the PlanAhead tool using a Tcl command-line option (Batch mode)
- Tcl shell mode

The following subsections describe Batch and Tcl shell modes.

## Batch Mode

When you start the PlanAhead tool, it looks for a Tcl initialization script in two different locations:

1. `installdir/planAhead/scripts/init.tcl`
2. `userdir/Xilinx/PlanAhead/init.tcl`

Where:

`installdir` is the installation directory where the PlanAhead tool is installed, and

`userdir` is your home directory.

- ◆ For Windows: `%APPDATA%/Xilinx/PlanAhead/init.tcl`
- ◆ For Linux: `$HOME/.Xilinx/PlanAhead/init.tcl`

If `init.tcl` exists, in one or both of those locations, the PlanAhead tool sources this file; first from the installation directory and second from your home directory.

- The `init.tcl` file in the installation directory allows a company or design group to support a common initialization script for all users. Anyone starting the PlanAhead tool from that installation location sources that `init.tcl` script.
- The `init.tcl` file in the home directory allows each user to specify additional commands, or to override commands from the tool installation to meet their specific design requirements.

The `init.tcl` file is a standard Tcl command file that can contain any valid Tcl command supported by the PlanAhead tool. You can also source another Tcl script file from within `init.tcl` by adding the following statement:

```
source path_to_file/file_name.tcl
```

## General Tcl Syntax Guidelines

Tcl uses the Linux file separator (/) convention regardless of the OS on which you are operating.

The following subsections describe the general syntax guidelines for using Tcl in the PlanAhead application.

## Sourcing a Tcl Script

A Tcl script can be sourced from either one of the command-line options or from the GUI. Within the PlanAhead GUI you can source a Tcl script from **Tools > Run Tcl Script**.

You can source a Tcl script from a command-line option:

```
source file_name
```

When you invoke a Tcl script from the GUI, a progress bar is displayed and all operations in the GUI are blocked until the scripts completes.

There is no way to interrupt script execution during run time; consequently, standard OS methods of killing a process must be used to force interruption of the tool. If the process is killed, you lose any work done since your last save.

Typing **help source** in the Tcl console will provide additional information regarding the **source** command.

## Using Tcl Eval

When executing Tcl commands, you can use variable substitution to replace some of the command line arguments accepted or required by the Tcl command. However, you must use the Tcl **eval** command to evaluate the command line with the Tcl variable as part of the command.

For instance, the help command can take the **-category** argument, with one of a number of command categories as options:

```
help -category ipflow
```

You can define a variable to hold the command category:

```
set cat "ipflow"
```

You can then evaluate the variable in the context of the Tcl command:

```
eval help -category $cat
```

or,

```
set cat "-category ipflow"
eval help $cat
```

You can also use braces {} in place of quotation marks "" to achieve the same result:

```
set runblocksOptDesignOpts { -sweep -retarget -propconst -remap }
eval opt_design $runblocksOptDesignOpts
```

Typing **help eval** in the Tcl console will provide additional information regarding the **eval** command.

## General Syntax Structure

The general structure of the PlanAhead application Tcl commands is:

```
command [optional_parameters] required_parameters
```

Command syntax is of the verb-noun and verb-adjective-noun structure separated by the underscore ("\_") character.

Commands are grouped together with common prefixes when they are related.

- Commands that query things are generally prefixed with **get\_**.
- Commands that set a value or a parameter are prefixed with **set\_**.
- Commands that generate reports are prefixed with **report\_**.

The commands are exposed in the global namespace. Commands are "flattened," meaning there are no "sub-commands" for a command.

## Example Syntax

Following is an example of the return format on the **get\_cells -help** command:

get\_cells

Description:

Get a list of cells in the current design

Syntax:

```
get_cells [-hsc arg ] [-hierarchical] [-regexp] [-nocase] [-filter arg ]
          [-of_objects args ] [-match_style arg ] [-quiet] [ patterns ]
```

Returns:

list of cell objects

Usage:

Name	Optional	Default	Description
-----			
-hsc	yes	/	Hierarchy separator
-hierarchical	yes		Search level-by-level in current instance
-regexp	yes		Patterns are full regular expressions
-nocase	yes		Perform case-insensitive matching (valid only when -regexp specified)
-filter	yes		Filter list with expression
-of_objects	yes		Get cells of these pins or nets
-match_style	yes	sdcc	Style of pattern matching, valid values are ucf, sdcc
-quiet	yes		Ignore command errors
patterns	yes	*	Match cell names against patterns

Categories:

SDC, XDC, Object

## Unknown Commands

Tcl contains a list of built-in commands that are generally supported by the language, PlanAhead-specific commands which are exposed to the Tcl interpreter, and user-defined procedures.

Commands that do not match any of these known commands are sent to the OS for execution in the shell from the exec command. This lets users execute shell commands that might be OS-specific. If there is no shell command, then an error message is issued to indicate that no command was found.

## Return Codes

Some Tcl commands are expected to provide a return value, such as a list or collection of objects on which to operate. Other commands perform an action but do not necessarily return a value that can be used directly by the user. Some tools that integrate Tcl interfaces return a 0 or a 1 to indicate success or error conditions when the command is run.

To properly handle errors in Tcl commands or scripts, you should use the Tcl built-in command catch. Generally, the catch command and the presence of numbered info, warning, or error messages should be relied upon to assess issues in Tcl scripted flows.

PlanAhead tool Tcl commands return either TCL\_OK or TCL\_ERROR upon completion. In addition, the PlanAhead application sets the global variable \$ERRORINFO through standard Tcl mechanisms.

To take advantage of the \$ERRORINFO variable, use the following line to report the variable after an error occurs in the Tcl console:

```
puts $ERRORINFO
```

This reports specific information to the standard display about the error. For example, the following code example shows a Tcl script (procs.tcl) being sourced, and a user-defined procedure (loads) being run. There are a few transcript messages, and then an error is encountered at line 5.

```
Line 1: PlanAhead % source procs.tcl
Line 2: PlanAhead% loads
Line 3: Found 180 driving FFs
Line 4: Processing pin a_reg_reg[1]/Q...
Line 5: ERROR: [HD-Tcl 53] Cannot specify '-patterns' with '-of_objects'.
Line 6: PlanAhead% puts $errorInfo
Line 7: ERROR: [HD-Tcl 53] Cannot specify '-patterns' with '-of_objects'. While executing
"get_ports -of objects $pin" (procedure "my_report" line 6) invoked from within procs.tcl
```

You can add **puts \$ERRORINFO** into catch clauses in your Tcl script files to report the details of an error when it is caught, or use the command interactively in the Tcl console immediately after an error is encountered to get the specific details of the error.

In the example code above, typing the **puts \$ERRORINFO** command in line 6, reports detailed information about the command and its failure in line 7.

## First Class Tcl Objects and Relationships

The Tcl commands in the PlanAhead application provide direct access to the object models for netlist, devices, and projects. These objects are first-class which means they are more than just a string representation, and they can be operated on and queried. There are a few exceptions to this rule, but generally “things” can be queried as objects, and these objects have properties that can be queried and they have relationships that allow you to get to other objects.

## Object Types and Definitions

There are many object types in the PlanAhead application; this chapter provides definitions and explanations of the basic types. The most basic and important object types are associated with entities in a design netlist, and these types are listed in the following subsections:

### Cell

A cell is an instance, either primitive or hierarchical inside a netlist. Examples of cells include flip-flops, LUTs, I/O buffers, RAM and DSPs, as well as hierarchical instances which are wrappers for other collections of cells.

### Pin

A pin is a point of logical connectivity on a cell. A pin allows the internals of a cell to be abstracted away and simplified for easier use, and can either be on hierarchical or primitive cells. Examples of pins include clock, data, reset, and output pins of a flop.

### Port

A port is a special type of hierarchical pin, a pin on the top level netlist object, module or entity. Ports are normally attached to I/O pads and connect externally to the FPGA device.

**Net**

A net is a wire or collection of wires that eventually be physically connected directly together. Nets can be hierarchical or flat, but always sorts a collection of pins together.

**Clock**

A clock is a periodic signal that propagates to sequential logic within a design. Clocks can be primary clock domains or generated by clock primitives such as a DCM, PLL, or MMCM. A clock is the rough equivalent to a TIMESPEC PERIOD constraint in UCF and forms the basis of static timing analysis algorithms.

## Querying Objects

All first class objects can be queried by a `get_ Tcl` command that generally has the following syntax:

```
get_object_type pattern
```

Where pattern is a search pattern, which includes if applicable a hierarchy separator to get a fully qualified name. Objects are generally queried by a string pattern match applied at each level of the hierarchy, and the search pattern also supports wildcard style search patterns to make it easier to find objects, for example:

```
get_cells */inst_1
```

This command searches for a cell named `inst_1` within the first level of hierarchy under the top-level of hierarchy. To recursively search for a pattern at every level of hierarchy, use the following syntax:

```
get_cells -hierarchical inst_1
```

This command searches every level of hierarchy for any instances that match `inst_1`.

For complete coverage of syntax, see the specific online help for the individual command:

- **help get\_cells**
- **get\_cells -help**

## Object Properties

Objects have properties that can be queried. Property names are unique for any given object type. To query a specific property for an object, the following command is provided:

```
get_property property_name object
```

An example would be the `lib_cell` property on cell objects, which tells you what UniSim component a given instance is mapped to:

```
get_property lib_cell [get_cell inst_1]
```

To discover all of the available properties for a given object type, use the **report\_property** command:

```
report_property [get_cells inst_1]
```

The following table shows the properties returned for a specific object.

### Reported Properties for Specified Object

Key	Value	Type
bel	OLOGICE1.OUTFF	string
class	cell	string
iob	TRUE	string
is_blackbox	0	bool
is_fixed	0	bool
is_partition	0	bool
is_primitive	1	bool
is_reconfigurable	0	bool
is_sequential	1	bool
lib_cell	FD	string
loc	OLOGIC_X1Y27	string
name	error	string
primitive_group	FD_LD	string
primitive_subgroup	flop	string
site	OLOGIC_X1Y27	string
type	FD & LD	string
XSTLIB	1	bool

Some properties are read-only and some are user-settable. Properties that map to attributes that can be annotated in UCF or in HDL are generally user-settable through Tcl with the `set_property` command:

```
set_property loc OLOGIC_X1Y27 [get_cell inst_1]
```

## Filtering Based on Properties

The object query `get_*` commands have a common option to filter the query based on any property value attached to the object. This is a powerful capability for the object query commands. For example, to query all cells of primitive type FD do the following:

```
get_cells * -hierarchical -filter "lib_cell == FD"
```

To do more elaborate string filtering, utilize the `=~` operator to do string pattern matching. For example, to query all flip-flop types in the design, do the following:

```
get_cells * -hierarchical -filter "lib_cell =~ FD*"
```

Multiple filter properties can be combined with other property filters with logical OR (`||`) and AND (`&&`) operators to make very powerful searches. To query every cell in the design that is of any flop type and has a placed location constraint:

```
get_cells * -hierarchical -filter {lib_cell =~ FD* && loc != ""}
```

**Note** In the example, the filter option value was wrapped with curly braces `{}` instead of double quotes. This is normal Tcl syntax that prevents command substitution by the interpreter and allows users to pass the empty string (`""`) to the `loc` property.

## Large Lists of Objects

Commands that return more than one object generally return a container that looks and behaves like a native Tcl list. This is a feature of the PlanAhead tool in that it allows dramatic optimization of large collections of Tcl objects handling without the need for special iteration commands like the `foreach_in_collection` command that other tools have implemented. This is handled with the Tcl built-in **foreach** command.

There are a few nuances with respect to large lists, particularly in the log files and the GUI Tcl console. Typically, when you set a Tcl variable to the result of a **get\_\*** command, the entire list is echoed to the console and to the log file. For large lists, this is truncated when printed to the console and log to prevent memory overloading of the buffers in the tool.

What is echoed is the list printed to the log and console is truncated and the last element appears to be “...” in the log and console, however the actual list in the variable assignment is still correct and the last element is not an error. An example of this is querying a single cell versus every cell in the design, which can be large:

```
get_cells inst_1
inst_1
get_cells * -hierarchical
XST_VCC XST_GND error readIngressFifo wbDataForInputReg fifoSelect_0 fifoSelect_1 fifoSelect_2 fifoSelect_3 ...
%set x [get_cells * -hierarchical]
XST_VCC XST_GND error readIngressFifo wbDataForInputReg fifoSelect_0 fifoSelect_1 fifoSelect_2 fifoSelect_3 ...
%lindex $x end
bftClk_BUFGP/bufg
%llength $x
4454
```

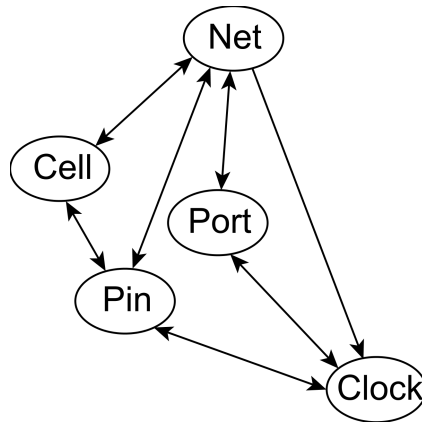
In this example, all four thousand cells were not printed to the console and the list was truncated with a “...” but the actual last element of the list is still correct in the Tcl variable.

## Object Relationships

Related objects can be queried using the **-of** option to the relevant **get\_\*** command. For example, to get a list of pins connected to a cell object, do the following:

```
get_pins -of [get_cells inst_1]
```

The following image shows object types in the PlanAhead tool and their relationships, where an arrow from one object to another object indicates that you can use the **-of** option to the **get\_\*** command to traverse logical connectivity and get Tcl references to any connected object.



## Errors, Warnings, Critical Warnings, and Info Messages

Messages that result from individual commands appear in the log file as well as in the GUI console if it is active. These messages are generally numbered to identify specific issues and are prefixed in the log file with "INFO", "WARNING", "CRITICAL\_Warning", "ERROR" followed by a subsystem identifier and a unique number.

The following example shows an INFO message that appears after reading the timing library.

```
INFO: [HD-LIB 1] Done reading timing library
```

These messages make it easier to search for specific issues in the log file to help to understand the context of operations during command execution.

Generally, when an error occurs in a Tcl command sourced from a Tcl script, further execution of subsequent commands is halted. This is to prevent unrecoverable error conditions. There are Tcl built-ins that allow users to intercept these error conditions, and to choose to continue. Consult any Tcl reference for the catch command for a description of how to handle errors using general Tcl mechanisms.

# *Tcl Commands Listed by Category*

---

## Categories

- [ChipScope](#)
- [FileIO](#)
- [Floorplan](#)
- [GUIControl](#)
- [IPFlow](#)
- [Netlist](#)
- [Object](#)
- [PartialReconfiguration](#)
- [Partition](#)
- [PinPlanning](#)
- [Power](#)
- [Project](#)
- [PropertyAndParameter](#)
- [Report](#)
- [SDC](#)
- [Simulation](#)
- [SysGen](#)
- [Timing](#)
- [ToolLaunch](#)
- [Tools](#)
- [XDC](#)
- [XPS](#)

## ChipScope

- [connect\\_debug\\_port](#)
- [create\\_debug\\_core](#)
- [create\\_debug\\_port](#)
- [delete\\_debug\\_core](#)
- [delete\\_debug\\_port](#)
- [disconnect\\_debug\\_port](#)
- [get\\_debug\\_cores](#)
- [get\\_debug\\_ports](#)
- [implement\\_debug\\_core](#)
- [launch\\_chipscope\\_analyzer](#)
- [read\\_chipscope\\_cdc](#)
- [report\\_debug\\_core](#)
- [write\\_chipscope\\_cdc](#)

## FileIO

- [infer\\_diff\\_pairs](#)
- [read\\_chipscope\\_cdc](#)
- [read\\_csv](#)
- [read\\_edif](#)
- [read\\_pxml](#)
- [read\\_twx](#)
- [read\\_ucf](#)
- [read\\_verilog](#)
- [read\\_vhdl](#)
- [read\\_xdl](#)
- [write\\_chipscope\\_cdc](#)
- [write\\_csv](#)
- [write\\_edif](#)
- [write\\_ibis](#)
- [write\\_timing](#)
- [write\\_ucf](#)
- [write\\_verilog](#)
- [write\\_vhdl](#)
- [write\\_xdc](#)

## Floorplan

- [add\\_cells\\_to\\_pblock](#)
- [create\\_pblock](#)
- [delete\\_pblock](#)
- [delete\\_rpm](#)
- [get\\_pblocks](#)
- [place\\_cell](#)
- [place\\_pblocks](#)
- [remove\\_cells\\_from\\_pblock](#)
- [reset\\_ucf](#)
- [resize\\_pblock](#)
- [swap\\_locs](#)
- [unplace\\_cell](#)

## GUIControl

- [endgroup](#)
- [get\\_selected\\_objects](#)
- [highlight\\_objects](#)
- [mark\\_objects](#)
- [redo](#)
- [select\\_objects](#)
- [start\\_gui](#)
- [startgroup](#)
- [stop\\_gui](#)
- [undo](#)
- [unhighlight\\_objects](#)
- [unmark\\_objects](#)
- [unselect\\_objects](#)

## IPFlow

- [create\\_ip](#)
- [generate\\_target](#)
- [get\\_ipdefs](#)
- [get\\_ips](#)
- [import\\_ip](#)
- [open\\_example\\_project](#)
- [reset\\_target](#)
- [update\\_ip\\_catalog](#)
- [upgrade\\_ip](#)

## Netlist

- [create\\_pin](#)
- [remove\\_pin](#)

## Object

- [filter](#)
- [get\\_bel\\_pins](#)
- [get\\_bels](#)
- [get\\_boards](#)
- [get\\_cells](#)
- [get\\_clock\\_regions](#)
- [get\\_debug\\_cores](#)
- [get\\_debug\\_ports](#)
- [get\\_designs](#)
- [get\\_files](#)
- [get\\_filesets](#)
- [get\\_interfaces](#)
- [get\\_iobanks](#)
- [get\\_ipdefs](#)
- [get\\_ips](#)
- [get\\_lib\\_cells](#)
- [get\\_lib\\_pins](#)
- [get\\_libs](#)
- [get\\_nets](#)
- [get\\_nodes](#)
- [get\\_package\\_pins](#)
- [get\\_parts](#)
- [get\\_pblocks](#)
- [get\\_pins](#)
- [get\\_pips](#)
- [get\\_ports](#)
- [get\\_projects](#)
- [get\\_property](#)
- [get\\_reconfig\\_modules](#)

- [get\\_runs](#)
- [get\\_selected\\_objects](#)
- [get\\_site\\_pins](#)
- [get\\_sites](#)
- [get\\_tiles](#)
- [get\\_wires](#)
- [list\\_property](#)
- [list\\_property\\_value](#)
- [report\\_property](#)
- [reset\\_property](#)
- [set\\_property](#)

## PartialReconfiguration

- [config\\_partition](#)
- [create\\_reconfig\\_module](#)
- [delete\\_reconfig\\_module](#)
- [demote\\_run](#)
- [get\\_reconfig\\_modules](#)
- [load\\_reconfig\\_modules](#)
- [promote\\_run](#)
- [verify\\_config](#)

## Partition

- [config\\_partition](#)
- [demote\\_run](#)
- [promote\\_run](#)

## PinPlanning

- [create\\_interface](#)
- [create\\_port](#)
- [delete\\_interface](#)
- [delete\\_port](#)
- [make\\_diff\\_pair\\_ports](#)
- [place\\_ports](#)
- [set\\_package\\_pin\\_val](#)
- [split\\_diff\\_pair\\_ports](#)

## Power

- [report\\_power](#)
- [reset\\_default\\_switching\\_activity](#)
- [reset\\_operating\\_conditions](#)
- [reset\\_switching\\_activity](#)
- [set\\_operating\\_conditions](#)
- [set\\_power\\_opt](#)
- [set\\_switching\\_activity](#)

## Project

- [add\\_files](#)
- [archive\\_project](#)
- [close\\_design](#)
- [close\\_project](#)
- [create\\_fileset](#)
- [create\\_project](#)
- [create\\_run](#)
- [current\\_fileset](#)
- [current\\_project](#)
- [current\\_run](#)
- [delete\\_fileset](#)
- [delete\\_run](#)
- [find\\_top](#)
- [generate\\_target](#)
- [get\\_boards](#)
- [get\\_files](#)
- [get\\_filesets](#)
- [get\\_ips](#)
- [get\\_projects](#)
- [get\\_runs](#)
- [help](#)
- [import\\_as\\_run](#)
- [import\\_files](#)
- [import\\_ip](#)
- [import\\_synplify](#)
- [import\\_xise](#)
- [import\\_xst](#)
- [launch\\_runs](#)
- [list\\_targets](#)
- [make\\_wrapper](#)
- [open\\_example\\_project](#)
- [open\\_io\\_design](#)

- [open\\_project](#)
- [open\\_rtl\\_design](#)
- [open\\_run](#)
- [refresh\\_design](#)
- [reimport\\_files](#)
- [remove\\_files](#)
- [reorder\\_files](#)
- [reset\\_run](#)
- [reset\\_target](#)
- [save\\_design](#)
- [save\\_design\\_as](#)
- [save\\_project\\_as](#)
- [set\\_speed\\_grade](#)
- [update\\_files](#)
- [wait\\_on\\_run](#)

## PropertyAndParameter

- [create\\_property](#)
- [filter](#)
- [get\\_param](#)
- [get\\_property](#)
- [list\\_param](#)
- [list\\_property](#)
- [list\\_property\\_value](#)
- [report\\_param](#)
- [report\\_property](#)
- [reset\\_param](#)
- [reset\\_property](#)
- [set\\_param](#)
- [set\\_property](#)

## Report

- [create\\_slack\\_histogram](#)
- [create\\_violation](#)
- [delete\\_timing\\_results](#)
- [delete\\_utilization\\_results](#)
- [get\\_msg\\_count](#)
- [get\\_msg\\_limit](#)
- [register\\_drc\\_rule](#)
- [report\\_carry\\_chains](#)
- [report\\_clock\\_interaction](#)
- [report\\_clocks](#)
- [report\\_config\\_timing](#)
- [report\\_control\\_sets](#)
- [report\\_debug\\_core](#)
- [report\\_drc](#)
- [report\\_environment](#)
- [report\\_io](#)
- [report\\_param](#)
- [report\\_power](#)
- [report\\_property](#)
- [report\\_resources](#)
- [report\\_ssn](#)
- [report\\_sso](#)
- [report\\_stats](#)
- [report\\_timing](#)
- [report\\_transformed\\_primitives](#)
- [report\\_user\\_drc\\_rule](#)
- [report\\_utilization](#)
- [reset\\_drc](#)
- [reset\\_msg\\_count](#)
- [reset\\_msg\\_limit](#)
- [reset\\_msg\\_severity](#)
- [reset\\_ssn](#)
- [reset\\_sso](#)
- [reset\\_timing](#)
- [set\\_msg\\_limit](#)
- [set\\_msg\\_severity](#)
- [version](#)

## SDC

- [all\\_clocks](#)
- [all\\_fanin](#)
- [all\\_fanout](#)
- [all\\_inputs](#)
- [all\\_outputs](#)
- [all\\_registers](#)
- [current\\_design](#)
- [current\\_instance](#)
- [get\\_cells](#)
- [get\\_hierarchy\\_separator](#)
- [get\\_lib\\_cells](#)
- [get\\_lib\\_pins](#)
- [get\\_libs](#)
- [get\\_nets](#)
- [get\\_pins](#)
- [get\\_ports](#)
- [report\\_operating\\_conditions](#)
- [reset\\_operating\\_conditions](#)
- [set\\_hierarchy\\_separator](#)
- [set\\_logic\\_unconnected](#)
- [set\\_operating\\_conditions](#)

## Simulation

- [add\\_files](#)
- [create\\_fileset](#)
- [delete\\_fileset](#)
- [import\\_files](#)
- [launch\\_isim](#)
- [launch\\_modelsim](#)
- [remove\\_files](#)
- [write\\_verilog](#)
- [write\\_vhdl](#)

## SysGen

- [create\\_sysgen](#)
- [make\\_wrapper](#)

## Timing

- [report\\_timing](#)
- [reset\\_timing](#)
- [update\\_timing](#)

## ToolLaunch

- [crossprobe\\_fed](#)
- [launch\\_chipscope\\_analyzer](#)
- [launch\\_fpga\\_editor](#)
- [launch\\_impact](#)
- [launch\\_isim](#)
- [launch\\_modelsim](#)
- [launch\\_sdk](#)
- [launch\\_xpa](#)

## Tools

[link\\_design](#)

## XDC

- [add\\_cells\\_to\\_pblock](#)
- [all\\_clocks](#)
- [all\\_cpus](#)
- [all\\_dsps](#)
- [all\\_fanin](#)
- [all\\_fanout](#)
- [all\\_ffs](#)
- [all\\_hsios](#)
- [all\\_inputs](#)
- [all\\_latches](#)
- [all\\_outputs](#)
- [all\\_rams](#)
- [all\\_registers](#)
- [config\\_timing\\_analysis](#)
- [config\\_timing\\_corners](#)
- [config\\_timing\\_pessimism](#)
- [create\\_interface](#)
- [create\\_pblock](#)
- [current\\_design](#)
- [current\\_instance](#)
- [delete\\_interface](#)
- [delete\\_pblock](#)

- `delete_power_results`
- `delete_timing_results`
- `filter`
- `get_cells`
- `get_designs`
- `get_hierarchy_separator`
- `get_interfaces`
- `get_iobanks`
- `get_lib_cells`
- `get_lib_pins`
- `get_libs`
- `get_nets`
- `get_package_pins`
- `get_pblocks`
- `get_pins`
- `get_ports`
- `get_sites`
- `remove_cells_from_pblock`
- `report_default_switching_activity`
- `report_operating_conditions`
- `report_switching_activity`
- `reset_default_switching_activity`
- `reset_operating_conditions`
- `reset_switching_activity`
- `resize_pblock`
- `set_hierarchy_separator`
- `set_logic_unconnected`
- `set_operating_conditions`
- `set_package_pin_val`
- `set_power_opt`
- `set_switching_activity`

## XPS

- `create_xps`
- `export_hardware`
- `generate_target`
- `get_boards`
- `launch_sdk`
- `list_targets`
- `make_wrapper`
- `reset_target`



## Tcl Commands Listed Alphabetically

This chapter contains all SDC and Tcl Commands, arranged alphabetically.

### add\_cells\_to\_pblock

Add cells to a Pblock.

#### Syntax

```
add_cells_to_pblock [-add_primitives] [-clear_locs] [-quiet]
[-verbose] pblock cells ...
```

#### Returns

Nothing

#### Usage

Name	Description
<i>[-add_primitives]</i>	Assign all the primitives of the specified instances to a pblock
<i>[-clear_locs]</i>	Clear instance location constraints
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>pblock</i>	Pblock to add cells to
<i>cells</i>	Cells to add

#### Categories

Floorplan, XDC

#### Description

Adds specified logic instances to a Pblock. Once cells have been added to a Pblock, you can place the Pblocks onto the fabric of the FPGA using the **place\_pblocks** command. Once Pblocks have been automatically placed you can manually move pblocks and resize them using the **resize\_pblock** command.

You can remove instances from the Pblock using the **remove\_cells\_from\_pblock** command.

## Arguments

**-add\_primitives** - Assign all primitives of the specified instances to a Pblock. This lets you specify a block module and automatically assign all of the instances within that module to the specified Pblock.

**-clear\_locs** - Clear instance location constraints for any cells that are already placed. This allows you to reset the LOC constraint for cells when defining new Pblocks for floorplanning purposes. When this option is not specified, any instances with assigned placement will not be unplaced as they are added to the Pblock.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*pblock* - The name of the Pblock to add the specified instances to.

*cells* - One or more cell objects to add to the specified Pblock.

## Examples

The following example creates a Pblock called pb\_cpuEngine, and then adds all of the primitives found in the cpuEngine module, clearing placement constraints for placed instances:

```
create_pblock pb_cpuEngine
add_cells_to_pblock pb_cpuEngine [get_cells cpuEngine] -add_primitives -clear_locs
```

## See Also

- [get\\_pblocks](#)
- [place\\_pblocks](#)
- [remove\\_cells\\_from\\_pblock](#)
- [resize\\_pblock](#)

## add\_files

Add sources to the active fileset.

### Syntax

```
add_files [-fileset arg] [-norecurse] [-scan_for_includes]
          [-quiet] [-verbose] [files ...]
```

### Returns

List of file objects that were added

### Usage

Name	Description
<i>[-fileset]</i>	Fileset name
<i>[-norecurse]</i>	Do not recursively search in specified directories
<i>[-scan_for_includes]</i>	Scan and add any included files found in the fileset's RTL sources
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ files ]</i>	Name of the files and/or directories to add. Must be specified if <i>-scan_for_includes</i> is not used.

### Categories

[Project](#), [Simulation](#)

### Description

Adds one or more source files or the source file contents of one or more directories to the specified fileset.

This command is different from the **import\_files** command, which copies the file into the local project folders as well as adding them to the specified fileset. This command only adds them by reference to the specified fileset.

### Arguments

**-fileset** *name* - (Optional) The fileset to which the specified source files should be added. An error is returned if the specified fileset does not exist. If no fileset is specified the files will be added to the source fileset by default.

**-norecurse** - (Optional) Do not recurse through subdirectories of any specified directories. Without this argument, the tool will search through any subdirectories for additional source files that can be added to a project.

**-scan\_for\_includes** - (Optional) Scan Verilog source files for any **'include** statements and add these referenced files to the specified fileset. By default, **'include** files are not added to the fileset.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*files* - One or more file names or directory names to be added to the fileset. If a directory name is specified, all valid source files found in the directory, and in subdirectories of the directory, will be added.

**Note** If the path is not specified as part of the file name, the tool will search for the specified file in the current working directory and then in the directory from which the tool was launched.

## Examples

The following example adds a file called `rtl.v` to the current project:

```
add_files rtl.v
```

In the preceding example the tool will look for the `rtl.v` file in the current working directory since no file path is specified, and the file will be added to the source fileset as a default since no fileset is specified.

The following example adds a file called **top.ucf** to the **constrs\_1** constraint fileset, as well as any appropriate source files found in the **project\_1** directory, and its subdirectories:

```
add_files -fileset constrs_1 -quiet c:/Design/top.ucf c:/Design/project_1
```

In addition, the tool will ignore any command line errors because the **-quiet** argument was specified.

If the **-norecurse** option had been specified then only constraint files found in the **project\_1** directory would have been added, but subdirectories would not be searched.

The following example adds an existing IP core file to the current project:

```
add_files -norecurse C:/Data/ip/c_addsub_v11_0_0.xci
```

**Note** Use the **import\_ip** command to import the IP file into the local project folders

## See Also

- [import\\_files](#)
- [import\\_ip](#)

## all\_clocks

Get a list of all clocks in the current design.

### Syntax

```
all_clocks [-quiet] [-verbose]
```

### Returns

List of clock objects

### Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

### Categories

[SDC](#), [XDC](#)

### Description

Returns a list of all clocks that have been declared in the current design. To get a list of specific clocks in the design, use the **get\_clocks** command.

Clocks can be defined by using the **create\_clock** or **create\_generated\_clock** commands.

### Arguments

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

### Examples

The following example shows all clocks in the sample CPU netlist project:

```
% all_clocks
cpuClk wbClk usbClk phy_clk_pad_0_i phy_clk_pad_1_i fftClk
```

The following example shows how the returned list can be passed to another command:

```
% set_propagated_clock [all_clocks]
```

This will apply the **set\_propagated\_clock** command to all the clocks in the design.

## See Also

- [create\\_clock](#)
- [create\\_generated\\_clock](#)
- [get\\_clocks](#)
- [set\\_propagated\\_clock](#)

## all\_cpus

Get a list of cpu cells in the current design.

### Syntax

```
all_cpus [-quiet] [-verbose]
```

### Returns

List of cpu cell objects

### Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

### Categories

[XDC](#)

### Description

Returns a list of all CPU cell objects in the current design. Creates a list of all the CPU cell objects that have been declared in the current design.

**Note** This command returns a list of CPU cell objects

### Arguments

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the `set_msg_limit` command.

### Examples

The following example returns all CPU objects in the current design:

```
all_cpus
```

The following example shows how the list returned can be passed to another command:

```
set_false_path -from [all_cpus] -to [all_registers]
```

## See Also

- [all\\_dsps](#)
- [all\\_hsios](#)
- [all\\_registers](#)
- [set\\_false\\_path](#)

## all\_dsps

Get a list of dsp cells in the current design.

### Syntax

```
all_dsps [-quiet] [-verbose]
```

### Returns

List of dsp cell objects

### Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

### Categories

[XDC](#)

### Description

Returns a list of all DSP cell objects that have been declared in the current design.

### Arguments

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

### Examples

The following example returns a list of all DSPs defined in the current design:

```
all_dsps
```

The following example shows how the list returned can be passed to another command:

```
set_false_path -from [all_dsps] -to [all_registers]
```

### See Also

- [all\\_cpus](#)
- [all\\_hsios](#)
- [all\\_registers](#)
- [set\\_false\\_path](#)

## all\_fanin

Get a list of pins or cells in fanin of specified sinks.

### Syntax

```
all_fanin [-startpoints_only] [-flat] [-only_cells]
[-levels arg] [-pin_levels arg] [-trace_arcs arg] [-quiet]
[-verbose] to
```

### Returns

List of cell or pin objects

### Usage

Name	Description
<i>[-startpoints_only]</i>	Find only the timing startpoints
<i>[-flat]</i>	Hierarchy is ignored
<i>[-only_cells]</i>	Only cells
<i>[-levels]</i>	Maximum number of cell levels to traverse: Value = 0 Default: 0
<i>[-pin_levels]</i>	Maximum number of pin levels to traverse: Value = 0 Default: 0
<i>[-trace_arcs]</i>	Type of network arcs to trace: Values: timing, enabled, all
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>to</i>	List of sink pins, ports, or nets

### Categories

[SDC](#), [XDC](#)

### Description

Returns a list of port, pin or cell objects in the fan-in of the specified sinks.

### Arguments

**-startpoints\_only** - (Optional) Find only the timing start points.

**-flat** - (Optional) Ignore the hierarchy of the design.

**-only\_cells** - (Optional) Return only the cell objects which are in the fan-in path of the specified sinks. Do not return pins or ports.

**-levels** *value* - (Optional) Maximum number of cell levels to traverse. The default value is 0.

**-pin\_levels** *value* - (Optional, Default Value of 0) Maximum number of pin levels to traverse. The default value is 0.

**-trace\_arcs** *value* - Type of network arcs to trace. Valid values are "timing", "enabled", and "all"

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

**to** - (Optional) The pins, ports, or nets from which you want the fan-in objects reported.

## Examples

The following example lists the timing fan-in of the DAT port:

```
all_fanin DAT
```

The following example traces back from the clock pin of the specified flip flop, through an MMCM, to the top level port called sysClk:

```
all_fanin -to [get_pins fftEngine/control_reg_reg[17]/C] -startpoints_only \  
-flat sysClk
```

## See Also

[all\\_fanout](#)

## all\_fanout

Get a list of pins or cells in fanout of specified sources.

### Syntax

```
all_fanout [-endpoints_only] [-flat] [-only_cells] [-levels arg]  
[-pin_levels arg] [-trace_arcs arg] [-quiet] [-verbose] from
```

### Returns

List of cell or pin objects

### Usage

Name	Description
<i>[-endpoints_only]</i>	Find only the timing endpoints
<i>[-flat]</i>	Hierarchy is ignored
<i>[-only_cells]</i>	Only cells
<i>[-levels]</i>	Maximum number of cell levels to traverse: Value = 0 Default: 0
<i>[-pin_levels]</i>	Maximum number of pin levels to traverse: Value = 0 Default: 0
<i>[-trace_arcs]</i>	Type of network arcs to trace: Values: timing, enabled, all
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>from</i>	List of source pins, ports, or nets

### Categories

SDC, XDC

### Description

Returns a list of port, pin, or cell objects in the fanout of the specified sources.

### Arguments

**-endpoints\_only** - (Optional) Find only the timing endpoints.

**-flat** - (Optional) Ignore the hierarchy of the design.

**-only\_cells** - (Optional) Return only the cell objects in the fanout path of the specified sources.

**-levels** *value* - (Optional) Maximum number of cell levels to traverse. The default value is 0.

**-pin\_levels** *value* - (Optional) Maximum number of pin levels to traverse. The default value is 0.

**-trace\_arcs** *value* - Type of network arcs to trace. Valid values are "timing", "enabled", and "all"

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*from* - (Optional) The source ports, pins, or nets from which to list the objects in the fanout path.

## Examples

The following example lists the timing fanout of port DAT in the current design:

```
all_fanout DAT
```

## See Also

[all\\_fanin](#)

## all\_ffs

Get a list of flip flop cells in the current design.

### Syntax

```
all_ffs [-quiet] [-verbose]
```

### Returns

List of flip flop cell objects

### Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

### Categories

[XDC](#)

### Description

Returns a list of all flip flop instances in the current design.

### Arguments

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

### Examples

The following example reports the currently assigned properties on the specified flop:

```
report_property [lindex [all_ffs] 2 ]
```

### See Also

- [all\\_registers](#)
- [report\\_property](#)

## all\_hsios

Get a list of hsio cells in the current design.

### Syntax

```
all_hsios [-quiet] [-verbose]
```

### Returns

List of hsio cell objects

### Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

### Categories

[XDC](#)

### Description

Returns a list of all High Speed IO (HSIO) cell objects that have been declared in the current design. These HSIO cell objects can be assigned to a variable or passed into another command.

### Arguments

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the `set_msg_limit` command.

### Examples

The following example returns all HSIO objects in the current design:

```
all_hsios
```

The following example shows how the list returned can be passed to another command:

```
set_false_path -from [all_hsios] -to [all_registers]
```

### See Also

- [all\\_cpus](#)
- [all\\_dsps](#)
- [all\\_registers](#)
- [set\\_false\\_path](#)

## all\_inputs

Get a list of all input ports in the current design.

### Syntax

```
all_inputs [-quiet] [-verbose]
```

### Returns

List of port objects

### Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

### Categories

[SDC](#), [XDC](#)

### Description

Returns a list of all input port objects in the current design.

### Arguments

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

### Examples

The following example returns all input ports in the current design:

```
all_inputs
```

The following example shows how the list returned can be passed to another command:

```
set_input_delay 5 -clock REFCLK [all_inputs]
```

### See Also

- [all\\_outputs](#)
- [set\\_input\\_delay](#)

## all\_latches

Get a list of all latch cells in the current design.

### Syntax

```
all_latches [-quiet] [-verbose]
```

### Returns

List of latch cell objects

### Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

### Categories

[XDC](#)

### Description

Returns a list of all latches that have been declared in the current design.

### Arguments

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the `set_msg_limit` command.

### Examples

The following example returns a list of all latches in the current design:

```
all_latches
```

The following example shows how the list returned can be passed to another command:

```
set_false_path -from [all_mults] -to [all_latches]
```

### See Also

- [all\\_ffs](#)
- [all\\_registers](#)
- [set\\_false\\_path](#)

## all\_outputs

Get a list of all output ports in the current design.

### Syntax

```
all_outputs [-quiet] [-verbose]
```

### Returns

List of port objects

### Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

### Categories

[SDC](#), [XDC](#)

### Description

Returns a list of all output port objects that have been declared in the current design.

### Arguments

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

### Examples

The following example returns all the output ports in the current design:

```
all_outputs
```

The following example sets the output delay for all outputs in the design:

```
set_output_delay 5 -clock REFCLK [all_outputs]
```

### See Also

- [all\\_inputs](#)
- [set\\_output\\_delay](#)

## all\_rams

Get a list of ram cells in the current design.

### Syntax

```
all_rams [-quiet] [-verbose]
```

### Returns

List of ram cell objects

### Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

### Categories

[XDC](#)

### Description

Returns a list of all the RAM cell objects that have been declared in the current design. These RAM cell objects can be assigned to a variable or passed into another command.

### Arguments

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the `set_msg_limit` command.

### Examples

The following example returns all RAM objects in the current design:

```
all_rams
```

### See Also

- [all\\_cpus](#)
- [all\\_dsps](#)
- [all\\_hsios](#)
- [all\\_registers](#)

## all\_registers

Get a list of register cells or pins in the current design.

### Syntax

```
all_registers [-clock args] [-rise_clock args]
[-fall_clock args] [-cells] [-data_pins] [-clock_pins]
[-async_pins] [-output_pins] [-level_sensitive] [-edge_triggered]
[-quiet] [-verbose]
```

### Returns

List of cell or pin objects

### Usage

Name	Description
<code>[-clock]</code>	Consider registers of this clock
<code>[-rise_clock]</code>	Consider registers triggered by clock rising edge
<code>[-fall_clock]</code>	Consider registers triggered by clock falling edge
<code>[-cells]</code>	Return list of cells (default)
<code>[-data_pins]</code>	Return list of register data pins
<code>[-clock_pins]</code>	Return list of register clock pins
<code>[-async_pins]</code>	Return list of async preset/clear pins
<code>[-output_pins]</code>	Return list of register output pins
<code>[-level_sensitive]</code>	Only consider level-sensitive latches
<code>[-edge_triggered]</code>	Only consider edge-triggered flip-flops
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

### Categories

[SDC](#), [XDC](#)

### Description

Returns a collection of sequential register cells or register pins in the current design.

The list of returned objects can be limited by the use of the arguments described below. You can limit the list of registers returned to a specific clock or clocks, or to registers triggered by the rising or falling edge of a specified clock.

You can also get a list of the pins of collected registers instead of the register objects by specifying one or more of the pin arguments.

### Arguments

**-cells** - (Optional) Return a list of register cell objects as opposed to a list of pin objects. This is the default behavior of the command.

**-clock** *args* - (Optional) Return a list of all registers whose clock pins are in the fanout of the specified clock.

**-rise\_clock** *args* - (Optional) Return a list of registers triggered by the rising edge of the specified clocks.

**-fall\_clock** *args* - (Optional) Return a list of registers triggered by the falling edge of the specified clocks.

**Note** Do not combine **-clock**, **-rise\_clock**, and **-fall\_clock** in the same command.

**-level\_sensitive** - (Optional) Return a list of the level-sensitive registers or latches.

**-edge\_triggered** - (Optional) Return a list of the edge-triggered registers or flip-flops.

**-data\_pins** - (Optional) Return a list of data pins of all registers in the design, or of the registers that meet the search requirement.

**-clock\_pins** - (Optional) Return a collection of clock pins of the registers that meet the search requirement.

**-async\_pins** - (Optional) Limit the search to asynchronous pins of the registers that meet the search requirement.

**-output\_pins** - (Optional) Return a collection of output pins of the registers that meet the search requirement.

**Note** Use \*\_pins arguments separately. If you specify multiple arguments, only one will be used, in the following order of precedence: **-data\_pins**, **-clock\_pins**, **-async\_pins**, **-output\_pins**.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

## Examples

The following example returns a list of registers that are triggered by the falling edge of any clock in the design:

```
all_registers -fall_clock [all_clocks]
```

The following example shows how the list returned can be passed to another command:

```
set_min_delay 2.0 -to [all_registers -clock CCLK -data_pins]
```

## See Also

- [all\\_clocks](#)
- [set\\_msg\\_limit](#)
- [set\\_min\\_delay](#)

## archive\_project

Archive the current project.

### Syntax

```
archive_project [-force] [-exclude_run_results] [-quiet]
[-verbose] [file]
```

### Returns

True

### Usage

Name	Description
<i>[-force]</i>	Overwrite existing archived file
<i>[-exclude_run_results]</i>	Exclude run results from the archive
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ file ]</i>	Name of the archive file

### Categories

Project

### Description

Archives a project to store as backup, or to encapsulate the design and send it to a remote site. The tool parses the hierarchy of the design, copies the required source files, include files, and remote files from the library directories, copies the constraint files, copies the results of the various synthesis, simulation, and implementation runs, and then creates a ZIP file of the project.

### Arguments

**-force** - Overwrite an existing ZIP file of the same name. If the ZIP file exists, the PlanAhead tool will return an error unless the **-force** argument is specified.

**-exclude\_run\_results** - Exclude the results of any synthesis or implementation runs. This command can greatly reduce the size of a project archive.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*file* - The name of the ZIP file to be created by the **archive\_project** command. If *file* is not specified, a ZIP file with the same name as the project will be created.

## Examples

The following command archives the current project:

```
archive_project
```

**Note** The project archive will be named *project\_name.zip* because no file name is specified.

The following example specifies `project_3` as the current project, and then archives that project into a file called `proj3.zip`:

```
current_project project_3  
archive_project -force -exclude_run_results proj3.zip
```

**Note** The use of the **-force** argument causes the PlanAhead tool to overwrite the `proj3.zip` file if one exists. The use of the **-exclude\_run\_results** argument causes the PlanAhead tool to leave any results from synthesis or implementation runs out of the archive. The various runs defined in the project will be included in the archive, but not any of the results.

## See Also

[current\\_project](#)

## close\_design

Close the current design.

### Syntax

```
close_design [-quiet] [-verbose]
```

### Returns

Nothing

### Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

### Categories

Project

### Description

Closes the currently active design. If the design has been modified, you will not be prompted to save the design prior to closing. You will need to run **save\_design** or **save\_design\_as** to save any changes made to the design before using the **close\_design** command.

### Arguments

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

### Examples

The following example closes the current design:

```
close_design
```

**Note** If multiple designs are open, you can specify the current design with the **current\_design** command prior to using **close\_design**.

The following example sets the current design, then closes it:

```
current_design rtl_1
close_design
```

**current\_design** sets **rtl\_1** as the active design, then the **close\_design** command closes it.

## See Also

- [current\\_design](#)
- [save\\_design](#)
- [save\\_design\\_as](#)

## close\_project

Close current opened project.

### Syntax

```
close_project [-delete] [-quiet] [-verbose]
```

### Returns

Nothing

### Usage

Name	Description
<code>[-delete]</code>	Delete the project from disk also
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

### Categories

[Project](#)

### Description

Closes the current open project in the PlanAhead tool.

### Arguments

**-delete** - Delete the project data from the hard disk after closing the project.

**Note** Use this argument with caution. You will not be prompted to confirm the deletion if you use this argument.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

### Examples

The following command closes the active project:

```
close_project
```

This example closes the current project. If you have multiple projects open, the **close\_project** command applies to the current project which can be defined with the **current\_project** command.

The following example sets `project_1` as the active project, and then closes the active project and deletes it from the computer hard disk:

```
current_project project_1  
close_project -delete
```

**Note** Use the **-delete** argument with caution. You will not be prompted to confirm the deletion if you use this argument.

## See Also

[current\\_project](#)

## config\_partition

Set module variants and states on a given run.

### Syntax

```
config_partition [-cell arg] [-reconfig_module arg] [-import]
[-implement] [-import_dir arg] [-preservation arg] [-quiet]
[-verbose] run
```

### Returns

Nothing

### Usage

Name	Description
<i>[-cell]</i>	Partition instance to configure in the given run. In order to modify top Partition do not specify this option.
<i>[-reconfig_module]</i>	Reconfigurable Module variant to apply to this instance in this run
<i>[-import]</i>	Set this instance (or static logic) to 'import' action for this run
<i>[-implement]</i>	Set this instance (or static logic) to 'implement' action for this run
<i>[-import_dir]</i>	Directory from which to import this previously implemented module
<i>[-preservation]</i>	set the preservation level for the Partition. Values: routing, placement, synthesis Default: routing
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>run</i>	Run to be modified

### Categories

[PartialReconfiguration](#), [Partition](#)

### Description

Configure a partition prior to synthesis or implementation.

Specify the partition module as a cell or as the top module, the action to take for the specific module, the path to import partition data from, and the level of partition data to preserve.

### Arguments

**-cell** - (Optional) Specify the cell name of the partition to configure.

**Note** If this option is not specified, the **config\_partition** command applies to the top partition in the design.

**-reconfig\_module** *name* - (Optional) The Reconfigurable Module variant to configure.

**-import** - (Optional) Import data for the specified partition from the **-import\_dir** during synthesis or implementation to preserve prior results.

**-implement** - (Optional) Discard prior results and reimplement the specified partition during synthesis or implementation. This treats the partition like any other hierarchical module in the design.

**Note** **-implement** and **-import** are mutually exclusive arguments.

**-import\_dir** *name* - (Optional) The directory to import partition data from.

**Note** If the path to the directory is not specified, the tool will look in your home directory for the partition data:

- For Windows: %APPDATA%/Xilinx/PlanAhead
- For Linux: \$HOME/.Xilinx/PlanAhead

**-preservation** *level* - (Optional) The preservation level for the imported partitions. Valid values are **routing**, **placement**, and **synthesis**. The default is to preserve routing.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*run* - - The run to be configured.

## Examples

The following example creates a partition on the module `sub_block_inst` and configures the partition to be implemented during `impl_1` run:

```
config_partition -run synth_1 -cell usbEngine0 -import -import_dir \  
C:/Data/DP_RTL/synth1 -preservation placement
```

## See Also

[promote\\_run](#)

## config\_timing\_analysis

Configure timing analysis general settings.

### Syntax

```
config_timing_analysis
[-disable_paths_between_unrelated_ucf_clocks arg]
[-enable_input_delay_default_clock arg]
[-enable_preset_clear_arcs arg] [-quiet] [-verbose]
```

### Returns

Nothing

### Usage

Name	Description
<i>[-disable_paths_between_unrelated_ucf_clocks]</i>	Disable timing paths between unrelated UCF clocks: Values: true, false; This option is not supported for SDC constraints
<i>[-enable_input_delay_default_clock]</i>	Launch SDC unlocked input delays from an internally defined clock: Values: true, false; This option is not supported for UCF constraints
<i>[-enable_preset_clear_arcs]</i>	Time paths through asynchronous preset or clear timing arcs: true, false;
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution

### Categories

XDC

### Description

This command configures general features of timing analysis.

**Note** This command operates silently and does not return direct feedback of its operation.

### Arguments

**-disable\_paths\_between\_unrelated\_ucf\_clocks** [ true | false ] - (Optional) This argument has the effect of specifying a false path between unrelated UCF clocks, disabling cross-clock domain analysis. The valid values are true or false, with a default setting of false.

**Note** This argument is for UCF based designs

**-enable\_input\_delay\_default\_clock** [ true | false ] - (Optional) Launch unlocked input delays from an internally defined clock for timing analysis. The valid values are true or false, with a default setting of false.

**Note** This argument applies to SDC based designs only

**-enable\_preset\_clear\_arcs** [ **true** | **false** ] - (Optional) Time paths through asynchronous preset or clear timing arcs. The valid values are true or false, with a default setting of false.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

## Examples

The following example ignores unrelated clocks in UCF designs:

```
config_timing_analysis -disable_paths_between_unrelated_ucf_clocks true
```

## See Also

- [config\\_timing\\_corners](#)
- [config\\_timing\\_pessimism](#)
- [report\\_timing](#)

## config\_timing\_corners

Configure single / multi corner timing analysis settings.

### Syntax

```
config_timing_corners [-corner arg] [-delay_type arg] [-setup]
[-hold] [-quiet] [-verbose]
```

### Returns

Nothing

### Usage

Name	Description
<code>[-corner]</code>	Name of the timing corner to be modified : Values: Slow, Fast
<code>[-delay_type]</code>	Type of path delays to be analysed for specified timing corner: Values: none, max, min, min_max
<code>[-setup]</code>	Enable timing corner for setup analysis (equivalent to <code>-delay_type max</code> )
<code>[-hold]</code>	Enable timing corner for hold analysis (equivalent to <code>-delay_type min</code> )
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

### Categories

XDC

### Description

This command configures the Slow and Fast timing corners for single or multi-corner timing analysis.

**Note** This command operates silently and does not return direct feedback of its operation.

### Arguments

**-corner** [ **Slow** | **Fast** ] - (Optional) Specifies the name of the timing corner to be configured. Valid values are "Slow" and "Fast".

**Note** The names of the corners are case sensitive.

**-delay\_type** *value* - (Optional) Specify the type of path delays to be analysed for the specified timing corner. Valid values are "max", "min" and "min\_max".

**-setup** - (Optional) Specifies setup analysis for the specified timing corner. This is the same as **-delay\_type max**.

**-hold** - (Optional) Specifies hold analysis for the timing corner. This is the same as **-delay\_type min**.

**Note** You can specify both **-setup** and **-hold** which is the same as **-delay\_type min\_max**.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

## Examples

The following example configures the Slow timing corner for both setup and hold analysis:

```
config_timing_corners -corner Slow -setup -hold  
config_timing_corners -corner Slow -delay_type min_max
```

**Note** The two preceding examples have the same effect.

The following example configures the Fast corner for min delay analysis:

```
config_timing_corners -corner Fast -delay_type min
```

## See Also

- [config\\_timing\\_analysis](#)
- [config\\_timing\\_pessimism](#)
- [report\\_timing](#)

## config\_timing\_pessimism

Configure timing analysis common node pessimism removal settings.

### Syntax

```
config_timing_pessimism [-enable] [-disable] [-transition arg]
                        [-threshold arg] [-common_node arg] [-quiet] [-verbose]
```

### Returns

Nothing

### Usage

Name	Description
<code>[-enable]</code>	Enable common node pessimism removal
<code>[-disable]</code>	Disable common node pessimism removal
<code>[-transition]</code>	Remove pessimism for specified transitions; Values: any_transition, same_transition
<code>[-threshold]</code>	Common node pessimism threshold Default: 0
<code>[-common_node]</code>	Perform pessimism removal to common node in routing network. Values: on, off
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

### Categories

[XDC](#)

### Description

Enables or disables clock reconvergence pessimism removal (CRPR) during timing analysis.

Clock reconvergence pessimism can lead to overly pessimistic timing results including false hold-time violations. To avoid this, CRPR is enabled by default.

**Note** This command operates silently and does not return direct feedback of its operation.

### Arguments

- enable** - (Optional) Enable clock reconvergence pessimism removal (CRPR).
- disable** - (Optional) Disable clock reconvergence pessimism removal (CRPR).
- transition arg** - (Optional) Remove pessimism for the specified transitions. Valid values are any\_transition or same\_transition. The default setting is for any\_transition.
- threshold arg** - (Optional) Common node pessimism threshold.
- common\_node [on, off]** - (Optional) Perform CRPR to common node in routing network.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

## Examples

The following example enables CRPR:

```
config_timing_pessimism -common_node on -enable
```

## See Also

[report\\_timing](#)

## connect\_debug\_port

Connect nets and pins to debug port channels.

### Syntax

```
connect_debug_port [-channel_start_index arg] [-quiet]
[-verbose] port nets ...
```

### Returns

Nothing

### Usage

Name	Description
<i>[-channel_start_index]</i>	Connect nets starting at channel index
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>port</i>	Debug port name
<i>nets</i>	List of nets or pins

### Categories

[ChipScope](#)

### Description

Connects a signal from the Netlist design to a port on a ChipScope debug core. The signal can either be connected to a specific channel index on the port, or simply connected to an available channel on the port.

If you try to connect too many signals to a port, or there are not enough channels to support the connection, the tool will return an error.

Additional ports can be added to a debug core through the use of the `create_debug_port` command, and you can increase the available channels on an existing port with the `set_property port_width` command. See the examples below.

You can disconnect signals from ports using the `disconnect_debug_port` command.

When the ChipScope debug core has been defined and connected, you can implement the debug core as a block for inclusion in the Netlist Design. Use the `implement_debug_core` command to use CoreGen to implement the core.

### Arguments

**-channel\_start\_index** *arg* - The channel index to use for the connection. If more than one signal has been specified, this is the channel index where connections will start to be added. Channel indexes are numbered starting at 0.

**Note** If this argument is not specified, the tool will place connections on the first available channel index.

*port* - The name of the port to connect signals to. The port must be referenced by the `core_name/port_name`.

*nets* - A list of one or more net names from the Netlist Design to connect to the specified debug port.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

## Examples

The following example creates a new TRIG port on the myCore debug core, increases the port\_width of the port in order to prepare it to receive the number of signals to be connected, then connects the signals to the port starting at the third channel position (index 2).

```
create_debug_port myCore TRIG
set_property port_width 8 [get_debug_ports myCore/TRIG0]
connect_debug_port myCore/TRIG0 [get_nets [list m0_ack_o m0_cyc_i m0_err_o \
m0_rty_o m0_stb_i m0_we_i ]] -channel_start_index 2
```

**Note** If you specify too many nets to connect to the available channels on the port, PlanAhead will return an error and will not connect the ports.

## See Also

- [create\\_debug\\_port](#)
- [disconnect\\_debug\\_port](#)
- [get\\_debug\\_ports](#)
- [get\\_nets](#)
- [implement\\_debug\\_core](#)
- [set\\_property](#)

## create\_debug\_core

Create a new ChipScope debug core.

### Syntax

```
create_debug_core [-quiet] [-verbose] name type
```

### Returns

New debug\_core object

### Usage

Name	Description
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>name</i>	Name of the new debug core instance
<i>type</i>	Type of the new debug core

### Categories

ChipScope

### Description

Defines a new ChipScope debug core to be added to an open Netlist Design in the current project. The debug core defines ports for connecting nets to for debug purposes.

**Note** A debug core can only be added to an open Netlist Design in the tool.

The default core that is created includes a CLK port and a trigger (TRIG) port. The CLK port only supports one clock signal, and so you must create a separate debug core for each clock domain.

Once the core is created you can add new ports to the debug core with the create\_debug\_port command, and connect signals to the ports using the connect\_debug\_port command.

### Arguments

*name* - The name of the ChipScope debug core to add to the project.

*type* - The ChipScope debug core to insert. Currently only the chipscope\_ila\_v1 type is supported in the tool. The ILA debug core simply adds another load onto a connected net without otherwise altering it. Refer to the *ChipScope Pro Software and Cores User Guide* (ug029) for more information on debug core types and purpose.

**Note** When the ILA core is added to the project, the tool also adds an ICON controller core as a container for one or more ILA cores. However, you cannot directly add an ICON core to the project.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

## Examples

The following example opens the Netlist Design, and creates a new ChipScope debug core:

```
open_netlist_design -name netlist_1
create_debug_core myCore chipscope_ila_v1
```

**Note** Currently the chipscope\_ila\_v1 is the only type of core supported by the tool.

The following example creates a new debug core called myCore and returns the properties of the newly created core:

```
report_property [create_debug_core myCore chipscope_ila_v1]
```

The properties of the debug core can be customized by using the **set\_property** command as in the following example:

```
set_property enable_storage_qualification false [get_debug_cores myCore]
```

## See Also

- [connect\\_debug\\_port](#)
- [create\\_debug\\_port](#)
- [delete\\_debug\\_core](#)
- [get\\_debug\\_cores](#)
- [implement\\_debug\\_core](#)
- [read\\_chipscope\\_cdc](#)
- [report\\_debug\\_core](#)
- [report\\_property](#)
- [set\\_property](#)
- [write\\_chipscope\\_cdc](#)

## create\_debug\_port

Create a new ChipScope debug port.

### Syntax

```
create_debug_port [-quiet] [-verbose] name type
```

### Returns

New debug\_port object

### Usage

Name	Description
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>name</i>	Name of the debug core instance
<i>type</i>	Type of the new debug port

### Categories

[ChipScope](#)

### Description

Defines a new port to be added to an existing ChipScope debug core. The port provides connection points to a debug core to attach nets from the design for debug purposes.

When a new debug core is created using the `create_debug_core` command, it includes a CLK and trigger (TRIG) port by default. However, you can also add DATA and trigger\_output (TRIG\_OUT) ports to the debug core as well as additional TRIG ports.

A port can have one or more connection points to support one or more nets to debug. As a default new ports are defined as having a width of 1, allowing only one net to be attached. You can change the port width of TRIG and DATA ports to support multiple signals using the **set\_property port\_width** command (see Examples).

**Note** CLK and TRIG\_OUT ports can only have a width of 1.

You can connect signals to ports using the **connect\_debug\_port** command, and disconnect signals with the **disconnect\_debug\_port** command.

### Arguments

*name* - The name of the ChipScope debug core to add the new port to. The debug core must already exist in the project having been created with `create_debug_port` or imported with `read_chipscope_cdc`.

*type* - The type of debug port to insert. There are four port types supported: CLK, DATA, TRIG, and TRIG\_OUT. Refer to the *ChipScope Pro Software and Cores User Guide* (ug029) for more information on port types and purpose.

**Note** Each ILA debug core can have only one CLK, DATA, and TRIG\_OUT port. However, you can create multiple trigger (TRIG) ports.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

## Examples

The following example creates a new ChipScope debug core, and then adds a DATA port to that core:

```
create_debug_core myCore chipscope_ila_v1
create_debug_port myCore DATA
```

The following example creates a new port on the myCore debug core, and then sets the port width to 8, and begins connecting signals to the port:

```
create_debug_port myCore TRIG
set_property PORT_WIDTH 8 [get_debug_ports myCore/TRIG0]
connect_debug_port -channel_start_index 1 myCore/TRIG0 {m1_cyc_i \
    m1_ack_o m1_err_o m1_rty_o}
```

**Note** The debug core is referenced by its name, and the debug port is referenced by the core\_name/port\_name.

## See Also

- [connect\\_debug\\_port](#)
- [create\\_debug\\_core](#)
- [disconnect\\_debug\\_port](#)
- [read\\_chipscope\\_cdc](#)
- [set\\_property](#)

## create\_fileset

Create a new fileset.

### Syntax

```
create_fileset [-constrset] [-simset] [-quiet] [-verbose] name
```

### Returns

New fileset object

### Usage

Name	Description
<i>[-constrset]</i>	Create fileset as constraints fileset (default)
<i>[-simset]</i>	Create fileset as simulation source fileset
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>name</i>	Name of the fileset to be create

### Categories

[Project](#), [Simulation](#)

### Description

Defines a new fileset within your project.

A fileset is a collection of files with a specific function within the project. One or more constraint files is a constraint set (**-constrset**); one or more simulation test benches is a simulation set (**-simset**). Only one fileset option can be specified when using the **create\_fileset** command. As a default, the PlanAhead tool will create a constraint fileset if the type is not specified.

The **create\_fileset** command returns the name of the newly created fileset, or will return an error message unless the **-quiet** argument has been specified.

### Arguments

**-constrset** - (Optional) Creates a constraint set to hold one or more constraint files. This is the default fileset created if neither the **-constrset** or **-simset** argument is specified.

**-simset** - (Optional) Create a simulation fileset to hold one or more simulation source files. You can only specify one type of fileset argument, either **-constrset** or **-simset**. You will get an error if both are specified.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*name* - The name of the fileset to be created.

## Examples

The following example creates a new constraint file set named `constraints2`:

```
create_fileset -constrset -quiet constraints2
```

**Note** With **-quiet** specified, the PlanAhead tool will not return anything if it encounters an error in trying to create the specified fileset.

The following example creates a new simulation fileset named `sim_1`:

```
create_fileset -simset sim_1
```

Files can be added to the newly created fileset using the **add\_files** command.

## See Also

- [add\\_files](#)
- [current\\_fileset](#)

## create\_interface

Create a new I/O port interface.

### Syntax

```
create_interface [-parent arg] [-quiet] [-verbose] name
```

### Returns

New interface object

### Usage

Name	Description
<i>[-parent]</i>	Assign new interface to this parent interface
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>name</i>	Name for new I/O port interface

### Categories

[XDC](#), [PinPlanning](#)

### Description

Creates a new interface for grouping scalar or differential I/O ports.

### Arguments

**-parent** *arg* - (Optional) Assign the new interface to the specified parent interface.

**Note** If the specified parent interface does not exist, an error will be returned.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*name* - The name of the I/O port interface to create.

### Examples

Create a new USB interface:

```
create_interface USB0
```

Create an Ethernet interface within the specified parent interface:

```
create_interface -parent Top_Int ENET0
```

## See Also

- [delete\\_interface](#)
- [create\\_port](#)
- [delete\\_port](#)
- [make\\_diff\\_pair\\_ports](#)
- [place\\_ports](#)
- [set\\_package\\_pin\\_val](#)
- [split\\_diff\\_pair\\_ports](#)

## create\_ip

Create an instance of a configurable IP and add it to the fileset.

### Syntax

```
create_ip [-vlnv arg] -module_name arg [-dir arg] [-vendor arg]
[-library arg] [-name arg] [-version arg] [-quiet] [-verbose]
```

### Returns

List of file objects that were added

### Usage

Name	Description
<code>[-vlnv]</code>	VLNV string for the Catalog IP from which the new IP will be created
<code>-module_name</code>	Name for the new IP that will be added to the project
<code>[-dir]</code>	Directory path for remote IP to be created and managed outside the project
<code>[-vendor]</code>	IP Vendor name
<code>[-library]</code>	IP Library name
<code>[-name]</code>	IP Name
<code>[-version]</code>	IP Version
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

### Categories

[IPFlow](#)

### Description

This command creates an XCI file for a configurable IP core from the IP catalog, and adds it to the source files of the current project. This creates an IP source object which must be instantiated into the HDL design to create an instance of the IP core in the netlist.

For multiple instances of the same core, simply instantiate the core module into the HDL design as many times as needed. However, to use the same IP core with different customizations, use the **create\_ip** command to create separate IP source objects.

The **create\_ip** command is used to import IP cores from the current IP catalog. Use the **import\_ip** command to read existing XCI and XCO files directly, without having to add IP to a catalog.

This command returns a transcript of the IP generation process, concluding with the file path and name of the imported IP core file.

**Note** IP cores are native to Vivado, and can be customized and regenerated within that tool. However, PlanAhead provides integration to ChipScope to support legacy IP cores, and any customization and regeneration will occur within that tool. The **convert\_ip** command lets you to convert legacy IP to native IP supported by Vivado.

## Arguments

**-vlnv** <arg> - (Optional) Specifies the VLNV string for the existing Catalog IP from which the new IP will be created. The VLNV is the *Vendor:Library:Name:Version* string which identifies the IP in the catalog. The VLNV string maps to the IPDEF property on the IP core.

**Note** You must specify either **-vlnv** or all of **-vendor**, **-library**, **-name**, and **-version**

**-module\_name** <arg> - Specifies the name for the new IP instance that will be added to the project

**-dir** <arg> - (Optional) The directory to write the IP core files into. If this option is not specified, the IP core files (.xci, .ngc, .veo...) are written into the hierarchy of the <project\_name>.srcs directory.

**Note** This argument is only available for use in Vivado. If you specify this argument in PlanAhead, or if the specified directory does not exist, an error will be returned.

**-vendor** <arg> - (Optional) Specifies the vendor name for the IP's creator.

**-library** <arg> - (Optional) Specifies the IP library from which the core should be added.

**-name** <arg> - (Optional) Specifies the name of the IP core in the catalog.

**-version** <arg> - (Optional) Specifies the version number for the IP core.

**Note** You must specify either **-vlnv** or all of **-vendor**, **-library**, **-name**, and **-version**

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

## Examples

The example below imports the IP core specified by the **-vlnv** string, and gives it the specified module name in the current project:

```
create_ip -vlnv xilinx.com:ip:c_addsub:11.0 -module_name test_addr
```

The following example, from Vivado, creates an IP block with the specified **-vendor**, **-library**, **-name**, **-version** values, and assigns it the specified module name. After the IP is created, attributes of the IP are customized using **set\_property** commands. Then the instantiation template and the synthesis targets are generated for the IP:

```
create_ip -name c_addsub -version 11.0 -vendor xilinx.com -library ip \
  -module_name c_addsub_v11_0_0
set_property -name CONFIG.Component_Name -value {c_addsub_v11_0_0} \
  -objects [get_ips c_addsub_v11_0_0]
set_property -name CONFIG.A_Width -value {32} \
  -objects [get_ips c_addsub_v11_0_0]
set_property -name CONFIG.B_Width -value {32} \
  -objects [get_ips c_addsub_v11_0_0]
set_property -name CONFIG.Add_Mode -value {Add_Subtract} \
  -objects [get_ips c_addsub_v11_0_0]
set_property -name CONFIG.C_In -value {true} \
  -objects [get_ips c_addsub_v11_0_0]
generate_target {instantiation_template synthesis} \
  [get_files C:/Data/c_addsub_v11_0_0/c_addsub_v11_0_0.xci \
  -of_objects [get_filesets sources_1]]
```

## See Also

- [convert\\_ip](#)
- [generate\\_target](#)
- [import\\_ip](#)
- [upgrade\\_ip](#)
- [validate\\_ip](#)

## create\_pblock

Create a new Pblock.

### Syntax

```
create_pblock [-parent arg] [-quiet] [-verbose] name
```

### Returns

New pblock object

### Usage

Name	Description
<i>[-parent]</i>	Parent of the new pblock
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>name</i>	Name of the new pblock

### Categories

[XDC](#), [Floorplan](#)

### Description

Defines a Pblock to allow you to add logic instances for floorplanning purposes. Creating a Pblock results in an AREA\_GROUP constraint that is written to the constraint file.

You can add logic elements to the Pblock using the **add\_cells\_to\_pblock** command, and then place the Pblocks onto the fabric of the FPGA using the **place\_pblocks** command. Once Pblocks have been automatically placed you can manually move pblocks and resize them using the **resize\_pblock** command.

You can nest one Pblock inside another for hierarchical floorplanning using the **-parent** option as shown in the first example. You can also nest an existing Pblock inside another Pblock using the **set\_property** command to define the PARENT property as shown in the second example.

**Note** The ISE® implementation software does not support extensive use of this feature. Occasionally, map and placement errors occur on designs with nested Pblocks.

### Arguments

**-parent** *arg* - The name of the parent Pblock to allow creation of nested Pblocks. If the parent is not specified, the default parent of Root is assumed, placing the Pblock at the top of the design. You can use the **get\_pblocks** command to report currently defined Pblocks that can be used as parents.

**Note** If the specified **parent** does not exist an error will be returned

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*name* - The name of the Pblock to be created.

## Examples

The following example creates a Pblock called Video1 inside another Pblock called Vid\_Array:

```
create_pblock -parent Vid_Array Video1
```

The following example creates Pblocks called cpu1 and cpu2, and creates a third Pblock called cpuEngine. Then cpu1 and cpu2 are nested inside cpuEngine using the **set\_property** command:

```
create_pblock cpu1
create_pblock cpu2
create_pblock cpuEngine
set_property PARENT cpuEngine [get_pblocks {cpu1 cpu2}]
```

## See Also

- [add\\_cells\\_to\\_pblock](#)
- [get\\_pblocks](#)
- [place\\_pblocks](#)
- [resize\\_pblock](#)
- [set\\_property](#)

## create\_pin

Create pins in the current design.

### Syntax

```
create_pin [-bus_from arg] [-bus_to arg] -direction arg [-quiet]
[-verbose] pins ...
```

### Returns

Nothing

### Usage

Name	Description
<i>[-bus_from]</i>	Starting bus index
<i>[-bus_to]</i>	Ending bus index
<b>-direction</b>	Pin direction Values: IN, OUT, INOUT
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>pins</i>	Names of pins to create

### Categories

[Netlist](#)

### Description

Add single pins or bus pins to the current netlist of an open Synthesized or Implemented Design. You may define attributes of the pin such as direction and bus width, as well as the pin name.

The pins must be created on an existing cell instance, or it is considered a top-level pin which should be created using the **create\_port** command. If the instance name of a cell is not specified as part of the pin name, an error will be returned.

Netlist editing changes the in-memory view of the netlist in the current design. It does not change the files in the source fileset, or change the persistent design on the disk. Changes made to the netlist may be saved to a design checkpoint using the **write\_checkpoint** command, or may be exported to a netlist file such as Verilog, VHDL, or EDIF, using the appropriate **write\_\*** command.

**Note** Netlist editing is not allowed on an RTL design.

### Arguments

**-bus\_from** *arg* - (Optional) The starting index of a bus pin.

**-bus\_to** *arg* - (Optional) The ending index of a bus pin.

**-direction** [ IN | OUT | INOUT ] - The direction of the pin. Valid values are IN, OUT, and INOUT.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*pins* - The name of the pins to create. You must specify the pin names hierarchically from the cell instance the pin is assigned to. Pins created at the top-level of the design are ports, and should be created with the **create\_port** command..

## Examples

The following example creates a new input pin on the cpuEngine module with the specified pin name:

```
create_pin -direction IN cpuEngine/inPin
```

The following example sets the hierarchy separator, creates a new black box instance of the reference cell, and creates a twenty-four bit bidirectional bus for that instance:

```
set_hierarchy_separator |  
create_cell -reference dmaBlock -black_box usbEngine0|myDMA  
create_pin -direction INOUT -bus_from 0 -bus_to 23 usbEngine0|myDMA|dataBus
```

## See Also

- [create\\_cell](#)
- [remove\\_cell](#)
- [remove\\_pin](#)
- [set\\_hierarchy\\_separator](#)
- [write\\_checkpoint](#)
- [write\\_edif](#)
- [write\\_verilog](#)
- [write\\_vhdl](#)

## create\_port

Create scalar or bus port.

### Syntax

```
create_port -direction arg [-from arg] [-to arg] [-diff_pair]
[-interface arg] [-quiet] [-verbose] name
```

### Returns

List of port objects that were created

### Usage

Name	Description
<b>-direction</b>	Direction of port. Valid arguments are IN, OUT and INOUT
<i>[-from]</i>	Beginning index of new bus
<i>[-to]</i>	Ending index of new bus
<i>[-diff_pair]</i>	Create differential pair of ports
<i>[-interface]</i>	Assign new port to this interface
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>name</i>	Name of the port

### Categories

[PinPlanning](#)

### Description

Creates a port and specifies such parameters as direction, width, single-ended or differential, and optionally assigns it to an existing interface. New ports are added at the top-level of the design hierarchy.

The create\_port command can be used to create a new port in an I/O Planning project, or while editing the netlist of an open Synthesized or Implemented design.

Netlist editing changes the in-memory view of the netlist in the current design. It does not change the files in the source filesset, or change the persistent design on the disk. Changes made to the netlist may be saved to a design checkpoint using the **write\_checkpoint** command, or may be exported to a netlist file such as Verilog, VHDL, or EDIF, using the appropriate **write\_\*** command.

**Note** Netlist editing is not allowed on an RTL design.

### Arguments

**-direction** [ IN | OUT | INOUT ] - The direction of the port. Valid arguments are IN, OUT, and INOUT.

**-from** *arg* - (Optional) The beginning index of a new bus.

**-to** *arg* - (Optional) The ending index of a new bus.

**Note** The **-to** value must be greater than the **-from** value, or an error will be returned.

**-diff\_pair** - (Optional) Create the specified port as a differential pair of ports. In this case both a P and N port will be created for the specified port *name*.

**-interface** *arg* - (Optional) Assign the port to the specified interface.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*name* - The name of the port to create.

## Examples

The following example creates a new input port, named PORT0:

```
create_port -direction IN PORT0
```

The following example creates a four-bit, differential pair output bus utilizing the specified interface:

```
create_port -direction OUT -from 0 -to 3 -diff_pair -interface INTERFACE D_BUS
```

## See Also

- [create\\_interface](#)
- [delete\\_port](#)
- [make\\_diff\\_pair\\_ports](#)
- [place\\_ports](#)
- [write\\_checkpoint](#)
- [write\\_edif](#)
- [write\\_verilog](#)
- [write\\_vhdl](#)

## create\_project

Create a new project.

### Syntax

```
create_project [-part arg] [-force] [-quiet] [-verbose] name
[ dir ]
```

### Returns

New project object

### Usage

Name	Description
<i>[-part]</i>	Target part
<i>[-force]</i>	Overwrite existing project directory
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>name</i>	Project name
<i>[ dir ]</i>	Directory where the project file is saved Default: .

### Categories

Project

### Description

Creates a project file in the specified directory.

### Arguments

**-part** *arg* - specifies the Xilinx part to be used for the project. This can be changed after the project is created. If the **-part** option is not specified, the default part will be used.

**-force** - This option is required to overwrite an existing project. If the project name is already define in the specified *dir* then you must also specify the **-force** option for the PlanAhead tool to overwrite the existing project.

**Note** If the existing project is currently open in the PlanAhead tool, the new project will overwrite the existing project on the disk, but both projects will be opened in the PlanAhead tool. In this case you should probably run the **close\_project** command prior to running **create\_project**.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*name* - This argument does not require a parameter name, however, it must appear before the specified *dir*. Since these commands do not have parameters, the PlanAhead tool interprets the first argument as *name* and uses the second argument as *dir*. A project file is created *name.ppr*, and a project data folder is also created *name.data* and both are written into the specified directory *dir*.

**Note** The project file created by the PlanAhead tool is an RTL source file by default. You must use the `set_property` command to set the `design_mode` property to change the project from an RTL source project to another type of project, such as an I/O Pin Planning project for instance.

*dir* - This argument specifies the directory name to write the new project file into. If the specified directory does not exist a new directory will be created. If the directory is specified with the complete path, the PlanAhead tool uses the specified path name. However, if *dir* is specified without a path, the PlanAhead tool looks for or creates the directory in the current working directory, or the directory from which the PlanAhead tool was launched.

**Note** When creating a project in GUI-mode, the PlanAhead tool appends the filename *name* to the directory name *dir* and creates a project directory with the name *dir/name* and places the new project file and project data folder into that project directory.

## Examples

The following example creates a project called Project1 in a directory called myDesigns:

```
create_project Project1 myDesigns
```

**Note** Because the *dir* is specified as the folder name only, the PlanAhead tool will create the project in the current working directory, or the directory from which the PlanAhead tool was launched.

The following example creates a project called Proj1 in a directory called FPGA in C:/Designs. In addition, the PlanAhead tool will overwrite an existing project if one is found to exist in the specified location. In the second and third lines, the location of **-force** is changed to show the flexibility of argument placement.

```
create_project Proj1 C:/Designs/FPGA -force
-or-
create_project Proj1 -force C:/Designs/FPGA
-or-
create_project -force Proj1 C:/Designs/FPGA
```

**Note** In all cases the first argument without a preceding keyword is interpreted as the *name* variable, and the second argument without a preceding keyword is the *dir* variable.

The following example creates a new project called `pin_project`, and then sets the **design\_mode** property as required for an I/O Pin Planning project, and finally opens an IO design:

```
create_project pin_project C:/Designs/PinPlanning
set_property design_mode PinPlanning [current_fileset]
open_io_design -name io_1
```

## See Also

- [current\\_project](#)
- [set\\_property](#)
- [open\\_io\\_design](#)

## create\_property

Create property for class of objects(s).

### Syntax

```
create_property [-type arg] [-quiet] [-verbose] name class
```

### Returns

The property that was created if success, "" if failure

### Usage

Name	Description
<i>[-type]</i>	Type of property to create; valid values are: string, int, long, double, bool Default: string
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>name</i>	Name of property to create
<i>class</i>	Object type to create property for; valid values are: design, net, cell, pin, port, pblock

### Categories

[PropertyAndParameter](#)

### Description

Creates a new property of the *type* specified with the user-defined *name* for the specified *class* of objects. The property that is created can be assigned to an object of the specified class with the **set\_property** command, but is not automatically associated with all objects of that class.

The **report\_property -all** command will not report the newly created property for an object of the specified class until the property has been assigned to that object.

### Arguments

**-type arg** - The type of property to create. There are four allowed property types:

- **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 long integer values. If a decimal value is specified for an **int** 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 false (0) value.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*name* - The name of the property to be defined. The name is case sensitive.

*class* - 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, and pblock.

## Examples

The following example defines a property called PURPOSE for cell objects:

```
create_property PURPOSE cell
```

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

The following example creates a pin property called COUNT which holds an Integer value:

```
create_property -type int COUNT pin
```

## See Also

- [get\\_property](#)
- [list\\_property](#)
- [list\\_property\\_value](#)
- [report\\_property](#)
- [reset\\_property](#)
- [set\\_property](#)

## create\_reconfig\_module

Create and add a new Reconfigurable Module to a cell. The cell will be marked as a Reconfigurable Partition if it is not already.

### Syntax

```
create_reconfig_module [-force] [-blackbox] [-quiet]
[-verbose] name cell
```

### Returns

New reconfigurable module object

### Usage

Name	Description
<i>[-force]</i>	Run the command, even if there are pending constraint changes, which will be lost
<i>[-blackbox]</i>	Create a Black Box Reconfigurable Module. Source and constraint files may not be added to a Black Box RM.
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>name</i>	Name of the new Reconfigurable Module
<i>cell</i>	Cell to receive the new Reconfigurable Module

### Categories

[PartialReconfiguration](#)

### Description

Create a reconfigurable module (RM) and assign it to the specified cell. This defines the specified cell as reconfigurable partition, allows you to associate design sources and constraints to the partition in the sources view, and creates a PBlock containing the partition logic.

A single partition can have multiple RMs to contain different netlists, constraints, or implementations. Use the **set\_property** command to define the design source for the RM fileset, using the **edif\_top\_file** property. See the example below.

You can also define a blackbox RM for the partition cell in order to blackout the contents of the partition as needed.

Use the **load\_reconfig\_module** command to make a specific module active for a partition cell. The combination of active modules in the design, and all other logic, is called a configuration of the design.

A design with multiple configurations should be checked to insure that the static logic and partition pins are consistent across all configurations. You can use the **verify\_config** command to check the configurations of a design.

This command returns the hierarchical name of the newly created RM.

## Arguments

**-force** - (Optional) Force the creation of the RM even when there are outstanding design changes to be saved. If there are pending design changes, and **-force** is not specified, an error will be returned.

**Note** You should use **save\_design** prior to using **create\_reconfig\_module**.

**-blackbox** - (Optional) Create a Black Box RM. Design source and constraint files may not be added to a Black Box RM.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*name* - The name of the new RM to create. This name is added to the cell name to create the hierarchical name of the RM *<cell>:<name>*.

*cell* - The name of the partition cell to assign the RM to.

## Examples

The example below creates a reconfigurable module with the name **BramFirst** on the specified cell, sets the **edif\_top\_file** property to reference the appropriate netlist file, and loads the module netlist into the current design:

```
create_reconfig_module -name BramFirst -cell U1_RP_Bram
set_property edif_top_file \
    C:/Data/PlanAhead_Projects/xpr_bram_led/Synth/BramFirst/recon_block_bram.ngc \
    [get_filesets U1_RP_Bram#BramFirst]
save_design
load_reconfig_modules -reconfig_modules U1_RP_Bram:BramFirst
```

**Note** The name of the module fileset specified in the **set\_property** command is the concatenated names of the cell and the reconfigurable module name: *<cell>#<name>*

The example below creates a Blackbox RM named **Bram\_BB** on the specified cell:

```
create_reconfig_module -blackbox -name Bram_BB -cell U1_RP_Bram
```

## See Also

- [config\\_partition](#)
- [create\\_pblock](#)
- [delete\\_reconfig\\_module](#)
- [save\\_design](#)
- [set\\_property](#)
- [verify\\_config](#)

## create\_run

Define a synthesis or implementation run for the current project.

### Syntax

```
create_run [-constrset arg] [-parent_run arg] [-part arg]
-flow arg [-strategy arg] [-quiet] [-verbose] name
```

### Returns

Run object

### Usage

Name	Description
<i>[-constrset]</i>	Constraint fileset to use
<i>[-parent_run]</i>	Synthesis run to link to new implementation run
<i>[-part]</i>	Target part
<b>-flow</b>	Flow name
<i>[-strategy]</i>	Strategy to apply to the run
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>name</i>	Name for new run

### Categories

[Project](#)

### Description

Defines a synthesis or implementation run. The attributes of the run can be configured with the use of the **set\_property** command.

### Arguments

**-constrset** *arg* - The constraint set to use for the synthesis or implementation run.

**-parent\_run** *arg* - The synthesis run which the implementation run will implement. For an RTL sources project, the parent\_run must be specified for implementation runs, but is not required for synthesis runs. For netlist-based projects the parent\_run argument is not required to define an implementation run.

**-part** *partName* - The Xilinx part to be used for the run. If the **-part** option is not specified, the default part defined for the project will be assigned as the part to use.

**-flow** *arg* - The tool flow and release version for the synthesis tool ({XST 14} or {Vivado Synthesis 2012}) or the implementation tool ({ISE 14} or {Vivado Implementation 2012}).

**-strategy** *arg* - The strategy to employ for the synthesis or implementation run. There are many different strategies to choose from within the tool, including custom strategies you can define. Refer to the appropriate user guide for a discussion of the available synthesis and implementation strategies. If the strategy argument is not specified, "Synthesis Defaults" or "Implementation Defaults" will be used as appropriate.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*name* - The name of the synthesis or implementation run to be created.

## Examples

The following example creates a run named `first_pass` referencing the XST 13 flow:

```
create_run -flow {XST 13} first_pass
```

**Note** The defaults of `sources_1`, `constrs_1`, and the default part for the project will be used in the synthesis run. In addition, since this is a synthesis run, the **-parent\_run** argument is not required.

The following example creates an implementation run based on the ISE 13 tool flow, and attaches it to the `first_pass` synthesis run previously created:

```
create_run -flow {Vivado Implementation 2012} -parent_run first_pass impl_1
```

**Note** The **-parent\_run** argument is required in this example because it is an implementation of synthesized RTL sources.

## See Also

- [current\\_run](#)
- [launch\\_runs](#)
- [set\\_property](#)

## create\_slack\_histogram

Create Histogram.

### Syntax

```
create_slack_histogram [-to args] [-delay_type arg]
[-num_bins arg] [-slack_less_than arg] [-slack_greater_than arg]
[-group args] [-report_unconstrained] [-significant_digits arg]
[-scale arg] [-name arg] [-quiet] [-verbose]
```

### Returns

Nothing

### Usage

Name	Description
<code>[-to]</code>	To clock
<code>[-delay_type]</code>	Type of path delay: Values: max, min, min_max Default: max
<code>[-num_bins]</code>	Maximum number of bins: Value =1 Default: 10
<code>[-slack_less_than]</code>	Display paths with slack less than this Default: 1e+30
<code>[-slack_greater_than]</code>	Display paths with slack greater than this Default: -1e+30
<code>[-group]</code>	Limit report to paths in this group(s)
<code>[-report_unconstrained]</code>	Report unconstrained end points
<code>[-significant_digits]</code>	Number of digits to display: Range: 0 to 13 Default: 3
<code>[-scale]</code>	Type of scale on which to draw the histogram; Values: linear, logarithmic Default: linear
<code>[-name]</code>	Output the results to GUI panel with this name
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

### Categories

Report

### Description

Create a slack histogram grouping paths into slack ranges, and displaying the results graphically.

This command provides a graphical slack histogram that requires the tool to be running in GUI mode the **-name** argument to be used.

## Arguments

**-to** *args* - (Optional) Specify a clock name, to analyze paths that end in the specified clock domain.

**-delay\_type** *arg* - (Optional) Specifies the type of path delay to analyze when creating the slack report. The valid values are min, max, and min\_max. The default setting for **-delay\_type** is max.

**-num\_bins** *args* - (Optional) Specify the number of slack bins to divide the results into. The number of bins determines the granularity of the histogram returned. The range of slack values calculated is divided evenly into the specified number of bins, and the paths are grouped into the bins according to their slack values. The value can be specified as a number => 1, with a default value of 10.

**-slack\_less\_than** *arg* - Report slack on paths with a calculated slack value less than the specified value. Used with **-slack\_greater\_than** to provide a range of slack values of specific interest.

**-slack\_greater\_than** *arg* - Report slack on paths with a calculated slack value greater than the specified value. Used with **-slack\_less\_than** to provide a range of slack values of specific interest.

**-group** *args* - Report slack for paths in the specified path groups. Currently defined path groups can be determined with the **get\_path\_groups** command.

**-report\_unconstrained** - Report delay slack on unconstrained paths. By default, unconstrained paths are not analyzed.

**-significant\_digits** *arg* - The number of significant digits in the output results. The valid range is 0 to 13. The default setting is 3 significant digits.

**-scale** [ **linear** | **logarithmic** ] - Specify the Y-axis scale to use when presenting the slack histogram. Logarithmic allows for a smoother presentation of greatly different values, but linear is the default.

**-name** *arg* - (Optional) Specifies the name of the results set for the GUI. If the name specified is currently opened, the **create\_slack\_histogram** will overwrite the current results.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

## Examples

The following example creates a slack histogram of the current design, using the default values, and outputting the results to the named result set in the GUI:

```
create_slack_histogram -name slack1
```

## See Also

- [delete\\_timing\\_results](#)
- [get\\_path\\_groups](#)
- [report\\_timing](#)

## create\_sysgen

Create DSP source for Xilinx System Generator and add to the source fileset.

### Syntax

```
create_sysgen [-quiet] [-verbose] name
```

### Returns

Name for the new sub module

### Usage

Name	Description
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>name</i>	Sub module name

### Categories

[SysGen](#)

## create\_violation

Create a drc violation.

### Syntax

```
create_violation -name arg [-severity arg] [-msg arg] [-quiet]
[-verbose] [objects ...]
```

### Returns

Nothing

### Usage

Name	Description
<b>-name</b>	Specify the name for this rule. This is the typically a 4-6 letter specification for your rule.
<i>[-severity]</i>	Specify severity level for a drc rule. Default: WARNING. Values: FATAL, ERROR, CRITICAL_WARNING, WARNING, ADVISORY.
<i>[-msg]</i>	Specify your message string for this drc rule.
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ objects ]</i>	Cells, ports, pins, nets, clock regions, sites, package banks to query.

### Categories

[Report](#)

### Description

Create a violation object and manage the collection of design objects associated with the violation for reporting purposes.

The violation object is created by the Tcl checker procedure that defines the DRC rule checking functionality. The **create\_violation** command is specified as part of the Tcl checker. The Tcl checker is registered as part of the registration for the DRC rule in the **register\_drc\_rule** command.

### Arguments

**-name** *arg* - The unique abbreviation for the rule in the violation library. This is the same name given to the violation in the **register\_drc\_rule** command.

**-severity** *arg* - (Optional) The severity of the violation. The default severity level for user-defined DRCs is WARNING. The supported values are:

- FATAL
- ERROR
- CRITICAL\_WARNING
- WARNING
- ADVISORY

**-msg** *arg* - (Optional) This is the string substituted for the \$STR in the message specified in the **register\_drc\_rule** if one is defined.

**Note** This must match the **-msg** specification from the **register\_drc\_rule**.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*objects* - (Optional) Cell, port, pin, net, clock region, site, and IO bank objects to check for violations. These are the netlist elements (%NETLIST\_ELEMENT), device sites (%SITE\_GROUP), clock regions (%CLOCK\_REGION), and package IO blocks (%BANK) that are substituted into the rule for the specified DRC abbreviation as defined in the **register\_drc\_rule** command.

## Examples

The following example creates a violation object named RIPO, which is an ERROR when found, returning the specified message string, and checking the cells stored in the two Tcl variables:

```
create_violation -name RIPO -msg "RIPO Error Found:" \
  -severity ERROR $mycell11 $mycell12
```

The following is a sample Tcl checker proc definition, incorporating the **create\_violation** command:

```
proc cell_check {} {
  puts "Executing DRC rule cell_check."
  set mycell13 [lindex [get_cells] 3]
  set mycell15 [lindex [get_cells] 5]
  puts $mycell13
  set vio_1 [create_violation -abbrev CLCHK -msg "Rule 1: failed." \
    -severity ERROR $mycell13 $mycell15]
  set vio_2 [create_violation -abbrev CLCHK -msg "Rule 1: failed again." \
    -severity ERROR $mycell15 $mycell13]
  set x [list $vio_1 $vio_2]
  return -code error $x
}
```

## See Also

- [register\\_drc\\_rule](#)
- [report\\_drc](#)
- [report\\_user\\_drc\\_rule](#)
- [reset\\_drc](#)

## create\_xps

Create embedded source for XPS and add to the source fileset.

### Syntax

```
create_xps [-quiet] [-verbose] name
```

### Returns

Source file name that was created

### Usage

Name	Description
<i>[-srcset]</i>	HIDDEN. Fileset name
<i>[-target_language]</i>	HIDDEN. Language type for the embedded source top ("verilog" or "vhdl")
<i>[-batch]</i>	HIDDEN. Generate embedded source project only. No design configuration.
<i>[-bsb]</i>	HIDDEN. Specify the BSB filename
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>name</i>	Source name

### Categories

[XPS](#)

## crossprobe\_fed

Crossprobe paths of Bels and Nets to FPGAEEditor.

### Syntax

```
crossprobe_fed [-run arg] [-path args] [-objects args] [-quiet]
[-verbose]
```

### Returns

Nothing

### Usage

Name	Description
<code>[-run]</code>	Implemented run to launch FED with
<code>[-path]</code>	Path connecting primitive cells and nets
<code>[-objects]</code>	List of cells and nets to crossprobe
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

### Categories

[ToolLaunch](#)

### Description

This command allows you to cross-probe from the PlanAhead tool to the Xilinx FPGA Editor that was opened with the **launch\_fpga\_editor** command. You can select both objects and timing paths to cross-probe between the editors.

### Arguments

**-run** *name* - The run name to use when cross-probing.

**-path** *paths* - One or more paths to cross-probe in the FPGA Editor.

**-objects** *objects* - One or more objects to cross-probe in the FPGA Editor. You can use any of the **get\_\*** commands, such as **get\_cells** or **get\_ports**, to select objects for cross-probing in the FPGA Editor. See the example below.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

## Examples

The following example defines a group of objects to be cross-probed in the FPGA Editor using the **get\_cells** command to hierarchically locate primitive cells:

```
crossprobe_fed -run impl_1 -objects [get_cells -hier -filter {IS_PRIMITIVE==1}]
```

The following example identifies a path to be cross-probed in the FPGA Editor:

```
crossprobe_fed -path {wbClk i_2090 wbClk_IBUF i_2089 n_0_2089 \  
egressLoop[4].egressFifo/buffer_fifo/infer_fifo.empty_reg_reg}
```

## See Also

- [get\\_cells](#)
- [get\\_ports](#)
- [launch\\_fpga\\_editor](#)

## current\_design

Set or get the current design.

### Syntax

```
current_design [-quiet] [-verbose] [design]
```

### Returns

Design object

### Usage

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

### Categories

[SDC](#), [XDC](#)

### Description

Defines the current design or returns the name of the current design in the active project.

The current design and current instance are the target of most Tcl commands, design edits and constraint changes made in the tool. The current instance can be defined using the **current\_instance** command.

You can use the **get\_designs** command to get a list of open designs in the active project, and use the **get\_projects** command to get a list of open projects.

### Arguments

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*design* - (Optional) The name of design to set as the current design. If a *design* is not specified, the command returns the current design of the active project.

### Examples

The following example sets the specified RTL design as the current design:

```
current_design rtl_1
```

## See Also

- [current\\_instance](#)
- [get\\_designs](#)
- [get\\_projects](#)

## current\_fileset

Get the current fileset.

### Syntax

```
current_fileset [-constrset] [-quiet] [-verbose]
```

### Returns

Current fileset (the current srcset by default)

### Usage

Name	Description
<i>[-constrset]</i>	Get the current constraints fileset
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution

### Categories

[Project](#)

### Description

Get the name of the currently active source constraint fileset within the current project.

### Arguments

**-constrset** - (Optional) Return the name of the currently active constraint set. Without this argument the active source fileset is returned.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

### Examples

The following example returns the name of the active constraint file set.

```
current_fileset -constrset -quiet
```

**Note** With the **-quiet** option specified, the PlanAhead tool will not return anything if it encounters an error in processing the command.

## current\_instance

Set or get the current instance.

### Syntax

```
current_instance [-quiet] [-verbose] [instance]
```

### Returns

Instance name

### Usage

Name	Description
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ instance ]</i>	Name of instance

### Categories

[SDC](#), [XDC](#)

### Description

Set the current instance in the design hierarchy to the specified instance cell or to the top module.

**Note** This command returns the name of the current instance object.

The current design and current instance are the target of most of the commands and design changes you will make. The current design can be defined using the **current\_design** command.

You must specify the *instance* name relative to the currently defined instance, and use the established hierarchy separator to define instance paths. Use '..' to traverse up the hierarchical instance path.

### Arguments

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*instance* - (Optional) The name of the instance to be set as the current instance of the current design. If the *instance* argument is omitted, the current instance is reset to the top module in the design hierarchy. If the *instance* is specified as '.' then the name of the current instance is returned, and the instance is not changed.

**Note** The hierarchical instance path must be specified using the currently defined hierarchy separator, which can be found with the **get\_hierarchy\_separator** command

## Examples

The following example sets the current instance to the top module of the current design:

```
current_instance
INFO: [PlanAhead-618] Current instance is the top level of design 'rtl_1'.
```

The following example first sets the hierarchy separator character, and then sets the current instance relative to the presently defined current instance:

```
set_hierarchy_separator |
current_instance ..|cpu_iwb_dat_o|buffer_fifo
```

The following example returns the name of the presently defined current instance:

```
current_instance .
cpuEngine|cpu_iwb_dat_o|buffer_fifo
```

## See Also

- [current\\_design](#)
- [get\\_hierarchy\\_separator](#)
- [set\\_hierarchy\\_separator](#)

## current\_project

Set or get current project.

### Syntax

```
current_project [-quiet] [-verbose] [project]
```

### Returns

Current or newly set project object

### Usage

Name	Description
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ project ]</i>	Project to set as current

### Categories

[Project](#)

### Description

Specifies the current project or returns the current project when no project is specified.

### Arguments

*project* - The name of the project to make current. This command can be used prior to the **close\_project** to make a specific project active and then to close the project.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

### Examples

The following example sets `project_2` as the current project:

```
current_project project_2
```

This command makes the current project the focus of all the PlanAhead tool commands. In the GUI mode, the current project is defined automatically when switching the GUI between projects.

The following example returns the name of the current project in the PlanAhead tool:

```
current_project
```

**Note** The returned value is the name of the PlanAhead project and not the name or path of the project file.

## See Also

- [close\\_project](#)
- [current\\_design](#)

## current\_run

Set or get the current run.

### Syntax

```
current_run [-synthesis] [-implementation] [-quiet] [-verbose]
[run]
```

### Returns

Run object

### Usage

Name	Description
<i>[-synthesis]</i>	Set or get the current synthesis run
<i>[-implementation]</i>	Set or get the current implementation run (default unless '-synthesis' is specified)
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[run]</i>	Run to set as current; optional

### Categories

[Project](#)

### Description

Defines the current synthesis or implementation run, or returns the name of the current run. The current run is the one automatically selected when the Synthesize or Implement commands are launched.

You can use the **get\_runs** command to determine the list of defined runs in the current design.

### Arguments

**-synthesis** - (Optional) Specifies that the **current\_run** command should set or return the name of the current synthesis run.

**-implementation** - (Optional) Specifies that the **current\_run** command should set or return the name of the current implementation run. This is the default used when neither **-synthesis** or **-implementation** are specified.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*run* - (Optional) Sets the name of the synthesis or implementation run to make the current run.

## Examples

The following example defines the `first_pass` run as the `current_run`:

```
current_run first_pass
```

**Note** The **-synthesis** and **-implementation** arguments are not required because the name allows the PlanAhead tool to identify the specific run of interest.

The following command returns the name of the current implementation run:

```
current_run -implementation -quiet
```

## See Also

[get\\_runs](#)

## delete\_debug\_core

Delete ChipScope debug core.

### Syntax

```
delete_debug_core [-quiet] [-verbose] cores ...
```

### Returns

Nothing

### Usage

Name	Description
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>cores</i>	ChipScope debug cores to delete

### Categories

[ChipScope](#)

### Description

Removes ChipScope debug cores from the current project. The debug cores may have been added by the **create\_debug\_core** command, or imported by the **read\_chipscope\_cdc** command. In either case the core will be removed from the current project.

### Arguments

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*cores* - One or more ChipScope debug core names to remove from the current project.

### Examples

The following command deletes the myCore debug core from the current project:

```
delete_debug_core myCore
```

The following command deletes all debug cores from the current project:

```
delete_debug_core [get_debug_cores]
```

**Note** The **get\_debug\_cores** command returns all debug cores as a default.

## See Also

- [create\\_debug\\_core](#)
- [get\\_debug\\_cores](#)
- [read\\_chipscope\\_cdc](#)

## delete\_debug\_port

Delete ChipScope debug port.

### Syntax

```
delete_debug_port [-quiet] [-verbose] ports ...
```

### Returns

Nothing

### Usage

Name	Description
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>ports</i>	ChipScope debug ports to delete

### Categories

[ChipScope](#)

### Description

Deletes ports from ChipScope debug cores in the current project. You can disconnect a signal from a debug port using **disconnect\_debug\_port**, or remove the port altogether using this command.

### Arguments

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*ports* - The core\_name/port\_name of the debug port to be removed from the core.

### Examples

The following example deletes the DATA port from myCore:

```
delete_debug_port myCore/DATA
```

**Note** Some ports cannot be deleted because an ILA port requires one CLK port and one TRIG port as a minimum.

The following example deletes the trigger ports (TRIG) from the myCore debug core:

```
delete_debug_port [get_debug_ports myCore/TRIG*]
```

**Note** This example will not delete all TRIG ports from myCore, because an ILA core must have at least one TRIG port. The effect of this command will be to delete the TRIG ports starting at TRIG0 and removing all of them except the last port.

## See Also

- [disconnect\\_debug\\_port](#)
- [get\\_debug\\_ports](#)

## delete\_fileset

Delete a fileset.

### Syntax

```
delete_fileset [-quiet] [-verbose] fileset
```

### Returns

Nothing

### Usage

Name	Description
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>fileset</i>	Fileset to be deleted

### Categories

[Project](#), [Simulation](#)

### Description

Deletes the specified fileset. However, if the fileset cannot be deleted, then no message is returned.

### Arguments

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*fileset* - The name of the fileset to delete. The last constraint or simulation fileset will not be deleted, and no error will be returned under these circumstances.

### Examples

The following example deletes the sim\_2 fileset from the current project.

```
delete_fileset sim_2
```

**Note** The fileset and all of its files are removed from the project. The files are not removed from the hard drive.

### See Also

- [create\\_fileset](#)
- [current\\_fileset](#)

## delete\_interface

Delete I/O port interfaces from the project.

### Syntax

```
delete_interface [-all] [-quiet] [-verbose] interfaces ...
```

### Returns

Nothing

### Usage

Name	Description
<i>[-all]</i>	Also delete all of the ports and buses belonging to the interface
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>interfaces</i>	I/O port interfaces to delete

### Categories

[XDC](#), [PinPlanning](#)

### Description

Deletes an existing interface and optionally deletes all of the associated ports and buses using the interface.

### Arguments

**-all** - (Optional) Delete all ports, buses, or nested interfaces associated with the specified interface.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*interfaces* - The name of interfaces to delete.

### Examples

The following example deletes the specified interface and all of its associated ports and buses:

```
delete_interface USB0
```

## See Also

- [create\\_interface](#)
- [create\\_port](#)

## delete\_pblock

Remove Pblock.

### Syntax

```
delete_pblock [-hier] [-quiet] [-verbose] pblocks ...
```

### Returns

Nothing

### Usage

Name	Description
<i>[-hier]</i>	Also delete all the children of Pblock
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>pblocks</i>	Pblocks to delete

### Categories

[Floorplan](#), [XDC](#)

### Description

Deletes the specified Pblocks from the design. Pblocks are created using the **create\_pblock** command.

### Arguments

**-hier** - (Optional) Specifies that Pblocks nested inside the specified Pblock should also be deleted. If the parent Pblock is deleted without the **-hier** option specified, the nested Pblocks will simply be moved up one level.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*pblocks* - One or more Pblocks to be deleted.

### Examples

The following example deletes the specified Pblock as well as any Pblocks nested inside:

```
delete_pblock -hier cpuEngine
```

### See Also

[create\\_pblock](#)

## delete\_port

Delete the given list of top ports from the netlist.

### Syntax

```
delete_port [-quiet] [-verbose] ports ...
```

### Returns

Nothing

### Usage

Name	Description
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>ports</i>	Ports and/or bus ports to delete

### Categories

[PinPlanning](#)

### Description

Deletes the specified ports or busses.

The **delete\_port** command will delete ports that have been added with the **create\_port** command, but cannot delete ports that are defined in the RTL or netlist design.

### Arguments

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*ports* - One or more names of ports to delete.

### Examples

The following example deletes the specified port:

```
delete_port PORT0
```

The following example deletes the two specified ports of a bus:

```
delete_port BUS[1] BUS[2]
```

The following example deletes both the N and P sides of a differential pair port:

```
delete_port D_BUS_P[0]
```

**Note** Deleting either the N or the P side of a differential pair will also delete the other side of the pair.

## See Also

- [create\\_port](#)
- [create\\_interface](#)
- [place\\_ports](#)

## delete\_power\_results

Delete power results that were stored in memory under a given name.

### Syntax

```
delete_power_results -name arg [-quiet] [-verbose]
```

### Returns

Nothing

### Usage

Name	Description
<b>-name</b>	Name for the set of results to clear
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution

### Categories

[XDC](#)

### Description

Deletes the power analysis results for the specified results set.

**Note** This command operates silently and does not return direct feedback of its operation

### Arguments

**-name** *arg* - The name of the results set to delete. This name was either explicitly defined, or was automatically defined when the **report\_power** command was run.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

### Examples

The following example runs power analysis, and then clears the results:

```
report_power -name my_set  
delete_power_results -name my_set
```

## See Also

- [power\\_opt\\_design](#)
- [report\\_power](#)
- [reset\\_switching\\_activity](#)
- [set\\_switching\\_activity](#)

## delete\_reconfig\_module

Remove Reconfigurable Module(s).

### Syntax

```
delete_reconfig_module [-quiet] [-verbose] reconfig_module
```

### Returns

Nothing

### Usage

Name	Description
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>reconfig_module</i>	Reconfigurable Module(s) to delete

### Categories

[PartialReconfiguration](#)

### Description

Removes the specified reconfigurable module (RM) from the current design.

### Arguments

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*reconfig\_module* - One or more reconfigurable modules to delete from the design. The RM is specified by the hierarchical name *<cell>:<name>*.

### Examples

In the example below the BramSecond reconfigurable module is deleted from the U1\_RP\_Bram partition cell:

```
delete_reconfig_module U1_RP_Bram:BramSecond
```

### See Also

[create\\_reconfig\\_module](#)

## delete\_rpm

Delete an RPM.

### Syntax

```
delete_rpm [-quiet] [-verbose] rpm
```

### Returns

Nothing

### Usage

Name	Description
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>rpm</i>	RPM to delete

### Categories

[Floorplan](#)

### Description

Deletes the specified Relationally Placed Macro (RPM) from the design.

An RPM is a collection of logic elements (FFS, LUT, CY4, RAM, etc.) collected into a set (U\_SET, H\_SET, and HU\_SET). The placement of each element within the set, relative to other elements of the set, is controlled by Relative Location Constraints (RLOCs). Logic elements with RLOC constraints and common set names are associated in an RPM. Refer to the Constraints Guide (UG625) for more information on defining these constraints.

Only user-defined RPMs can be deleted from the design. RPMs defined by the hierarchy or defined in the netlist cannot be deleted by this command.

### Arguments

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*rpm* - The RPM to be deleted.

### Examples

The following example deletes the specified RPM (cs\_ila\_0/U0) from the design:

```
delete_rpm cs_ila_0/U0
```

## delete\_run

Delete an existing run.

### Syntax

```
delete_run [-noclean_dir] [-quiet] [-verbose] run
```

### Returns

Nothing

### Usage

Name	Description
<i>[-noclean_dir]</i>	Do not remove all output files and directories from disk
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>run</i>	Run to modify

### Categories

[Project](#)

### Description

Deletes the specified run from the project, and deletes all results of the run from the project directory on the hard drive unless otherwise specified.

### Arguments

**-noclean\_dir** - Do not delete the run results from the hard drive. The run will be deleted from the project, but the run files will remain in the project directory.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*run* - The name of the synthesis or implementation run to delete from the project.

### Examples

The following example deletes the `first_pass` run from the project:

```
delete_run first_pass
```

**Note** In this example, all run results will also be removed from the project directory on the hard drive.

The following command deletes the first\_pass run, but leaves the run results on the hard drive:

```
delete_run -noclean_dir first_pass
```

## See Also

- [create\\_run](#)
- [current\\_run](#)

## delete\_timing\_results

Clear a set of timing results from memory.

### Syntax

```
delete_timing_results [-type arg] [-quiet] [-verbose] name
```

### Returns

Nothing

### Usage

Name	Description
<i>[-type]</i>	Type of timing results to clear; Values: timing_path, slack_histogram, clock_interaction, check_timing, min_pulse_width, timing_summary
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>name</i>	Name for the set of results to clear

### Categories

Report, XDC

### Description

Clear the specified timing results from the named result set. Both the type of the timing report, and the name of the timing report must be specified, or the command will fail.

### Arguments

**-type** *arg* - (Optional) Specifies the type of timing results to be cleared. The available types are: timing\_path, slack\_histogram, clock\_interaction, check\_timing, min\_pulse\_width, timing\_summary.

**Note** The default type is the **report\_timing** report. Do not specify the **-type** argument to delete reports generated by the **report\_timing** command.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

**-name** *arg* - Specifies the name of the timing results to be cleared.

## Examples

The following example clears the specified results set from memory:

```
delete_timing_results -type clock_interaction -name clkNets
```

## See Also

- [check\\_timing](#)
- [create\\_slack\\_histogram](#)
- [report\\_clock\\_interaction](#)
- [report\\_min\\_pulse\\_width](#)
- [report\\_timing](#)
- [report\\_timing\\_summary](#)

## delete\_utilization\_results

Delete utilization results that were stored in memory under a given name.

### Syntax

```
delete_utilization_results -name arg [-quiet] [-verbose]
```

### Returns

Nothing

### Usage

Name	Description
<b>-name</b>	Name for the set of results to clear
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution

### Categories

[Report](#)

### Description

Clear the specified utilization results from the named result set.

### Arguments

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

**-name *arg*** - Specifies the name of the results to be cleared.

### Examples

The following example clears the specified results set from memory:

```
delete_utilization_results -name SS01
```

### See Also

[report\\_utilization](#)

## demote\_run

Unpromote previously promoted Partitions so that they are no longer available for import.

### Syntax

```
demote_run [-run arg] [-partition_names args] [-promote_dir arg]
[-quiet] [-verbose]
```

### Returns

Nothing

### Usage

Name	Description
<code>[-run]</code>	Promoted run to be demoted
<code>[-partition_names]</code>	List of Partitions to be promoted
<code>[-promote_dir]</code>	Directory to be demoted
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

### Categories

[PartialReconfiguration](#), [Partition](#)

### Description

Delete previously promoted partitions so they are no longer available for import using design preservation.

WARNING - This command will delete the specified partition directory and data from the hard drive. Be sure this is the desired result prior to executing this command.

### Arguments

**-run** *args* - (optional) The run to be demoted.

**-partition\_names** *args* - (optional) The names of partitions to be demoted.

**-promote\_dir** *arg* - (optional) The path to the partition data to be demoted.

**Note** If the path to the directory is not specified, the tool will look in your home directory for the partition data:

- For Windows: %APPDATA%/Xilinx/PlanAhead
- For Linux: \$HOME/.Xilinx/PlanAhead

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

## Examples

The following example demotes the specified partition data for the usbEngine0 partition:

```
demote_run -promote_dir C:/Data/partition/partition_DP_RTL.promote/Ximpl_1 \  
-partition_names usbEngine0
```

## See Also

- [promote\\_run](#)
- [verify\\_config](#)

## disconnect\_debug\_port

Disconnect nets and pins from debug port channels.

### Syntax

```
disconnect_debug_port [-channel_index arg] [-quiet]  
[-verbose] port
```

### Returns

Nothing

### Usage

Name	Description
<i>[-channel_index]</i>	Disconnect the net at channel index
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>port</i>	Debug port name

### Categories

[ChipScope](#)

### Description

Disconnect signals from the debug ports.

Signals from the Netlist Design are connected to ports of a ChipScope debug core using the **connect\_debug\_port** command.

A port can also be deleted from the debug core rather than simply disconnected by using the **delete\_debug\_port** command.

If you need to determine the specific name of a port on a debug core, use the **get\_debug\_ports** command to list all ports on a core. You can also use the **report\_debug\_core** command to list all of the cores in the projects, and their specific parameters.

### Arguments

**-channel\_index** *value* - The channel index of the port to disconnect.

**Note** The entire port is disconnected if **channel\_index** is not specified.

*port* - The name of the port on the debug core to disconnect. The port name must be specified as core\_name/port\_name. See the examples below.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

## Examples

The following example disconnects only the specified channel index from the TRIG0 port of myCore:

```
disconnect_debug_port -channel_index 2 myCore/TRIG0
```

If you do not specify the `channel_index`, all of the channels of the specified port will be disconnected, as in the following example:

```
disconnect_debug_port myCore/TRIG0
```

## See Also

- [connect\\_debug\\_port](#)
- [delete\\_debug\\_port](#)
- [get\\_debug\\_ports](#)
- [report\\_debug\\_core](#)

## endgroup

End a set of commands that can be undone/redone as a group.

### Syntax

```
endgroup [-quiet] [-verbose]
```

### Returns

Nothing

### Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

### Categories

[GUIControl](#)

### Description

Ends a sequence of commands that can be undone or redone as a series. Use **startgroup** to start the sequence of commands.

**Note** You can have multiple command groups to **undo** or **redo**, but you cannot nest command groups. You must use **endgroup** to end a command sequence before using **startgroup** to create a new command sequence

### Arguments

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

### Examples

The following example defines a startgroup, executes a sequence of related commands, and then executes the endgroup. This sequence of commands can be undone as a group:

```
startgroup
create_pblock pblock_wbArbEngine
create_pblock pblock_usbEng
add_cells_to_pblock pblock_wbArbEngine [get_cells [list wbArbEngine]] -clear_locs
add_cells_to_pblock pblock_usbEng [get_cells [list usbEngine1/usbEngineSRAM]] -clear_locs
endgroup
```

## See Also

- [startgroup](#)
- [redo](#)
- [undo](#)

## export\_hardware

Export system hardware platform for SDK.

### Syntax

```
export_hardware [-bitstream] [-dir arg] [-quiet]  
[-verbose] files [run]
```

### Returns

Nothing

### Usage

Name	Description
<i>[-bitstream]</i>	Export bitstream data to SDK export directory
<i>[-dir]</i>	Export directory
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>files</i>	Source files for which the hardware data needs to be exported
<i>[ run ]</i>	Current implementation run

### Categories

[XPS](#)

## filter

Filter a list, resulting in new list.

### Syntax

```
filter [-regexp] [-nocase] [-quiet] [-verbose] [objects]  
[filter]
```

### Returns

New list

### Usage

Name	Description
<i>[-regexp]</i>	Operators =~ and !~ use regular expressions
<i>[-nocase]</i>	Perform case-insensitive matching (valid only when -regexp specified)
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ objects ]</i>	List of objects to filter
<i>[ filter ]</i>	Filter list with expression

### Categories

[Object](#), [PropertyAndParameter](#), [XDC](#)

### Description

Takes a list of objects, and returns a reduced list of objects that match the specified filter search pattern.

### 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. See [http://www.tcl.tk/man/tcl8.4/TclCmd/re\\_syntax.htm](http://www.tcl.tk/man/tcl8.4/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/tcl8.4/TclCmd/regexp.htm>.

**-nocase** - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of -regexp only.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*objects* - A list of objects that should be filtered to reduce the set to the desired results. The list of objects can be obtained by using one of the many **get\_\*** commands such as **get\_parts**.

*filter* - The expression to use for filtering. The specified pattern filters the list of objects returned based on property values on the objects. You can find out what properties are on an object with the **report\_property** or **list\_property** command. Any property/value pair can be used as a filter. In the case of the "part" object, "DEVICE", "FAMILY" and "SPEED" are some of the properties that can be used to filter results.

The specific operators that can be used in the filter expression are **==**, **!=**, and **=~**, as well as **&&** and **||** between filter expressions. Numeric comparison operators **<**, **>**, **<=**, and **>=** can also be used. In addition, for boolean (**bool**) type properties, the filter expression can simply evaluate the property directly as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

## Examples

The following example returns a list of parts filtered for the specified speed grade:

```
filter [get_parts] {speed == -3}
```

The following example filters parts based according to speed grade -3 OR speed grade -2. All parts matching either speed grade will be returned.

```
filter [get_parts] {speed == -3 || speed == -2}
```

The following example uses regular expression and returns a list of VStatus ports in the design, with zero or more wildcards, and the numbers 0 to 9 appearing one or more times within square brackets:

```
filter -regexp [get_ports] {NAME =~ VStatus.*\[[0-9]+\]}
```

## See Also

- [get\\_parts](#)
- [get\\_ports](#)

## find\_top

Find top module candidates in the supplied files, fileset, or active fileset. Returns a rank ordered list of candidates.

### Syntax

```
find_top [-fileset arg] [-files args] [-quiet] [-verbose]
```

### Returns

Nothing

### Usage

Name	Description
<i>[-fileset]</i>	Fileset to parse to search for top candidates
<i>[-files]</i>	Files to parse to search for top candidates
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution

### Categories

[Project](#)

## generate\_target

Generate target data for the specified source.

### Syntax

```
generate_target [-force] [-quiet] [-verbose] name objects
```

### Returns

Nothing

### Usage

Name	Description
<i>[-force]</i>	Force target data regeneration
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>name</i>	List of targets to be generated, or 'all' to generate all supported targets
<i>objects</i>	The objects for which data needs to be generated

### Categories

[Project](#), [XPS](#), [IPFlow](#)

### Description

This command generates target data for the specified IP cores. The target data are the files necessary to support the IP core in the context of the broader design project.

Instantiation Template, Synthesis, and Simulation are the standard targets. However, each IP in the catalog may also support its own set of targets such as Testbench, Example, Miscellaneous, etc. Legacy IP support only instantiation\_template and synthesis targets. Native IP support all targets that are supported by that core.

### Arguments

**-force** - (Optional) Force target data regeneration, and overwrite any existing target data files. Without **-force**, the tool will not regenerate any target data that is up-to-date.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*name* - The name of the type of target data to create for the specified IP core. The specific targets supported by an IP core are listed in the SUPPORTED\_TARGETS property on the object. You can query this property to see what targets a specific core supports. Standard values are:

- **all** - Generate all targets for the specified core.
  - **synthesis** - Synthesis targets deliver HDL files that are used during synthesis for native IP, or deliver a synthesized netlist file (NGC) generated by XST.
  - **simulation** - Simulation targets deliver HDL files that are used in simulation.
- Note** The **Simulation** target is only available in Vivado. An error will be returned when specified in PlanAhead.
- **instantiation\_template** - Generate the Instantiation template used to add the RTL module definition for the IP core into the current design. The instantiation template can be copied into any desired level of the design hierarchy.
  - **example** - Some native IP cores support the ability to open example projects containing the core. You must first generate the example target data before opening the core using the **open\_example\_project** command.
  - **testbench** - Used to deliver a test bench that can be used to simulate the IP.
  - **miscellaneous** - Some IP use the miscellaneous target to deliver documentation or scripts used in working with the IP.

*objects* - The IP core objects, or the IP source files (XCI or XCO) to generate the target data from.

## Examples

The following example generates the implementation template for all of the IP cores in the current project, forcing regeneration of any targets which are up-to-date:

```
generate_target instantiation_template [get_ips] -force
```

The following example generates the data files needed to support synthesis and simulation for the specified IP core (XCI file):

```
generate_target {Synthesis Simulation} \
  [get_files C:/data/Projects/test_ip/test_addr_ip/test_addr_ip.xci \
  -of_objects [get_filesets sources_1]]
```

The following example queries the specified IP object to report the SUPPORTED\_TARGETS property, and then generates the Example target data:

```
report_property -all [get_ips blk_mem*]
generate_target {example} [get_ips blk_mem*]
```

## See Also

- [create\\_ip](#)
- [import\\_ip](#)
- [open\\_example\\_project](#)
- [report\\_property](#)
- [reset\\_target](#)

## get\_bel\_pins

Get a list of bel\_pins.

### Syntax

```
get_bel_pins [-regexp] [-nocase] [-filter arg]
[-of_objects args] [-quiet] [-verbose] [patterns]
```

### Returns

Bel\_pindex

### Usage

Name	Description
<i>[-regexp]</i>	Patterns are full regular expressions
<i>[-nocase]</i>	Perform case-insensitive matching. (valid only when -regexp specified)
<i>[-filter]</i>	Filter list with expression
<i>[-of_objects]</i>	Get the bel_pindex of these bels.
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ patterns ]</i>	Match bel_pindex against patterns Default: *

### Categories

[Object](#)

## get\_bels

Get a list of bels.

### Syntax

```
get_bels [-regexp] [-nocase] [-filter arg] [-of_objects args]
[-quiet] [-verbose] [patterns]
```

### Returns

Bels

### Usage

Name	Description
<i>[-regexp]</i>	Patterns are full regular expressions
<i>[-nocase]</i>	Perform case-insensitive matching. (valid only when -regexp specified)
<i>[-filter]</i>	Filter list with expression
<i>[-of_objects]</i>	Get the bels of these sites.
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ patterns ]</i>	Match bels against patterns Default: *

### Categories

Object

### Description

Basic Elements, or BELs, are building blocks of logic, such as flip-flops, LUTs, and carry logic, that make up a SLICE. This command returns a list of BELs on the target part that match a specified search pattern in an open design.

The default command gets a list of all BELs on the device.

**Note** To improve memory and performance, the `get_*` commands return a container list of a single type of objects (i.e. cells, nets, pins, 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. See [http://www.tcl.tk/man/tcl8.4/TclCmd/re\\_syntax.htm](http://www.tcl.tk/man/tcl8.4/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/tcl8.4/TclCmd/regexp.htm>.

**-nocase** - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

**-filter** *args* - Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get\_bels** based on property values on the BELs. You can find the properties on an object with the **report\_property** or **list\_property** commands. Any property/value pair can be used as a filter. In the case of the BEL object, "IS\_OCCUPIED" and "TYPE" are two of the properties that can be used to filter results.

The specific operators that can be used in the filter expression are ==, !=, and =~, as well as && and || between filter expressions. Numeric comparison operators <, >, <=, and >= can also be used. In addition, for boolean (**bool**) type properties, the filter expression can simply evaluate the property directly as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

**-of\_objects** *args* - This option can be used with the **get\_sites** command to return the BELs of specified site objects.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*patterns* - Match BELs against the specified patterns. The default pattern is the wildcard "\*" which gets a list of all BELs on the device. More than one search pattern can be specified to find BELs 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 returns the total number of BELs on the target part:

```
llength [get_bels]
```

The following example returns the BELs associated with the specified site:

```
get_bels -of_objects [get_sites PHASER_IN_PHY_X0Y5]
```

## See Also

- [get\\_sites](#)
- [list\\_property](#)
- [report\\_property](#)

## get\_boards

Get the list of boards available in the project.

### Syntax

```
get_boards [-filter arg] [-quiet] [-verbose]
```

### Returns

List of boards objects

### Usage

Name	Description
<code>[-filter]</code>	Filter list with expression
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

### Categories

[Object](#), [Project](#), [XPS](#)

### Description

Gets a list of evaluation boards available for the current project.

**Note** To improve memory and performance, the `get_*` commands return a container list of a single type of objects (i.e. cells, nets, pins, 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

**-filter** *args* - Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get\_boards** based on property values on the boards. You can find the properties on an object with the **report\_property** or **list\_property** commands. Any property/value pair can be used as a filter. In the case of the board object, "NAME", "DEVICE", and "FAMILY" are some of the properties that can be used to filter results.

The specific operators that can be used in the filter expression are `==`, `!=`, and `=~`, as well as `&&` and `||` between filter expressions. Numeric comparison operators `<`, `>`, `<=`, and `>=` can also be used. In addition, for boolean (**bool**) type properties, the filter expression can simply evaluate the property directly as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns `TCL_OK` regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

## Examples

The following example reports the properties of the specified evaluation board:

```
report_property [get_boards -filter {NAME==ml505}]
```

## See Also

- [list\\_property](#)
- [report\\_property](#)

## get\_cells

Get a list of cells in the current design.

### Syntax

```
get_cells [-hsc arg] [-hierarchical] [-regexp] [-nocase]
[-filter arg] [-of_objects args] [-match_style arg] [-quiet]
[-verbose] [patterns]
```

### Returns

List of cell objects

### Usage

Name	Description
<code>[-hsc]</code>	Hierarchy separator Default: /
<code>[-hierarchical]</code>	Search level-by-level in current instance
<code>[-regexp]</code>	Patterns are full regular expressions
<code>[-nocase]</code>	Perform case-insensitive matching (valid only when -regexp specified)
<code>[-filter]</code>	Filter list with expression
<code>[-of_objects]</code>	Get cells of these pins, timing paths or nets
<code>[-match_style]</code>	Style of pattern matching Default: sdc Values: ucf, sdc
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[ patterns ]</code>	Match cell names against patterns Default: *

### Categories

[SDC](#), [XDC](#), [Object](#)

### Description

Gets a list of cell objects in the current design that match a specified search pattern. The default command returns a list of all cells in the design.

**Note** To improve memory and performance, the `get_*` commands return a container list of a single type of objects (i.e. cells, nets, pins, 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

`-hsc arg` - (Optional) Set the hierarchy separator. The default hierarchy separator is '/'.



## Examples

The following example gets a list of properties and property values attached to a specific cell object:

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

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

The following example prints a list of the library cells instantiated into the design at all levels of the hierarchy, sorting the list for unique names so that each cell is only printed one time:

```
foreach cell [lsort -unique [get_property LIB_CELL [get_cells -hier -filter \
{IS_PRIMITIVE==1}]]] {puts $cell}
```

The following example demonstrates the effect of **-hierarchical** searches, without and with **-regexp**:

```
get_cells -hierarchical *mmcm*
mmcm_replicator_inst_1
mmcm_replicator_inst_1/mmcm_stage[0].mmcm_channel[0].mmcm
get_cells -hierarchical -regexp .*mmcm.*
mmcm_replicator_inst_1
mmcm_replicator_inst_1/mmcm_stage[0].mmcm_channel[0].mmcm
mmcm_replicator_inst_1/mmcm_stage[0].mmcm_channel[0].mmcm/GND
mmcm_replicator_inst_1/mmcm_stage[0].mmcm_channel[0].mmcm/MMCM_Base
```

**Note** The last two cells (GND and MMCM\_Base) were not returned in the first example (without **-regexpr**) because the cell names do not match the search pattern, as it is applied to each level of the hierarchy.

## See Also

- [get\\_lib\\_cells](#)
- [get\\_nets](#)
- [get\\_pins](#)
- [list\\_property](#)
- [report\\_property](#)

## get\_clock\_regions

Get the clock regions for the current device.

### Syntax

```
get_clock_regions [-regexp] [-nocase] [-filter arg]
                  [-of_objects args] [-quiet] [-verbose] [patterns]
```

### Returns

Clock\_regions

### Usage

Name	Description
<i>[-regexp]</i>	Patterns are full regular expressions.
<i>[-nocase]</i>	Perform case-insensitive matching. (valid only when -regexp specified).
<i>[-filter]</i>	Filter list with expression
<i>[-of_objects]</i>	Get the clock_regions of these sites
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ patterns ]</i>	Match objects' name against patterns. Default: *

### Categories

Object

### Description

Gets a list of clock regions on the target part that match a specified search pattern. The default command gets a list of all clock regions on the device in an open design.

**Note** To improve memory and performance, the `get_*` commands return a container list of a single type of objects (i.e. cells, nets, pins, 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. See [http://www.tcl.tk/man/tcl8.4/TclCmd/re\\_syntax.htm](http://www.tcl.tk/man/tcl8.4/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/tcl8.4/TclCmd/regexp.htm>.

**-nocase** - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

**-filter** *args* - Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get\_clock\_regions** based on property values on the clock regions. You can find the properties on an object with the **report\_property** or **list\_property** commands. Any property/value pair can be used as a filter. In the case of the clock region object, "COLUMN\_INDEX", "HIGH\_X", and "LOW\_X" are some of the properties that can be used to filter results.

The specific operators that can be used in the filter expression are **==**, **!=**, and **=~**, as well as **&&** and **||** between filter expressions. Numeric comparison operators **<**, **>**, **<=**, and **>=** can also be used. In addition, for boolean (**bool**) type properties, the filter expression can simply evaluate the property directly as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

**-of\_objects** *args* - This option can be used with the **get\_sites** command to return the clock region that the specified site is found in.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns **TCL\_OK** regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*patterns* - Match clock regions against the specified patterns. The default pattern is the wildcard **"\*"** which gets a list of all clock regions on the device. More than one search pattern can be specified to find clock regions 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 returns the clock regions matching the search pattern:

```
get_clock_regions X0*
```

The following example returns the clock regions filtered by the specified property:

```
get_clock_regions -filter {LOW_X==0}
```

**Note** These two examples return the same set of clock regions

## See Also

- [get\\_sites](#)
- [list\\_property](#)
- [report\\_property](#)

## get\_debug\_cores

Get a list of ChipScope debug cores in the current design.

### Syntax

```
get_debug_cores [-filter arg] [-of_objects args] [-regexp]
[-nocase] [-quiet] [-verbose] [patterns]
```

### Returns

List of debug\_core objects

### Usage

Name	Description
<i>[-filter]</i>	Filter list with expression
<i>[-of_objects]</i>	Get cores of these debug ports or nets
<i>[-regexp]</i>	Patterns are full regular expressions
<i>[-nocase]</i>	Perform case-insensitive matching (valid only when -regexp specified)
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ patterns ]</i>	Match debug cores against patterns Default: *

### Categories

Object, ChipScope

### Description

Gets a list of ChipScope debug cores in the current project that match a specified search pattern. The default command gets a list of all debug cores in the project.

Debug cores are added to the project with the **create\_debug\_core** or the **read\_chipscope\_cdc** commands. When a ChipScope debug core is added to the project, it is contained within an ICON controller core, and includes a CLK port and a trigger port (TRIG) as a default. Additional ports can be added to the debug core with the use of the **create\_debug\_port** command.

**Note** To improve memory and performance, the **get\_\*** commands return a container list of a single type of objects (i.e. cells, nets, pins, 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

**-filter** *args* - Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get\_debug\_cores** based on property values on the parts. You can find the properties on an object with the **report\_property** or **list\_property** commands.

The specific operators that can be used in the filter expression are `==`, `!=`, and `=~`, as well as `&&` and `||` between filter expressions. Numeric comparison operators `<`, `>`, `<=`, and `>=` can also be used. In addition, for boolean (**bool**) type properties, the filter expression can simply evaluate the property directly as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

**-of\_objects** *args* - Get the ChipScope debug cores associated with the specified debug ports, or nets.

**-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. See [http://www.tcl.tk/man/tcl8.4/TclCmd/re\\_syntax.htm](http://www.tcl.tk/man/tcl8.4/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/tcl8.4/TclCmd/regexp.htm>.

**-nocase** - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*patterns* - Match parts against the specified patterns. The default pattern is the wildcard `"*"` which gets all parts. More than one pattern can be specified to find multiple parts 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 command gets a list of the ChipScope debug cores in the current project:

```
get_debug_cores
```

**Note** An ICON core is returned as one of the debug cores in the project. You cannot directly create this core, but it is automatically added by the tool when you add any ILA cores to the project.

The following example gets the properties of the specified debug core:

```
report_property [get_debug_cores myCore]
```

The values of the properties returned depend on how the core is configured. You can use the **set\_property** command to configure specific core properties as shown in the following example:

```
set_property enable_storage_qualification false [get_debug_cores myCore]
```

## See Also

- [create\\_debug\\_core](#)
- [create\\_debug\\_port](#)
- [get\\_debug\\_ports](#)
- [list\\_property](#)
- [read\\_chipscope\\_cdc](#)
- [report\\_property](#)
- [set\\_property](#)

## get\_debug\_ports

Get a list of ChipScope debug ports in the current design.

### Syntax

```
get_debug_ports [-filter arg] [-of_objects args] [-regexp]
[-nocase] [-quiet] [-verbose] [patterns]
```

### Returns

List of debug\_port objects

### Usage

Name	Description
<i>[-filter]</i>	Filter list with expression
<i>[-of_objects]</i>	Get ports of these debug cores
<i>[-regexp]</i>	Patterns are full regular expressions
<i>[-nocase]</i>	Perform case-insensitive matching (valid only when -regexp specified)
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ patterns ]</i>	Match debug ports against patterns Default: *

### Categories

[Object](#), [ChipScope](#)

### Description

Gets a list of ports defined on ChipScope debug cores in the current project that match a specified search pattern. The default command gets a list of all debug ports in the project.

Debug ports are defined when ChipScope debug cores are created with the **read\_chipscope\_cdc** command, or the **create\_debug\_core** command. Ports can be added to existing debug cores with the **create\_debug\_port** command.

**Note** To improve memory and performance, the `get_*` commands return a container list of a single type of objects (i.e. cells, nets, pins, 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

**-filter** *args* - Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get\_debug\_ports** based on property values on the ports. You can find the properties on an object with the **report\_property** or **list\_property** commands. Any property/value pair can be used as a filter. In the case of the debug\_port object, "PORT\_WIDTH", and "MATCH\_TYPE" are some of the properties that can be used to filter results.

The specific operators that can be used in the filter expression are `==`, `!=`, and `=~`, as well as `&&` and `||` between filter expressions. Numeric comparison operators `<`, `>`, `<=`, and `>=` can also be used. In addition, for boolean (**bool**) type properties, the filter expression can simply evaluate the property directly as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

**-of\_objects** *args* - Get the ChipScope debug ports associated with the specified debug cores.

**-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. See [http://www.tcl.tk/man/tcl8.4/TclCmd/re\\_syntax.htm](http://www.tcl.tk/man/tcl8.4/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/tcl8.4/TclCmd/regexp.htm>.

**-nocase** - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns `TCL_OK` regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*patterns* - Match parts against the specified patterns. The default pattern is the wildcard `"*"` which gets all parts. More than one pattern can be specified to find multiple parts 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 command gets a list of the ports from the ChipScope debug cores in the current project, with a `PORT_WIDTH` property of 8:

```
get_debug_ports -filter {PORT_WIDTH==8}
```

The following example gets the properties attached to the specified debug port:

```
report_property [get_debug_ports myCore/TRIG0]
```

**Note** The debug port is defined by the `core_name/port_name` combination.

## See Also

- [create\\_debug\\_core](#)
- [create\\_debug\\_port](#)
- [list\\_property](#)
- [read\\_chipscope\\_cdc](#)
- [report\\_property](#)

## get\_designs

Get a list of designs in the current design.

### Syntax

```
get_designs [-regexp] [-nocase] [-filter arg] [-quiet]
            [-verbose] [patterns]
```

### Returns

List of design objects

### Usage

Name	Description
<i>[-regexp]</i>	Patterns are full regular expressions
<i>[-nocase]</i>	Perform case-insensitive matching (valid only when -regexp specified)
<i>[-filter]</i>	Filter list with expression
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ patterns ]</i>	Match design names against patterns Default: *

### Categories

XDC, Object

### Description

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

**Note** To improve memory and performance, the `get_*` commands return a container list of a single type of objects (i.e. cells, nets, pins, 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. See [http://www.tcl.tk/man/tcl8.4/TclCmd/re\\_syntax.htm](http://www.tcl.tk/man/tcl8.4/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/tcl8.4/TclCmd/regexp.htm>.

**-nocase** - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

**-filter** *args* - Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get\_designs** based on property values on the designs. You can find the properties on an object with the **report\_property** or **list\_property** commands. In the case of the "design" object, "CONSTRSET", and "PART" are some of the properties that can be used to filter results.

The specific operators that can be used in the filter expression are **==**, **!=**, and **=~**, as well as **&&** and **||** between filter expressions. Numeric comparison operators **<**, **>**, **<=**, and **>=** can also be used. In addition, for boolean (**bool**) type properties, the filter expression can simply evaluate the property directly as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns **TCL\_OK** regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

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

## Examples

The following example gets all open designs in the current project:

```
get_designs
```

The following example gets the assigned properties of an open design matching the search pattern:

```
report_property [get_designs r*]
```

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

## See Also

[report\\_property](#)

## get\_files

Get a list of source files.

### Syntax

```
get_files [-regexp] [-nocase] [-filter arg] [-of_objects args]
          [-quiet] [-verbose] [patterns]
```

### Returns

List of file objects

### Usage

Name	Description
<i>[-regexp]</i>	Patterns are full regular expressions
<i>[-nocase]</i>	Perform case-insensitive matching (valid only when -regexp specified)
<i>[-filter]</i>	Filter list with expression
<i>[-of_objects]</i>	Get files of these filesets
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ patterns ]</i>	Match file names against patterns Default: *

### Categories

[Object](#), [Project](#)

### Description

Gets a list of files in the current project that match a specified search pattern. The default command gets a list of all files in the project.

**Note** To improve memory and performance, the `get_*` commands return a container list of a single type of objects (i.e. cells, nets, pins, 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. See [http://www.tcl.tk/man/tcl8.4/TclCmd/re\\_syntax.htm](http://www.tcl.tk/man/tcl8.4/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/tcl8.4/TclCmd/regexp.htm>.

**-nocase** - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

**-filter** *args* - Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get\_files** based on property values on the files. You can find the properties on an object with the **report\_property** or **list\_property** commands. Any property/value pair can be used as a filter. In the case of the "file" object, "FILE\_TYPE", and "IS\_ENABLED" are some of the properties that can be used to filter results.

The specific operators that can be used in the filter expression are **==**, **!=**, and **=~**, as well as **&&** and **||** between filter expressions. Numeric comparison operators **<**, **>**, **<=**, and **>=** can also be used. In addition, for boolean (**bool**) type properties, the filter expression can simply evaluate the property directly as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

**-of\_objects** *args* - Specifies one or more filesets to search for the files. The default is to search all filesets.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*patterns* - 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 gets a list of all files in the current project:

```
get_files
```

The following example gets a list of the Verilog files (\*.v) found in the constrs\_1 and sim\_1 filesets:

```
get_files -of_objects {constrs_1 sim_1} *.v
```

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

## See Also

[report\\_property](#)

## get\_filesets

Get a list of filesets in the current project.

### Syntax

```
get_filesets [-regexp] [-nocase] [-filter arg] [-quiet]
[-verbose] [patterns]
```

### Returns

List of fileset objects

### Usage

Name	Description
<i>[-regexp]</i>	Patterns are full regular expressions
<i>[-nocase]</i>	Perform case-insensitive matching (valid only when -regexp specified)
<i>[-filter]</i>	Filter list with expression
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ patterns ]</i>	Match fileset names against patterns Default: *

### Categories

[Object](#), [Project](#)

### Description

Gets a list of filesets in the current project that match a specified search pattern. The default command gets a list of all filesets in the project.

**Note** To improve memory and performance, the `get_*` commands return a container list of a single type of objects (i.e. cells, nets, pins, 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. See [http://www.tcl.tk/man/tcl8.4/TclCmd/re\\_syntax.htm](http://www.tcl.tk/man/tcl8.4/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/tcl8.4/TclCmd/regexp.htm>.



## get\_hierarchy\_separator

Get hierarchical separator character.

### Syntax

```
get_hierarchy_separator [-quiet] [-verbose]
```

### Returns

Nothing

### Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

### Categories

[SDC](#), [XDC](#)

### Description

Gets the character currently used for separating levels of hierarchy in the design. You can set the hierarchy separator using the **set\_hierarchy\_separator** command.

### Arguments

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

### Examples

The following example gets the currently defined hierarchy separator:

```
get_hierarchy_separator
```

### See Also

[set\\_hierarchy\\_separator](#)

## get\_interfaces

Get a list of I/O port interfaces in the current design.

### Syntax

```
get_interfaces [-regexp] [-nocase] [-filter arg]
               [-of_objects args] [-quiet] [-verbose] [patterns]
```

### Returns

List of interface objects

### Usage

Name	Description
<i>[-regexp]</i>	Patterns are full regular expressions
<i>[-nocase]</i>	Perform case-insensitive matching (valid only when -regexp specified)
<i>[-filter]</i>	Filter list with expression
<i>[-of_objects]</i>	Get interfaces of these pins or nets
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ patterns ]</i>	Match I/O port interfaces against patterns Default: *

### Categories

XDC, Object

### Description

Gets a list of IO interfaces in the current project that match a specified search pattern. The default command gets a list of all IO interfaces in the project.

**Note** To improve memory and performance, the `get_*` commands return a container list of a single type of objects (i.e. cells, nets, pins, 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. See [http://www.tcl.tk/man/tcl8.4/TclCmd/re\\_syntax.htm](http://www.tcl.tk/man/tcl8.4/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/tcl8.4/TclCmd/regexp.htm>.

**-nocase** - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

**-filter** *args* - Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get\_interfaces** based on property values on the interfaces. You can find the properties on an object with the **report\_property** or **list\_property** commands.

The specific operators that can be used in the filter expression are **==**, **!=**, and **=~**, as well as **&&** and **||** between filter expressions. Numeric comparison operators **<**, **>**, **<=**, and **>=** can also be used. In addition, for boolean (**bool**) type properties, the filter expression can simply evaluate the property directly as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

**-of\_objects** *args* - One or more pins or nets to which the interfaces are assigned.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns **TCL\_OK** regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*patterns* - Match interfaces against the specified pattern. The default pattern is the wildcard **"\*"** which gets a list of all interfaces in the project.

## Examples

The following example gets a list of all interfaces in the project:

```
get_interfaces
```

## See Also

- [create\\_interface](#)
- [delete\\_interface](#)

## get\_iobanks

Get a list of iobanks.

### Syntax

```
get_iobanks [-regexp] [-nocase] [-filter arg] [-of_objects args]
[-quiet] [-verbose] [patterns]
```

### Returns

Iobanks

### Usage

Name	Description
<i>[-regexp]</i>	Patterns are full regular expressions
<i>[-nocase]</i>	Perform case-insensitive matching. (valid only when -regexp specified)
<i>[-filter]</i>	Filter list with expression
<i>[-of_objects]</i>	Get the iobanks of these package_pins.
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ patterns ]</i>	Match iobanks against patterns Default: *

### Categories

XDC, Object

### Description

Gets a list of I/O Banks on the target device in the current project that match a specified search pattern. The default command gets a list of all I/O Banks on the target device.

**Note** To improve memory and performance, the `get_*` commands return a container list of a single type of objects (i.e. cells, nets, pins, 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. See [http://www.tcl.tk/man/tcl8.4/TclCmd/re\\_syntax.htm](http://www.tcl.tk/man/tcl8.4/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/tcl8.4/TclCmd/regexp.htm>.

**-nocase** - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

**-filter** *args* - Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get\_iobanks** based on property values on the I/O Banks. You can find the properties on an object with the **report\_property** or **list\_property** commands. In the case of the iobank object, "DCI\_CASCADE", and "INTERNAL\_VREF" are some of the properties that can be used to filter results.

The specific operators that can be used in the filter expression are **==**, **!=**, and **=~**, as well as **&&** and **||** between filter expressions. Numeric comparison operators **<**, **>**, **<=**, and **>=** can also be used. In addition, for boolean (**bool**) type properties, the filter expression can simply evaluate the property directly as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

**-of\_objects** *arg* - Get a list of the iobanks connected to the specified package pins.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns **TCL\_OK** regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*patterns* - Match I/O Banks against the specified pattern. The default pattern is the wildcard **"\*"** which gets a list of all I/O Banks in the project.

## Examples

The following example returns the I/O Bank of the specified package pin:

```
get_iobanks -of_objects [get_package_pins H4]
```

## See Also

- [get\\_package\\_pins](#)
- [report\\_property](#)

## get\_ipdefs

Get a list of IP from the current IP Catalog.

### Syntax

```
get_ipdefs [-regexp] [-nocase] [-filter arg] [-quiet] [-verbose]
[patterns ...]
```

### Returns

List of Catalog IP objects

### Usage

Name	Description
<i>[-regexp]</i>	Patterns are full regular expressions
<i>[-nocase]</i>	Perform case-insensitive matching
<i>[-filter]</i>	Filter list with expression
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ patterns ]</i>	Match Catalog IP names against patterns Default: * Values: The default search pattern is the wildcard *, or .* when -regexp is specified.

### Categories

Object, IPFlow

### Description

Get a list of IP cores defined in the IP catalog of the current project, based on the specified search pattern. The default is to return all IP cores defined in the catalog.

### 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. See [http://www.tcl.tk/man/tcl8.4/TclCmd/re\\_syntax.htm](http://www.tcl.tk/man/tcl8.4/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/tcl8.4/TclCmd/regexp.htm>.

**-nocase** - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of -regexp only.

**-filter** *args* - Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get\_ipdefs** based on property values on the objects. You can find the properties on an object with the **report\_property** or **list\_property** commands. In the case of the "ipdefs" object, "VLNV", "NAME" and "IS\_AXI" are some of the properties that can be used to filter results.

The specific operators that can be used in the filter expression are ==, !=, and =~, as well as && and || between filter expressions. Numeric comparison operators <, >, <=, and >= can also be used. In addition, for boolean (**bool**) type properties, the filter expression can simply evaluate the property directly as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*patterns* - 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 {} to present the list as a single element.

## Examples

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

```
get_ipdefs -filter {NAME=~*agilent*}
```

**Note** The filter operator '=~' loosely matches the specified pattern

The following example returns a list of all AXI compliant IP cores:

```
get_ipdefs -filter {IS_AXI==1}
```

## See Also

- [get\\_ips](#)
- [create\\_ip](#)
- [generate\\_target](#)
- [import\\_ip](#)
- [update\\_ip\\_catalog](#)

## get\_ips

Get a list of IPs in the current design.

### Syntax

```
get_ips [-regexp] [-nocase] [-filter arg] [-quiet] [-verbose]
[patterns ...]
```

### Returns

List of IP objects

### Usage

Name	Description
<i>[-regexp]</i>	Patterns are full regular expressions
<i>[-nocase]</i>	Perform case-insensitive matching
<i>[-filter]</i>	Filter list with expression
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ patterns ]</i>	Match IP names against patterns Default: *

### Categories

Object, Project, IPFlow

### Description

Get a list of IP cores in the current project based on the specified search pattern. The default command returns a list of all IPs in the project.

**Note** To improve memory and performance, the `get_*` commands return a container list of a single type of objects (i.e. cells, nets, pins, 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. See [http://www.tcl.tk/man/tcl8.4/TclCmd/re\\_syntax.htm](http://www.tcl.tk/man/tcl8.4/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/tcl8.4/TclCmd/regexp.htm>.

**-nocase** - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

**-filter** *args* - Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get\_ips** based on property values on the objects. You can find the properties on an object with the **report\_property** or **list\_property** commands. In the case of the "IP" object, "NAME" and "DELIVERED\_TARGETS" are some of the properties that can be used to filter results.

The specific operators that can be used in the filter expression are ==, !=, and =~, as well as && and || between filter expressions. Numeric comparison operators <, >, <=, and >= can also be used. In addition, for boolean (**bool**) type properties, the filter expression can simply evaluate the property directly as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*patterns* - Match IP cores in the design against the specified search patterns. The default pattern is the wildcard '\*' which gets a list of all IP cores in the project. More than one pattern can be specified to find multiple cores 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 returns a list of IP cores with names beginning with the string "EDK":

```
get_ips EDK*
```

## See Also

- [get\\_ipdefs](#)
- [create\\_ip](#)
- [import\\_ip](#)
- [update\\_ip\\_catalog](#)

## get\_lib\_cells

Get a list of library cells. By default, returns all lib cells associated with the current\_design's part.

### Syntax

```
get_lib_cells [-regexp] [-nocase] [-filter arg]
[-of_objects args] [-quiet] [-verbose] [patterns]
```

### Returns

List of library cells

### Usage

Name	Description
<i>[-regexp]</i>	Patterns are full regular expressions
<i>[-nocase]</i>	Perform case-insensitive matching (valid only when -regexp specified)
<i>[-filter]</i>	Filter list with expression
<i>[-of_objects]</i>	Get library cells of cells/insts, or libpins
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ patterns ]</i>	Match lib cell names against patterns; Pattern should explicitly specify library name e.g. libname /*, you can find library names using 'get_libs' command Default: *

### Categories

SDC, XDC, Object

### Description

Get a list of cells in the library for the target part of the current design. Use this command to query and look for a specific library cell, or type of cell and to get the properties of the cells.

This command requires a hierarchical name which includes the library name as well as the cell name: lib\_name/cell\_name.

**Note** To improve memory and performance, the get\_\* commands return a container list of a single type of objects (i.e. cells, nets, pins, 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. See [http://www.tcl.tk/man/tcl8.4/TclCmd/re\\_syntax.htm](http://www.tcl.tk/man/tcl8.4/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/tcl8.4/TclCmd/regexp.htm>.

**-nocase** - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

**-filter** *args* - Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get\_lib\_cells** based on property values on the cells. You can find the properties on an object with the **report\_property** or **list\_property** commands.

The specific operators that can be used in the filter expression are ==, !=, and =~, as well as && and || between filter expressions. Numeric comparison operators <, >, <=, and >= can also be used. In addition, for boolean (**bool**) type properties, the filter expression can simply evaluate the property directly as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

**-of\_objects** *arg* - Get a list of library cells of specific instances (cells/insts), or library pins (get\_lib\_pins).

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*patterns* - Match library cells against the specified patterns. The pattern must specify both the library name and the cell name.

## Examples

The following example gets the number of the cells in the library for the target part in the current design, and then gets the number of AND type cells in that library:

```
llength [get_lib_cells [get_libs]/*]  
795  
llength [get_lib_cells [get_libs]/AND*]  
18
```

The following example gets the library cell for the specified cell object:

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

## See Also

- [get\\_cells](#)
- [get\\_libs](#)
- [get\\_lib\\_pins](#)
- [list\\_property](#)
- [report\\_property](#)

## get\_lib\_pins

Create list of library cell pins.

### Syntax

```
get_lib_pins [-regexp] [-nocase] [-filter arg]
[-of_objects args] [-quiet] [-verbose] [patterns]
```

### Returns

List of library cell pins

### Usage

Name	Description
<i>[-regexp]</i>	Patterns are full regular expressions
<i>[-nocase]</i>	Perform case-insensitive matching (valid only when -regexp specified)
<i>[-filter]</i>	Filter list with expression
<i>[-of_objects]</i>	Get library cell pins these pins or libcells
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ patterns ]</i>	Match lib cell names against patterns Default: *

### Categories

[SDC](#), [XDC](#), [Object](#)

### Description

Gets a list of the pins on a specified cell of the cell library for the target part in the current design.

**Note** This command requires a hierarchical name which includes the library name and cell name as well as the pins: lib\_name/cell\_name/pins.

**Note** To improve memory and performance, the get\_\* commands return a container list of a single type of objects (i.e. cells, nets, pins, 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. See [http://www.tcl.tk/man/tcl8.4/TclCmd/re\\_syntax.htm](http://www.tcl.tk/man/tcl8.4/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/tcl8.4/TclCmd/regexp.htm>.

**-nocase** - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

**-filter** *args* - Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get\_lib\_pins** based on property values on the pins. You can find the properties on an object with the **report\_property** or **list\_property** commands.

The specific operators that can be used in the filter expression are ==, !=, and =~, as well as && and || between filter expressions. Numeric comparison operators <, >, <=, and >= can also be used. In addition, for boolean (**bool**) type properties, the filter expression can simply evaluate the property directly as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

**-of\_objects** *arg* - Get a list of library cell pins of the specified pin objects or library cells (**get\_lib\_cells**).

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*patterns* - Match lib pins against the specified patterns. The pattern must specify the library name, cell name, and the pins.

## Examples

The following example gets a list of all library cell pins:

```
get_lib_pins xt_virtex6/AND2/*
```

The following example gets a list of all pins, of all cells in the cell library for the target device:

```
get_lib_pins [get_libs]/*/*
```

## See Also

- [get\\_libs](#)
- [get\\_lib\\_cells](#)
- [list\\_property](#)
- [report\\_property](#)

## get\_libs

Create list of libraries.

### Syntax

```
get_libs [-regexp] [-nocase] [-filter arg] [-of_objects args]
         [-quiet] [-verbose] [patterns]
```

### Returns

List of libraries

### Usage

Name	Description
<i>[-regexp]</i>	Patterns are full regular expressions
<i>[-nocase]</i>	Perform case-insensitive matching (valid only when -regexp specified)
<i>[-filter]</i>	Filter list with expression
<i>[-of_objects]</i>	Get library of these libcells
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ patterns ]</i>	Match library names against patterns Default: *

### Categories

SDC, XDC, Object

### Description

Gets the cell library for the target device in the current design. There is a library for each device family because there are primitives that may be available in one device family but not in others.

**Note** To improve memory and performance, the `get_*` commands return a container list of a single type of objects (i.e. cells, nets, pins, 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. See [http://www.tcl.tk/man/tcl8.4/TclCmd/re\\_syntax.htm](http://www.tcl.tk/man/tcl8.4/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/tcl8.4/TclCmd/regexp.htm>.

**-nocase** - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

**-filter** *args* - Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get\_libs** based on property values on the libs. You can find the properties on an object with the **report\_property** or **list\_property** commands.

The specific operators that can be used in the filter expression are **==**, **!=**, and **=~**, as well as **&&** and **||** between filter expressions. Numeric comparison operators **<**, **>**, **<=**, and **>=** can also be used. In addition, for boolean (**bool**) type properties, the filter expression can simply evaluate the property directly as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

**-of\_objects** *arg* - Get a list of libraries of the specified object.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns **TCL\_OK** regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*patterns* - Match libraries against the specified patterns. The default pattern is the wildcard **\*** which gets a list of all libraries in the project.

## Examples

The following example gets the cell library for the target part:

```
get_libs
```

## See Also

- [get\\_lib\\_cells](#)
- [get\\_lib\\_pins](#)
- [list\\_property](#)
- [report\\_property](#)

## get\_msg\_count

Get message count.

### Syntax

```
get_msg_count [-severity arg] [-id arg] [-quiet] [-verbose]
```

### Returns

Message count

### Usage

Name	Description
<i>[-severity]</i>	Message severity to query (not valid with -id,) e.g. "ERROR" or "CRITICAL WARNING" Default: ALL
<i>[-id]</i>	Unique message id to be queried (not valid with -severity,) e.g Common-99
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution

### Categories

[Report](#)

### Description

Gets the number of messages that since the tool was invoked. This can give you an idea of how close to the message limit the tool may be getting. You can check the current message limit with the **get\_msg\_limit** command. You can change the message limit with the **set\_msg\_limit** command.

By default this command returns the message count for all messages. You can also get the count of a specific severity of message, or for a specific message ID.

### Arguments

**-id value** - is found in the PlanAhead tool in the Messages view or other reports when the message is reported. Use the specific message ID for the message of interest for use in this command.

**-severity** *value* - Specifies the severity of the message. There are five message severities:

- **ERROR** - An ERROR condition implies an issue has been encountered which will render design results unusable and cannot be resolved without user intervention.
- **{CRITICAL WARNING}** - A CRITICAL WARNING message indicates that certain input/constraints will either not be applied or are outside the best practices for a FPGA family. User action is strongly recommended.

**Note** Since this is a two word value, it must be enclosed in {}.

- **WARNING** - A WARNING message indicates that design results may be sub-optimal because constraints or specifications may not be applied as intended. User action may be taken or may be reserved.
- **INFO** - An INFO message is the same as a STATUS message, but includes a severity and message ID tag. An INFO message includes a message ID to allow further investigation through answer records if needed.
- **STATUS** - A STATUS message communicates general status of the process and feedback to the user regarding design processing. A STATUS message does not include a message ID.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

## Examples

The following example gets the message count for all messages:

```
get_msg_count -severity ALL
get_msg_count
```

**Note** Both lines return the same thing since the default is to return the count for all messages when -severity or -id is not specified.

The following example gets the message count of the specified message ID:

```
get_msg_count -id Netlist-1129
```

## See Also

- [get\\_msg\\_limit](#)
- [set\\_msg\\_limit](#)

## get\_msg\_limit

Get message limit.

### Syntax

```
get_msg_limit [-severity arg] [-id arg] [-quiet] [-verbose]
```

### Returns

Message limit

### Usage

Name	Description
<i>[-severity]</i>	Message severity to query (not valid with -id,) e.g. "ERROR" or "CRITICAL WARNING" Default: ALL
<i>[-id]</i>	Unique message id to be queried (not valid with -severity,) e.g Common-99
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution

### Categories

[Report](#)

### Description

Gets the number of messages that will be reported by the tool while invoked. When the tool reaches the defined message limit, it stops reporting messages. The default value is 4,294,967,295. This default value can be changed with the **set\_msg\_limit** command.

By default this command gets the message limit for all messages. You can also get the limit of a specific severity of message, or for a specific message ID. For instance the following are two messages returned by the PlanAhead application under different circumstances:

```
INFO: [common-99] This is an example INFO message
CRITICAL WARNING: [Netlist-1129] This message is a CRITICAL WARNING and
requires user attention
```

**Note** You can change the severity of a specific message ID with the **set\_msg\_severity** command.

### Arguments

**-id value** - is found in the PlanAhead application in the Messages view or other reports when the message is reported. Use the specific message ID for the message of interest for use in this command. The message IDs above are common-99 and Netlist-1129.

**-severity** *value* - Specifies the severity of the message. There are five message severities:

- **ERROR** - An ERROR condition implies an issue has been encountered which will render design results unusable and cannot be resolved without user intervention.
- **{CRITICAL WARNING}** - A CRITICAL WARNING message indicates that certain input/constraints will either not be applied or are outside the best practices for a FPGA family. User action is strongly recommended.

**Note** Since this is a two word value, it must be enclosed in {}.

- **WARNING** - A WARNING message indicates that design results may be sub-optimal because constraints or specifications may not be applied as intended. User action may be taken or may be reserved.
- **INFO** - An INFO message is the same as a STATUS message, but includes a severity and message ID tag. An INFO message includes a message ID to allow further investigation through answer records if needed.
- **STATUS** - A STATUS message communicates general status of the process and feedback to the user regarding design processing. A STATUS message does not include a message ID.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

## Examples

The following example returns the limit for CRITICAL WARNING messages:

```
get_msg_limit -severity {CRITICAL WARNING}
```

The default when -severity or -id is not specified is to return the limit for all messages.

The following example returns the message limit of the specified message ID:

```
get_msg_limit -id Netlist-1129
```

## See Also

- [set\\_msg\\_limit](#)
- [set\\_msg\\_severity](#)

## get\_nets

Get a list of nets in the current design.

### Syntax

```
get_nets [-hsc arg] [-hierarchical] [-regexp] [-nocase]
[-filter arg] [-of_objects args] [-match_style arg]
[-top_net_of_hierarchical_group] [-segments]
[-boundary_type arg] [-quiet] [-verbose] [patterns]
```

### Returns

List of net objects

### Usage

Name	Description
<code>[-hsc]</code>	Hierarchy separator Default: /
<code>[-hierarchical]</code>	Search level-by-level in current instance
<code>[-regexp]</code>	Patterns are full regular expressions
<code>[-nocase]</code>	Perform case-insensitive matching (valid only when -regexp specified)
<code>[-filter]</code>	Filter list with expression
<code>[-of_objects]</code>	Get nets of these pins/ports,cells,timing paths or clocks
<code>[-match_style]</code>	Style of pattern matching, valid values are ucf, sdc Default: sdc
<code>[-top_net_of_hierarchical_group]</code>	Return net segment(s) which belong(s) to the high level of a hierarchical net
<code>[-segments]</code>	Return all segments of a net across the hierarchy
<code>[-boundary_type]</code>	Return net segment connected to a hierarchical pin which resides at the same level as the pin (upper) or at the level below (lower), or both. Valid values are : upper, lower, both Default: upper
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[ patterns ]</code>	Match net names against patterns Default: *

### Categories

[SDC](#), [XDC](#), [Object](#)

### Description

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



**-boundary\_type** - (Optional) Gets the net segment at the level (upper) of a specified hierarchical pin, at the level below (lower) the pin or port, or both the level of and the level below. Valid values are upper, lower, or both. The default value is upper.

**Note** This argument must be used with the **-of\_objects** argument to specify the hierarchical pin.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*patterns* - (Optional) Match nets against the specified patterns. The default pattern is the wildcard '\*' which returns a list of all nets in the project. 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 a list of nets attached to the specified cell:

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

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

The following example returns a list of nets that have been marked for debug with the **connect\_debug\_port** command:

```
get_nets -hier -filter {MARK_DEBUG==1}
```

## See Also

- [connect\\_debug\\_port](#)
- [get\\_cells](#)
- [get\\_clocks](#)
- [get\\_pins](#)
- [get\\_ports](#)
- [list\\_property](#)
- [report\\_property](#)

## get\_nodes

Get a list of nodes in the device.

### Syntax

```
get_nodes [-regexp] [-nocase] [-filter arg] [-of_objects args]
[-uphill] [-downhill] [-flyover] [-quiet] [-verbose] [patterns]
```

### Returns

Nodes

### Usage

Name	Description
<i>[-regexp]</i>	Patterns are full regular expressions
<i>[-nocase]</i>	Perform case-insensitive matching. (valid only when -regexp specified)
<i>[-filter]</i>	Filter list with expression
<i>[-of_objects]</i>	Get the nodes of these site_pins, nodes, tiles, wires.
<i>[-uphill]</i>	Get the nodes uphill (driver) from the provided site_pin, node or in the given tile(s).
<i>[-downhill]</i>	Get the nodes downhill (loads) from the provided site_pin, node or in the given tile(s).
<i>[-flyover]</i>	Get the nodes that fly over the given tile(s).
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ patterns ]</i>	Match nodes against patterns Default: *

### Categories

Object

### Description

Returns a list of nodes on the device that match a specified search pattern in an open design.

The default command gets a list of all nodes on the device.

**Note** To improve memory and performance, the `get_*` commands return a container list of a single type of objects (i.e. cells, nets, pins, 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. See [http://www.tcl.tk/man/tcl8.4/TclCmd/re\\_syntax.htm](http://www.tcl.tk/man/tcl8.4/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/tcl8.4/TclCmd/regexp.htm>.

**-nocase** - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

**-filter** *args* - Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get\_nodes** based on property values on the nodes. You can find the properties on an object with the **report\_property** or **list\_property** commands. Any property/value pair can be used as a filter. In the case of the node object, "IS\_INPUT\_PIN", "IS\_BEL\_PIN" and "NUM\_WIRES" are some of the properties that can be used to filter results.

The specific operators that can be used in the filter expression are ==, !=, and ~, as well as && and || between filter expressions. Numeric comparison operators <, >, <=, and >= can also be used. In addition, for boolean (**bool**) type properties, the filter expression can simply evaluate the property directly as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

**-of\_objects** *args* - Return the nodes of the specified site\_pin, nodes, tiles, or wires.

**-uphill** - Return the nodes uphill of the specified site\_pin, node, or tile. Uphill nodes precede the specified object in the logic network.

**-downhill** - Return the nodes downhill of the specified site\_pin, node, or tile. Downhill nodes follow the specified object in the logic network.

**-flyover** - Return the nodes that pass through (or flyover) the specified tiles.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*patterns* - Return nodes matching the specified search patterns. The default pattern is the wildcard "\*" which gets a list of all nodes on the device. More than one search pattern can be specified to find nodes 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 returns the nodes associated with the specified tile:

```
get_nodes -of_objects [get_tiles CLBLM_R_X11Y158]
```

The following example returns the nodes downhill from the specified node:

```
get_nodes -downhill LIOB33_SING_X0Y199/IOB_PADOUT0
```

## See Also

- [get\\_nodes](#)
- [get\\_site\\_pins](#)
- [get\\_tiles](#)
- [get\\_wires](#)
- [list\\_property](#)
- [report\\_property](#)

## get\_package\_pins

Get a list of package pins.

### Syntax

```
get_package_pins [-regexp] [-nocase] [-filter arg]
[-of_objects args] [-quiet] [-verbose] [patterns]
```

### Returns

List of package pin objects

### Usage

Name	Description
<i>[-regexp]</i>	Patterns are full regular expressions
<i>[-nocase]</i>	Perform case-insensitive matching. (valid only when -regexp specified)
<i>[-filter]</i>	Filter list with expression
<i>[-of_objects]</i>	Get the list of package pin objects of these sites iobanks ports.
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ patterns ]</i>	Match list of package pin objects against patterns Default: *

### Categories

[XDC](#), [Object](#)

### Description

Gets a list of the pins on the selected package for the target device. The default command gets a list of all pins on the package.

**Note** To improve memory and performance, the `get_*` commands return a container list of a single type of objects (i.e. cells, nets, pins, 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. See [http://www.tcl.tk/man/tcl8.4/TclCmd/re\\_syntax.htm](http://www.tcl.tk/man/tcl8.4/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/tcl8.4/TclCmd/regexp.htm>.

**-nocase** - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

**-filter** *args* - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get\_package\_pins** based on property values on the pins. You can find the properties on an object with the **report\_property** or **list\_property** commands. In the case of the package pin object, "IS\_CLK\_CAPABLE", "IS\_VREF" and "IS\_GLOBAL\_CLK" are some of the properties that can be used to filter results.

The specific operators that can be used in the filter expression are ==, !=, and =~, as well as && and || between filter expressions. Numeric comparison operators <, >, <=, and >= can also be used. In addition, for boolean (**bool**) type properties, the filter expression can simply evaluate the property directly as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

**-of\_objects** *arg* - (Optional) Get the package pins connected to the specified site or iobank objects.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*patterns* - (Optional) Match pins against the specified patterns. The default pattern is the wildcard '\*' which returns all pins on the package. More than one pattern can be specified to find multiple pins based on different search criteria.

## Examples

The following example gets a list of all pins on the package of the target device:

```
get_package_pins
```

The following example gets the number of clock capable (CC) pins on the package:

```
llength [get_package_pins -filter {IS_CLK_CAPABLE==1}]
```

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

## See Also

- [get\\_iobanks](#)
- [get\\_sites](#)
- [list\\_property](#)
- [report\\_property](#)

## get\_param

Get a parameter value.

### Syntax

```
get_param [-quiet] [-verbose] name
```

### Returns

Parameter value

### Usage

Name	Description
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>name</i>	Parameter name

### Categories

[PropertyAndParameter](#)

### Description

This command gets the currently defined value for a specified tool parameter. These parameters are user-definable configuration settings that control various behaviors within the tool. Refer to **report\_param** for a description of what each parameter configures or controls.

### Arguments

*name* - 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, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

### Examples

The following example gets the current value of the specified parameter:

```
get_param tcl.statsThreshold
```

## See Also

- [list\\_param](#)
- [report\\_param](#)
- [reset\\_param](#)
- [set\\_param](#)

## get\_parts

Get a list of parts available in the software.

### Syntax

```
get_parts [-regexp] [-nocase] [-filter arg] [-quiet] [-verbose]
[patterns]
```

### Returns

List of part objects

### Usage

Name	Description
<i>[-regexp]</i>	Patterns are full regular expressions
<i>[-nocase]</i>	Perform case-insensitive matching. (valid only when -regexp specified)
<i>[-filter]</i>	Filter list with expression
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ patterns ]</i>	Match part names against patterns Default: * Values: The default search pattern is the wildcard *, or .* when -regexp is specified.

### Categories

Object

### Description

Gets a list of parts in the current project that match a specified search pattern. The default command gets a list of all parts in the project.

**Note** To improve memory and performance, the `get_*` commands return a container list of a single type of objects (i.e. cells, nets, pins, 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. See [http://www.tcl.tk/man/tcl8.4/TclCmd/re\\_syntax.htm](http://www.tcl.tk/man/tcl8.4/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/tcl8.4/TclCmd/regexp.htm>.

**-nocase** - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

**-filter** *args* - Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get\_parts** based on property values on the parts. You can find the properties on an object with the **report\_property** or **list\_property** commands. Any property/value pair can be used as a filter. In the case of the part object, "DEVICE", "FAMILY" and "SPEED" are some of the properties that can be used to filter results.

The specific operators that can be used in the filter expression are **==**, **!=**, and **=~**, as well as **&&** and **||** between filter expressions. Numeric comparison operators **<**, **>**, **<=**, and **>=** can also be used. In addition, for boolean (**bool**) type properties, the filter expression can simply evaluate the property directly as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns **TCL\_OK** regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*patterns* - Match parts against the specified patterns. The default pattern is the wildcard **"\*"** which gets a list of all parts. More than one search pattern can be specified to find parts 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 7vx485t parts, speed grade -1:

```
get_parts -filter {DEVICE == 7vx485t && speed == -1}
```

The following example gets the number of 7 series and 6 series Virtex parts:

```
llength [get_parts -regexp {xc7v.* xc6v.*} -nocase]
```

**Note** If there are no parts matching the pattern the PlanAhead tool will return a warning indicating that no parts matched the specified pattern.

## See Also

- [list\\_property](#)
- [report\\_property](#)

## get\_pblocks

Get a list of pblocks in the current design.

### Syntax

```
get_pblocks [-regexp] [-nocase] [-filter arg] [-of_objects args]
[-quiet] [-verbose] [patterns]
```

### Returns

List of pblock objects

### Usage

Name	Description
<i>[-regexp]</i>	Patterns are full regular expressions
<i>[-nocase]</i>	Perform case-insensitive matching (valid only when -regexp specified)
<i>[-filter]</i>	Filter list with expression
<i>[-of_objects]</i>	Get pblocks of these cells
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ patterns ]</i>	Match pblock names against patterns Default: *

### Categories

Object, Floorplan, XDC

### Description

Gets a list of Pblocks defined in the current project that match a specific pattern. The default command gets a list of all Pblocks in the project.

**Note** To improve memory and performance, the `get_*` commands return a container list of a single type of objects (i.e. cells, nets, pins, 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. See [http://www.tcl.tk/man/tcl8.4/TclCmd/re\\_syntax.htm](http://www.tcl.tk/man/tcl8.4/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/tcl8.4/TclCmd/regexp.htm>.

**-nocase** - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

**-filter** *args* - Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get\_pblocks** based on property values on the Pblocks. You can find the properties on an object with the **report\_property** or **list\_property** commands. In the case of the Pblock object, "NAME" and "GRID\_RANGES" are some of the properties that can be used to filter results.

The specific operators that can be used in the filter expression are **==**, **!=**, and **=~**, as well as **&&** and **||** between filter expressions. Numeric comparison operators **<**, **>**, **<=**, and **>=** can also be used. In addition, for boolean (**bool**) type properties, the filter expression can simply evaluate the property directly as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

**-of\_objects** *arg* - Get the Pblocks to which the specified cells are assigned.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns **TCL\_OK** regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*patterns* - Match Pblocks against the specified patterns. The default pattern is the wildcard **'\*'** which returns all Pblocks in the project.

## Examples

The following example gets a list of all Pblocks in the current project:

```
get_pblocks
```

## See Also

- [create\\_pblock](#)
- [get\\_cells](#)

## get\_pins

Get a list of pins in the current design.

### Syntax

```
get_pins [-hsc arg] [-hierarchical] [-regexp] [-nocase] [-leaf]
[-filter arg] [-of_objects args] [-match_style arg] [-quiet]
[-verbose] [patterns]
```

### Returns

List of pin objects

### Usage

Name	Description
<code>[-hsc]</code>	Hierarchy separator Default: /
<code>[-hierarchical]</code>	Search level-by-level in current instance
<code>[-regexp]</code>	Patterns are full regular expressions
<code>[-nocase]</code>	Perform case-insensitive matching (valid only when -regexp specified)
<code>[-leaf]</code>	Get leaf/global pins of nets with -of_objects
<code>[-filter]</code>	Filter list with expression
<code>[-of_objects]</code>	Get pins of these cells, nets, timing paths or clocks
<code>[-match_style]</code>	Style of pattern matching, valid values are ucf, sdc Default: sdc
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[ patterns ]</code>	Match pin names against patterns Default: *

### Categories

SDC, XDC, Object

### 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.

**Note** To improve memory and performance, the `get_*` commands return a container list of a single type of objects (i.e. cells, nets, pins, 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

**-hsc** *arg* - (Optional) The default hierarchy separator is '/'. Use this argument to specify a different hierarchy separator.

**-hierarchical** - (Optional) Get pins from all levels of the design hierarchy. Without this argument, the command will only get pins from the top of the design hierarchy. When using **-hierarchical**, the search pattern should not contain a hierarchy separator because the search pattern is applied at each level of the hierarchy, not to the full hierarchical cell name. For instance, searching for U1/\* searches each level of the hierarchy for instances with U1/ in the name. This may not return the intended results. See **get\_cells** for examples of **-hierarchical** searches.

**Note** When used with **-regexpr**, the specified search string is matched against the full hierarchical name, and the U1/\* search pattern will work as intended

**-regexpr** - (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. See [http://www.tcl.tk/man/tcl8.4/TclCmd/re\\_syntax.htm](http://www.tcl.tk/man/tcl8.4/TclCmd/re_syntax.htm) for help with regular expression syntax.

**Note** The Tcl built-in command **regexpr** is not anchored, and works as a standard Tcl command. For more information refer to <http://www.tcl.tk/man/tcl8.4/TclCmd/regexpr.htm>.

**-nocase** - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexpr** only.

**-leaf** - (Optional) Include leaf pins for the parameters specified with the **-of\_object** argument

**-filter args** - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get\_pins** based on property values on the pins. You can find the properties on an object with the **report\_property** or **list\_property** commands. In the case of the pins object, "PARENT" and "TYPE" are some of the properties that can be used to filter results.

The specific operators that can be used in the filter expression are ==, !=, and =~, as well as && and || between filter expressions. Numeric comparison operators <, >, <=, and >= can also be used. In addition, for boolean (**bool**) type properties, the filter expression can simply evaluate the property directly as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

**-of\_objects arg** - (Optional) Get the pins connected to the specified cell, pin, port, or clock.

**-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, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** 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 {} to present the list as a single element.

## Examples

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

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

**Note** If there are no pins matching the pattern the PlanAhead tool will return a warning indicating that no pins matched the specified pattern.

## See Also

- [list\\_property](#)
- [report\\_property](#)

## get\_pips

Get a list of programmable interconnect points (pips) on the current device.

### Syntax

```
get_pips [-regexp] [-nocase] [-filter arg] [-of_objects args]
[-uphill] [-downhill] [-quiet] [-verbose] [patterns]
```

### Returns

Pips

### Usage

Name	Description
<i>[-regexp]</i>	Patterns are full regular expressions
<i>[-nocase]</i>	Perform case-insensitive matching. (valid only when -regexp specified)
<i>[-filter]</i>	Filter list with expression
<i>[-of_objects]</i>	Get the pips of these sites, tiles, wires, or pips.
<i>[-uphill]</i>	Get the pips uphill from the provided wire or pip.
<i>[-downhill]</i>	Get the pips downhill from the provided wire or pip.
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ patterns ]</i>	Match pips against patterns Default: *

### Categories

Object

### Description

Programmable interconnect points, or PIPs, provide the physical routing paths on the device used to connect logic networks. This command returns a list of PIPs on the device that match a specified search pattern. The command requires a design to be open.

The default command gets a list of all PIPs on the device. However, this is not a recommended use of the command due to the number of pips on a device. You should specify the **-of\_objects** argument to limit the number of pips returned.

**Note** To improve memory and performance, the `get_*` commands return a container list of a single type of objects (i.e. cells, nets, pins, 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. See [http://www.tcl.tk/man/tcl8.4/TclCmd/re\\_syntax.htm](http://www.tcl.tk/man/tcl8.4/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/tcl8.4/TclCmd/regexp.htm>.

**-nocase** - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

**-filter** *args* - Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get\_pips** based on property values on the PIPs. You can find the properties on an object with the **report\_property** or **list\_property** commands. Any property/value pair can be used as a filter. In the case of the PIP object, "IS\_DIRECTIONAL" and "FROM\_PIN" are two of the properties that can be used to filter results.

The specific operators that can be used in the filter expression are ==, !=, and =~, as well as && and || between filter expressions. Numeric comparison operators <, >, <=, and >= can also be used. In addition, for boolean (**bool**) type properties, the filter expression can simply evaluate the property directly as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

**-of\_objects** *args* - Return the PIPs of the specified site, tile, or wire objects.

**Note** You are recommended to always use the **-of\_objects** argument to limit the runtime and memory used by the **get\_pips** command. The number of programmable interconnect points returned can be considerable

**-uphill** - Return the PIPs uphill of the specified wire or PIPs. Uphill PIPs precede the specified object in the logic network.

**-downhill** - Return the PIPs downhill of the specified wire or PIPs. Downhill PIPs follow the specified object in the logic network.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*patterns* - Return PIPs matching the specified search patterns. The default pattern is the wildcard "\*" which gets a list of all PIPs on the device. More than one search pattern can be specified to find PIPs 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 returns the PIPs associated with the specified tile:

```
get_pips -of_object [get_tiles DSP_R_X9Y75]
```

## See Also

- [get\\_sites](#)
- [get\\_tiles](#)
- [get\\_wires](#)
- [list\\_property](#)
- [report\\_property](#)

## get\_ports

Get a list of ports in the current design.

### Syntax

```
get_ports [-regexp] [-nocase] [-filter arg] [-of_objects args]
          [-match_style arg] [-quiet] [-verbose] [patterns]
```

### Returns

List of port objects

### Usage

Name	Description
<i>[-regexp]</i>	Patterns are full regular expressions
<i>[-nocase]</i>	Perform case-insensitive matching (valid only when -regexp specified)
<i>[-filter]</i>	Filter list with expression
<i>[-of_objects]</i>	Get ports of these nets, timing paths, clocks
<i>[-match_style]</i>	Style of pattern matching, valid values are ucf, sdc Default: sdc
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ patterns ]</i>	Match port names against patterns Default: *

### Categories

[SDC](#), [XDC](#), [Object](#)

### Description

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

**Note** To improve memory and performance, the `get_*` commands return a container list of a single type of objects (i.e. cells, nets, pins, 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. See [http://www.tcl.tk/man/tcl8.4/TclCmd/re\\_syntax.htm](http://www.tcl.tk/man/tcl8.4/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/tcl8.4/TclCmd/regexp.htm>.

**-nocase** - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

**-filter** *args* - (Optional) Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by `get_ports` based on property values on the ports. You can find the properties on an object with the **report\_property** or **list\_property** commands. In the case of the "ports" object, "PARENT" and "TYPE" are some of the properties that can be used to filter results.

The specific operators that can be used in the filter expression are `==`, `!=`, and `~=`, as well as `&&` and `||` between filter expressions. Numeric comparison operators `<`, `>`, `<=`, and `>=` can also be used. In addition, for boolean (**bool**) type properties, the filter expression can simply evaluate the property directly as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

**-of\_objects** *arg* - Get the ports connected to the specified cell, net, clock, or timing path objects.

**-match\_style** [**sd** | **ucf**] - Indicates that the search pattern matches UCF constraints or SDC constraints. The default is SDC.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns `TCL_OK` regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*patterns* - Match ports against the specified patterns. The default pattern is the wildcard "\*" which gets a list of all ports in the project. 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 a list of pins attached to the specified cell:

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

**Note** If there are no ports matching the pattern the PlanAhead tool will return a warning indicating that no ports matched the specified pattern.

## See Also

- [list\\_property](#)
- [report\\_property](#)

## get\_projects

Get a list of projects.

### Syntax

```
get_projects [-regexp] [-nocase] [-filter arg] [-quiet]
[-verbose] [patterns]
```

### Returns

List of project objects

### Usage

Name	Description
<i>[-regexp]</i>	Patterns are full regular expressions
<i>[-nocase]</i>	Perform case-insensitive matching (valid only when -regexp specified)
<i>[-filter]</i>	Filter list with expression
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ patterns ]</i>	Match project names against patterns Default: *

### Categories

[Object](#), [Project](#)

### Description

Gets a list of open projects that match the specified search pattern. The default gets a list of all open projects.

**Note** To improve memory and performance, the `get_*` commands return a container list of a single type of objects (i.e. cells, nets, pins, 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. See [http://www.tcl.tk/man/tcl8.4/TclCmd/re\\_syntax.htm](http://www.tcl.tk/man/tcl8.4/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/tcl8.4/TclCmd/regexp.htm>.

**-nocase** - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

**-filter** *args* - Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get\_projects** based on property values on the projects. You can find the properties on an object with the **report\_property** or **list\_property** commands. In the case of the "projects" object, "NAME", "DIRECTORY" and "TARGET\_LANGUAGE" are some of the properties that can be used to filter results.

The specific operators that can be used in the filter expression are **==**, **!=**, and **=~**, as well as **&&** and **||** between filter expressions. Numeric comparison operators **<**, **>**, **<=**, and **>=** can also be used. In addition, for boolean (**bool**) type properties, the filter expression can simply evaluate the property directly as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns **TCL\_OK** regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*patterns* - Match projects against the specified patterns. The default pattern is the wildcard **"\*"** which gets a list of all parts. More than one pattern can be specified to find multiple projects based on different search criteria.

## Examples

The following example gets a list of all open projects.

```
get_projects
```

The following example sets a variable called **project\_found** to the length of the list of projects returned by **get\_projects**, then prints either that projects were found or were not found as appropriate:

```
set project_found [llength [get_projects ISC*] ]
if {$project_found > 0} {puts "Project Found."} else {puts "No Projects Found."}
```

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

## See Also

- [create\\_project](#)
- [current\\_project](#)
- [open\\_project](#)

## get\_property

Get properties of object.

### Syntax

```
get_property [-quiet] [-verbose] name object
```

### Returns

Property value

### Usage

Name	Description
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>name</i>	Name of property whose value is to be retrieved
<i>object</i>	Object to query for properties

### Categories

[Object](#), [PropertyAndParameter](#)

### Description

Gets the current value of the named property from the specified object. If the property is not currently assigned to the object, or is assigned without a value, then the **get\_property** command returns nothing.

### Arguments

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*name* - The name of the property to be returned. The name is not case sensitive.

*object* - The object to query.

### Examples

The following example gets the NAME property from the specified cell:

```
get_property NAME [lindex [get_cells] 3]
```

## See Also

- [create\\_property](#)
- [get\\_cells](#)
- [list\\_property](#)
- [list\\_property\\_value](#)
- [report\\_property](#)
- [reset\\_property](#)
- [set\\_property](#)

## get\_reconfig\_modules

Get a list of Reconfigurable Modules in the current project.

### Syntax

```
get_reconfig_modules [-regexp] [-nocase] [-filter arg]
[-of_objects args] [-quiet] [-verbose] [patterns]
```

### Returns

List of reconfigurable module objects

### Usage

Name	Description
<i>[-regexp]</i>	Patterns are full regular expressions
<i>[-nocase]</i>	Perform case-insensitive matching (valid only when -regexp specified)
<i>[-filter]</i>	Filter list with expression
<i>[-of_objects]</i>	Get Reconfigurable Modules for these cells
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ patterns ]</i>	Match Reconfigurable Module names against patterns

### Categories

[Object](#), [PartialReconfiguration](#)

### Description

Gets a list of reconfigurable modules in the current project that match a specified search pattern. The default command gets a list of all reconfigurable modules in the project.

**Note** To improve memory and performance, the `get_*` commands return a container list of a single type of objects (i.e. cells, nets, pins, 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. See [http://www.tcl.tk/man/tcl8.4/TclCmd/re\\_syntax.htm](http://www.tcl.tk/man/tcl8.4/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/tcl8.4/TclCmd/regexp.htm>.

**-filter** *args* - Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by `get_reconfig_modules` based on property values on the modules. You can find the properties on an object with the **report\_property** or **list\_property** commands.

The specific operators that can be used in the filter expression are `==`, `!=`, and `=~`, as well as `&&` and `||` between filter expressions. Numeric comparison operators `<`, `>`, `<=`, and `>=` can also be used. In addition, for boolean (**bool**) type properties, the filter expression can simply evaluate the property directly as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

**-of\_object** *arg* - Get reconfigurable modules from the specified object.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns `TCL_OK` regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*patterns* - Match reconfigurable modules against the specified patterns. The default pattern is the wildcard `*` which gets a list of all modules in the project.

## Examples

The following example gets a list of all reconfigurable modules in the project:

```
get_reconfig_modules
```

## See Also

- [create\\_reconfig\\_module](#)
- [list\\_property](#)
- [report\\_property](#)

## get\_runs

Get a list of runs.

### Syntax

```
get_runs [-regexp] [-nocase] [-filter arg] [-quiet] [-verbose]
[patterns]
```

### Returns

List of run objects

### Usage

Name	Description
<i>[-regexp]</i>	Patterns are full regular expressions
<i>[-nocase]</i>	Perform case-insensitive matching (valid only when -regexp specified)
<i>[-filter]</i>	Filter list with expression
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ patterns ]</i>	Match run names against patterns Default: *

### Categories

[Object](#), [Project](#)

### Description

Gets a list of synthesis and implementation runs in the current project that match a specified search pattern. The default command gets a list of all runs defined in the project.

**Note** To improve memory and performance, the `get_*` commands return a container list of a single type of objects (i.e. cells, nets, pins, 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. See [http://www.tcl.tk/man/tcl8.4/TclCmd/re\\_syntax.htm](http://www.tcl.tk/man/tcl8.4/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/tcl8.4/TclCmd/regexp.htm>.

**-nocase** - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

**-filter** *args* - Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get\_runs** based on property values on the runs. You can find the properties on an object with the **report\_property** or **list\_property** commands. Any property/value pair can be used as a filter. In the case of the runs object, "CONSTRSET", "IS\_IMPLEMENTATION", "IS\_SYNTHESIS", and "FLOW" are some of the properties that can be used to filter results.

The specific operators that can be used in the filter expression are ==, !=, and =~, as well as && and || between filter expressions. Numeric comparison operators <, >, <=, and >= can also be used. In addition, for boolean (**bool**) type properties, the filter expression can simply evaluate the property directly as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*patterns* - Match run names against the specified patterns. The default pattern is the wildcard '\*' which gets a list of all defined runs in the project. More than one pattern can be specified to find multiple runs based on different search criteria.

## Examples

The following example gets a list of all incomplete runs in the current project:

```
get_runs -filter {PROGRESS < 100}
```

The following example gets a list of runs matching the specified pattern:

```
get_runs imp*
```

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

## See Also

[report\\_property](#)

## get\_selected\_objects

Get selected objects.

### Syntax

```
get_selected_objects [-primary] [-quiet] [-verbose]
```

### Returns

List of selected objects

### Usage

Name	Description
<i>[-primary]</i>	Do not include objects that were selected due to selection rules
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution

### Categories

[Object](#), [GUIControl](#)

### Description

Gets the objects currently selected by the **select\_objects** command. Can get the primary selected object and any secondary selected objects as well.

Primary objects are directly selected, while secondary objects are selected based on the selection rules currently defined by the **Tools > Options** command. Refer to the *PlanAhead User Guide* (ug632) for more information on setting selection rules.

### Arguments

**-primary** - Indicates that only the primary selected object or objects should be returned; not secondary objects. As a default **get\_selected\_objects** will return all currently selected objects.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

### Examples

The following example reports the properties of all currently selected objects, both primary and secondary:

```
report_property [get_selected_objects]
```

## See Also

[select\\_objects](#)

## get\_site\_pins

Get a list of site\_pins.

### Syntax

```
get_site_pins [-regexp] [-nocase] [-filter arg]
              [-of_objects args] [-quiet] [-verbose] [patterns]
```

### Returns

Site\_pins

### Usage

Name	Description
<i>[-regexp]</i>	Patterns are full regular expressions
<i>[-nocase]</i>	Perform case-insensitive matching. (valid only when -regexp specified)
<i>[-filter]</i>	Filter list with expression
<i>[-of_objects]</i>	Get the site_pins of these sites, bels, routedSites or logical cell pins.
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ patterns ]</i>	Match site_pins against patterns Default: *

### Categories

Object

### Description

Returns a list of site pins of the specified site, BELs, or logical cell pin objects in an open design. This command requires the use of the **-of\_objects** argument.

**Note** To improve memory and performance, the `get_*` commands return a container list of a single type of objects (i.e. cells, nets, pins, 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. See [http://www.tcl.tk/man/tcl8.4/TclCmd/re\\_syntax.htm](http://www.tcl.tk/man/tcl8.4/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/tcl8.4/TclCmd/regexp.htm>.

**-nocase** - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

**-filter** *args* - Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get\_site\_pins** based on property values on the site pins. You can find the properties on an object with the **report\_property** or **list\_property** commands. Any property/value pair can be used as a filter. In the case of the site pin object, "IS\_CLOCK", "IS\_DATA" and "IS\_PART\_OF\_BUS" are some of the properties that can be used to filter results.

The specific operators that can be used in the filter expression are ==, !=, and =~, as well as && and || between filter expressions. Numeric comparison operators <, >, <=, and >= can also be used. In addition, for boolean (**bool**) type properties, the filter expression can simply evaluate the property directly as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

**-of\_objects** *args* - Return the site pins of specified site, routed sites, BELs, or logical cell pin objects.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

## Examples

The following example returns the site\_pins of the specified BELs:

```
get_site_pins -of_objects [get_bels SLICE_X21Y92/B5FF]
CE CK D SR Q
get_site_pins -of_objects [get_bels SLICE_X21Y92/A5LUT]
A1 A2 A3 A4 A5 O5
get_site_pins -of_objects [get_bels SLICE_X21Y92/CARRY4_CMUX]
0 1 S0 OUT
```

The following example returns the site\_pins associated with the specified site:

```
get_site_pins -of_objects [get_sites SLICE_X21Y92]
A1 A2 A3 A4 A5 A6 AX B1 B2 B3 B4 B5 B6 BX C1 C2 C3 C4 C5 C6 CE CIN CLK CX \
D1 D2 D3 D4 D5 D6 DX SR A AMUX AQ B BMUX BQ C CMUX COUT CQ D DMUX DQ
```

## See Also

- [get\\_bels](#)
- [get\\_sites](#)
- [list\\_property](#)
- [report\\_property](#)

## get\_sites

Get a list of Sites.

### Syntax

```
get_sites [-regexp] [-filter arg] [-range args]
          [-of_objects args] [-quiet] [-verbose] [patterns]
```

### Returns

List of site objects

### Usage

Name	Description
<i>[-regexp]</i>	Patterns are full regular expressions
<i>[-filter]</i>	Filter list with expression
<i>[-range]</i>	Match site names which fall into the range. Range is defined by exactly two site names.
<i>[-of_objects]</i>	Get the sites of the objects passed in here.
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ patterns ]</i>	Match site names against patterns. Bonded sites will also match on package pin names. Default: *

### Categories

[XDC](#), [Object](#)

### Description

Gets a list of sites on the target device that match a specified search pattern. The default command gets a list of all sites on the target device.

**Note** To improve memory and performance, the `get_*` commands return a container list of a single type of objects (i.e. cells, nets, pins, 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. See [http://www.tcl.tk/man/tcl8.4/TclCmd/re\\_syntax.htm](http://www.tcl.tk/man/tcl8.4/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/tcl8.4/TclCmd/regexp.htm>.

**-nocase** - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

**-filter** *args* - Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get\_sites** based on property values on the sites. You can find the properties on an object with the **report\_property** or **list\_property** commands. In the case of the site object, "SITE\_TYPE", "IS\_OCCUPIED", "NUM\_INPUTS", and "NUM\_OUTPUTS" are some of the properties that can be used to filter results.

The specific operators that can be used in the filter expression are **==**, **!=**, and **=~**, as well as **&&** and **||** between filter expressions. Numeric comparison operators **<**, **>**, **<=**, and **>=** can also be used. In addition, for boolean (**bool**) type properties, the filter expression can simply evaluate the property directly as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

**-range** *arg* - Get all the sites that fall into a specified range. The range of sites must be specified with two site values, of the same SITE\_TYPE, such as {Slice\_X2Y12 Slice\_X3Y15}. The SITE\_TYPE of a site can be determined by the **report\_property** command.

**Note** Specifying a range with two different types will result in an error

**-of\_objects** *arg* - Get sites from the specified object or objects. Only Tile, Bel, Pin, Package Pin, and I/O Banks are supported objects.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*patterns* - Match sites against the specified patterns. The default pattern is the wildcard "\*" which gets a list of all sites on the target device.

## Examples

The following example gets a list of all sites available on the target device:

```
get_sites
```

The following example gets the number of unoccupied sites on the device:

```
llength [get_sites -filter {IS_OCCUPIED==0}]
```

**Note** If no sites match the pattern you will get a warning.

The following example gets all of the sites on the device, and returns the unique SITE\_TYPES:

```
set sites [get_sites]
foreach x $sites {
    set prop [get_property SITE_TYPE $x]
    if { [lsearch -exact $type $prop] == -1 } { lappend type $prop }
}
foreach y $type { puts "SITE_TYPE: $y" }
```

The following example gets the sites within the specified range of sites:

```
get_sites -range {Slice_X0Y0 Slice_X1Y1}  
SLICE_X0Y0 SLICE_X0Y1 SLICE_X1Y0 SLICE_X1Y1  
get_sites -range Slice_X0Y0 -range Slice_X1Y1  
SLICE_X0Y0 SLICE_X0Y1 SLICE_X1Y0 SLICE_X1Y1
```

**Note** The two methods of specifying the range of sites above return the same results.

## See Also

- [get\\_cells](#)
- [get\\_property](#)
- [list\\_property](#)
- [report\\_property](#)

## get\_tiles

Get a list of tiles.

### Syntax

```
get_tiles [-regexp] [-nocase] [-filter arg] [-of_objects args]
[-quiet] [-verbose] [patterns]
```

### Returns

Tiles

### Usage

Name	Description
<i>[-regexp]</i>	Patterns are full regular expressions
<i>[-nocase]</i>	Perform case-insensitive matching. (valid only when -regexp specified)
<i>[-filter]</i>	Filter list with expression
<i>[-of_objects]</i>	Get the tiles of these sites, bels, site_pins, nodes, wires, pips, nets.
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ patterns ]</i>	Match tiles against patterns Default: *

### Categories

Object

### Description

This command returns a list of tiles on the device in an open design. The default command gets a list of all tiles on the device.

**Note** To improve memory and performance, the `get_*` commands return a container list of a single type of objects (i.e. cells, nets, pins, 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. See [http://www.tcl.tk/man/tcl8.4/TclCmd/re\\_syntax.htm](http://www.tcl.tk/man/tcl8.4/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/tcl8.4/TclCmd/regexp.htm>.

**-nocase** - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

**-filter** *args* - Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get\_tiles** based on property values on the tile objects. You can find the properties on an object with the **report\_property** or **list\_property** commands. Any property/value pair can be used as a filter. In the case of the tile object, "NUM\_ARCS", "NUM\_SITES", and "IS\_GT\_SITE\_TILE" are some of the properties that can be used to filter results.

The specific operators that can be used in the filter expression are ==, !=, and =~, as well as && and || between filter expressions. Numeric comparison operators <, >, <=, and >= can also be used. In addition, for boolean (**bool**) type properties, the filter expression can simply evaluate the property directly as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

**-of\_objects** *args* - Can be used to return the tiles associated with specified sites, BELs, site\_pins, nodes, wires, and PIPs.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*patterns* - Match tiles against the specified patterns. The default pattern is the wildcard '\*' which gets a list of all tiles on the device. More than one search pattern can be specified to find tiles 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 returns the total number of tiles where the number of timing arcs is greater than 100 and 150 respectively:

```
llength [get_tiles -filter {NUM_ARCS>100} ]
13468
llength [get_tiles -filter {NUM_ARCS>150} ]
11691
```

## See Also

- [get\\_bels](#)
- [get\\_nodes](#)
- [get\\_pips](#)
- [get\\_site\\_pins](#)
- [get\\_sites](#)
- [get\\_wires](#)
- [list\\_property](#)
- [report\\_property](#)

## get\_wires

Get a list of wires.

### Syntax

```
get_wires [-regexp] [-nocase] [-filter arg] [-of_objects args]
          [-quiet] [-verbose] [patterns]
```

### Returns

Wires

### Usage

Name	Description
<i>[-regexp]</i>	Patterns are full regular expressions
<i>[-nocase]</i>	Perform case-insensitive matching. (valid only when -regexp specified)
<i>[-filter]</i>	Filter list with expression
<i>[-of_objects]</i>	Get the wires of these tiles, nodes, pips.
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ patterns ]</i>	Match wires against patterns Default: *

### Categories

[Object](#)

### Description

Returns a list of wires in the design that match a specified search pattern in an open design.

The default command gets a list of all wires in the design.

**Note** To improve memory and performance, the `get_*` commands return a container list of a single type of objects (i.e. cells, nets, pins, 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. See [http://www.tcl.tk/man/tcl8.4/TclCmd/re\\_syntax.htm](http://www.tcl.tk/man/tcl8.4/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/tcl8.4/TclCmd/regexp.htm>.

**-nocase** - (Optional) Perform case-insensitive matching when a pattern has been specified. This argument applies to the use of **-regexp** only.

**-filter** *args* - Filter the results list with the specified expression. The **-filter** argument filters the list of objects returned by **get\_wires** based on property values on the wires. You can find the properties on an object with the **report\_property** or **list\_property** commands. Any property/value pair can be used as a filter. In the case of the wire object, "NAME", "NUM\_DOWNHILL\_PIPS" and "NUM\_UPHILL\_PIPS" are some of the properties that can be used to filter results.

The specific operators that can be used in the filter expression are ==, !=, and =~, as well as && and || between filter expressions. Numeric comparison operators <, >, <=, and >= can also be used. In addition, for boolean (**bool**) type properties, the filter expression can simply evaluate the property directly as true or not true:

```
-filter {IS_PRIMITIVE && !IS_LOC_FIXED}
```

**-of\_objects** *args* - Return the nodes of the specified nodes, PIPs, or tiles.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*patterns* - Return wires matching the specified search patterns. The default pattern is the wildcard "\*" which gets a list of all wires in the design. More than one search pattern can be specified to find wires 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 returns the wires associated with the specified tile:

```
get_wires -of_objects [get_tiles IO_INT_INTERFACE_L_X0Y198]
```

## See Also

- [get\\_nodes](#)
- [get\\_pips](#)
- [get\\_tiles](#)
- [list\\_property](#)
- [report\\_property](#)

## help

Display help for one or more topics.

### Syntax

```
help [-category arg] [-args] [-syntax] [-long] [-quiet]
[-verbose] [pattern]
```

### Returns

Nothing

### Usage

Name	Description
<i>[-category]</i>	Search for topics in the specified category
<i>[-args]</i>	Display arguments description
<i>[-syntax]</i>	Display syntax description
<i>[-long]</i>	Display long help description
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ pattern ]</i>	Display help for topics that match the specified pattern Default: *

### Categories

[Project](#)

### Description

Returns a brief or long description of the specified Tcl command; a list of available Xilinx Tcl command categories; or a list of commands matching a specific pattern.

The default **help** command without any arguments returns a list of Tcl command categories that can be further explored. Command categories are groups of commands performing a specific function, like File I/O commands for instance.

Different arguments for the **help** command can return just the command syntax for a quick reminder of how the command should be structured; the command syntax and a brief description of each argument; or the long form of the command with more detailed descriptions and examples of the command.

### Arguments

**-category** *arg* - (Optional) Get a list of the commands grouped under the specified command category.

**-syntax** - (Optional) Returns only the syntax line for the command as a quick reminder.

**-args** - (Optional) Get abbreviated help text for the specified command. The default is to return the detailed help for the specified command. Use this argument to keep it brief.

**-long** - (Optional) Returns the extended help description for the command, including the syntax, a brief description of the arguments, and a more detailed description of the command with examples. This is the default setting for the help command.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

**pattern** - (Optional) Returns information related to the specified command, or a list of commands that match the specified pattern.

## Examples

The following example returns a list of Xilinx Tcl command categories:

```
help
```

The following example returns a list of all commands matching the specified search pattern:

```
help *file*
```

This list can be used to quickly locate a command for a specific purpose, such as **remove\_files** or **delete\_files**.

The following help command returns a long description of the **remove\_files** command and its arguments:

```
help remove_files
```

**Note** You can also use the **-args** option to get a brief description of the command.

The following example defines a procedure called **short**, and returns the **-args** form of help for the specified command:

```
proc short cmdName {help -args $cmdName}
```

**Note** You can add this procedure to the `$HOME/.Xilinx/PlanAhead/init.tcl` file to load this command every time the tool is launched. Refer to *Chapter 1, Introduction* for more information on the `init.tcl` file

## highlight\_objects

Highlight objects in different colors.

### Syntax

```
highlight_objects [-color_index arg] [-rgb args] [-color arg]
[-quiet] [-verbose] objects
```

### Returns

Nothing

### Usage

Name	Description
<code>[-color_index]</code>	Color index
<code>[-rgb]</code>	RGB color index list
<code>[-color]</code>	Valid values are red green blue magenta yellow cyan and orange
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>objects</code>	Objects to highlight

### Categories

[GUIControl](#)

### Description

Highlights the specified object or objects in a color as determined by one of the color options. Objects can be unhighlighted with the **unhighlight\_objects** command.

**Note** Only one color option should be used to specify the highlight color. However, if more than one color option is specified, the order of precedence used to define the color is -rgb, -color\_index, and -color

### Arguments

**-color\_index** arg - The color index to use for highlighting the selected object or objects. The color index is defined by the Highlight category of the Tools > Options > Themes command. Refer to the *PlanAhead User Guide* (ug632) for more information on setting themes.

**-rgb** args - The color to use in the form of an RGB code specified as {R G B}. For instance, {255 255 0} specifies the color yellow.

**-color** arg - The color to use for highlighting the specified object or objects. Supported highlight colors are: red, green, blue, magenta, yellow, cyan, and orange.

**Note** White is the color used to display selected objects with the **select\_objects** command

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*objects* - Specifies one or more objects to be highlighted.

## Examples

The following example highlights the currently selected objects in the color red:

```
highlight_objects -color red [get_selected_objects]
```

## See Also

- [get\\_selected\\_objects](#)
- [unhighlight\\_objects](#)

## implement\_debug\_core

Implement ChipScope debug core.

### Syntax

```
implement_debug_core [-quiet] [-verbose] [cores ...]
```

### Returns

Nothing

### Usage

Name	Description
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ cores ]</i>	ChipScope debug core

### Categories

[ChipScope](#)

### Description

Implements the ChipScope debug core in the CORE Generator tool. The CORE Generator tool will be run once for any ILA debug cores specified, and run one more time for the ICON controller core if all cores are specified. The ILA core is the only core type currently supported by the `create_debug_core` command. The PlanAhead tool automatically adds an ICON controller core to contain and configure the ILA cores in the project.

The PlanAhead tool creates ChipScope ICON and ILA cores initially as black boxes. These cores must be implemented prior to running through place and route. After the core is created with `create_debug_core`, and while ports are being added and connected with `create_debug_port` and `connect_debug_port`, the content of the debug core is not defined or visible within the design.

ChipScope debug core implementation is automatic when you launch an implementation run using the `launch_runs` command. However, you can also use the `implement_debug_core` command to implement one or more of the cores in the CORE Generator tool without having to implement the whole design.

### Arguments

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the `set_msg_limit` command.

*cores* - One or more debug cores to implement.

## Examples

The following example implements all ChipScope debug cores in the current project:

```
implement_debug_core [get_debug_cores]
```

This results in both the ICON and ILA type cores being implemented in the CORE Generator tool for inclusion in the Netlist Design.

## See Also

- [connect\\_debug\\_port](#)
- [create\\_debug\\_core](#)
- [create\\_debug\\_port](#)
- [get\\_debug\\_cores](#)
- [launch\\_runs](#)

## import\_as\_run

Import an NCD and an optional TWX as a run.

### Syntax

```
import_as_run [-run arg] [-twx arg] [-pcf arg] [-quiet]
[-verbose] ncd
```

### Returns

Nothing

### Usage

Name	Description
<code>[-run]</code>	Import the results into this run
<code>[-twx]</code>	TWX file to import
<code>[-pcf]</code>	PCF file to import
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>ncd</code>	Routed NCD file to import

### Categories

Project

### Description

Imports an NCD and an optional TWX into an implementation run in the current project. This command is one of the steps involved in importing a previously placed and routed design from ISE into the PlanAhead tool.

### Arguments

**-run** *arg* - The name of the implementation run to be imported into.

**Note** This command does not create a run, but rather imports the required NCD file, and an optional TWX file if specified, into the specified run and sets its state to implemented.

**-twx** *arg* - The path and file name of an optional TRACE Timing file (.TWX) which can be imported along with the placement and routing data found in the .NCD file.

**-pcf** *arg* - A Physical Constraint File to load. Logical constraints in the NGD file are read by MAP. MAP uses some of the constraints to map the design and converts logical constraints to physical constraints and writes them to a Physical Constraints File (PCF).

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the `set_msg_limit` command.

*ncd* - The path and name of the NCD file to import into the specified implementation run.

## Examples

The following example creates a new empty RTL source project; changes the `design_mode` property to `GateLvl`, or netlist entry project; specifies the EDIF file representing the top-module of the design; and finally imports an NCD file into the implementation run which was initially created with the project.

```
create_project myProject C:/Data/myProject
set_property design_mode GateLvl [current_fileset]
set_property edif_top_file C:/Data/ise/drp_des/drp_demo.ngc [current_fileset]
import_as_run -run impl_1 C:/Data/ise/drp_des/drp_demo.ncd
```

**Note** The implementation run is created along with a synthesis run when the project is first created. The **import\_as\_run** command cannot be used on that implementation run because the synthesis run has not yet been completed. However, when the `design_mode` property is next set to a `GateLvl` design, the synthesis run is removed as unnecessary, and only the implementation run is left.

The following example resets an implemented run to prepare it for use by the **import\_as\_run** command, and then imports an NCD file, a TWX file, and a PCF file from an ISE placed and routed design:

```
reset_run impl_1
import_as_run -run impl_1 -twx C:/Data/ise/drp_des/drp_demo.twx-pcf \
C:/Data/ise/drp_des/drp_demo.pcf C:/Data/ise/drp_des/drp_demo.ncd
```

**Note** If the **reset\_run** command had not been used on the **impl\_1** run, the following error would have been returned: ERROR: Run needs to be reset before importing into it.

## See Also

- [create\\_project](#)
- [reset\\_run](#)
- [set\\_property](#)

## import\_files

Import files and/or directories into the active fileset.

### Syntax

```
import_files [-fileset arg] [-force] [-norecurse] [-flat]
[-relative_to arg] [-quiet] [-verbose] [files ...]
```

### Returns

A list of file objects that were imported

### Usage

Name	Description
<code>[-fileset]</code>	Fileset name
<code>[-force]</code>	Overwrite files of the same name in project directory
<code>[-norecurse]</code>	Disables the default behavior of recursive directory searches
<code>[-flat]</code>	Import the files into a flat directory structure
<code>[-relative_to]</code>	Import the files with respect to the given relative directory
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[ files ]</code>	Name of the files to import into fileset

### Categories

[Project](#), [Simulation](#)

### Description

Imports one or more files or the source file contents of one or more directories to the specified fileset.

This command is different from the **add\_files** command, which adds files by reference into the specified fileset. This command imports the files into the local project folders under `project.srsrcs\<fileset>\imports` and then adds the file to the specified fileset.

### Arguments

**-fileset** *name* - The fileset to which the specified source files should be added. If the specified fileset does not exist, the PlanAhead tool will return an error. If no fileset is specified the files will be added to the source fileset by default.

**-force** - Overwrite files of the same name in the local project directory and in the fileset.

**-norecurse** - Do not recurse through subdirectories of any specified directories. Without this argument the tool will also search through any subdirectories for additional source files that can be added to a project.

**-flat** - Import all files into the imports folder without preserving their relative paths. By default the directory structure of files is preserved as they are imported into the design.

**-relative\_to** *arg* - Import the files relative to the specified directory. This allows you to preserve the path to the imported files in the directory structure of the local project. The files will be imported to the imports folder with the path relative to the specified directory.

**Note** The **relative\_to** argument is ignored if the **-flat** argument is also specified. The **-flat** command eliminates the directory structure of the imported files.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*files* - One or more file names or directory names to be added to the specified fileset. If a directory name is specified, all valid source files found in the directory, and in subdirectories of the directory, will be added.

**Note** If the path is not specified as part of the file name, the current working directory is used, or the directory from which the tool was launched.

## Examples

The following example imports the top.ucf file into the constrs\_1 constraint fileset.

```
import_files -fileset constrs_1 top.ucf
```

The following example imports the valid source files into the source fileset (sources\_1) as a default since the **-fileset** argument is not specified. In addition, the **-norecurse** argument restricts the PlanAhead tool to looking only in the specified \level1 directory and not searching any subdirectories. All valid source files will be imported into the \imports folder of the project because the **-flat** argument has been specified.

```
import_files C:/Data/FPGA_Design/level1 -norecurse -flat
```

**Note** Without the **-flat** option a \level1 directory would be created inside of the \imports folder of the project.

The following example imports files into the source fileset (sources\_1) because the **-fileset** argument is not specified. Valid source files are imported from the \level1 directory, and all subdirectories, and the files will be written into the \imports folder of the project starting at the \Data directory due to the use of the **-relative\_to** argument.

```
import_files C:/Data/FPGA_Design/level1 -relative_to C:/Data
```

## See Also

[add\\_files](#)

## import\_ip

Import an IP file and add it to the fileset.

### Syntax

```
import_ip [-srcset arg] -file arg -name arg [-quiet] [-verbose]
```

### Returns

List of file objects that were added

### Usage

Name	Description
<i>[-srcset]</i>	Source set name
<b>-file</b>	Name of the IP file to be imported
<b>-name</b>	Name of new IP to be created
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution

### Categories

[Project](#), [IPFlow](#)

### Description

Imports an existing XCI or XCO file as an IP source into the current project.

The **import\_ip** command allows you to read existing IP files directly, and import them into the local project folders. Use the **add\_files** command to add IP files by reference into the current project.

Use the **create\_ip** command to create new IP files from the current IP catalog.

### Arguments

**-file** *arg* - The IP file to be imported into the current project. The IP must be in the form of an existing XCI file or XCO file. An XCI file is an IP-XACT format file that contains information about the IP parameterization. An XCO file is a CORE Generator log file that records all the customization parameters used to create the IP core and the project options in effect when the core was generated. The XCI or XCO files are used to recreate the core in the current project.

**Note** If the path is not specified as part of the file name, the tool will search for the specified file in the current working directory and then in the directory from which the tool was launched.

**-name** *arg* - The name to assign to the IP object as it is added to the current source fileset.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

## Examples

The following example imports the 10gig ethernet core into the current project, and assigns it a name of IP\_block1:

```
import_ip C:/Data/FPGA_Design/10gig_eth.xci -name IP_block1
```

## See Also

- [add\\_files](#)
- [create\\_ip](#)
- [generate\\_target](#)

## import\_synplify

Imports the given Synplify project file.

### Syntax

```
import_synplify [-copy_sources] [-quiet] [-verbose] file
```

### Returns

List of files object that were imported from the Synplify file

### Usage

Name	Description
<i>[-copy_sources]</i>	Copy all the sources from synplify project file into the created project
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>file</i>	Name of the Synplify project file to be imported

### Categories

[Project](#)

## import\_xise

Import XISE project file settings into the created project.

### Syntax

```
import_xise [-copy_sources] [-quiet] [-verbose] file
```

### Returns

True

### Usage

Name	Description
<i>[-copy_sources]</i>	Copy all ISE sources into the created project
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>file</i>	Name of the XISE project file to be imported

### Categories

[Project](#)

### Description

Imports an ISE project file (XISE) into the current project. This allows ISE projects to be quickly migrated for synthesis, simulation, and implementation. All project source files, constraint files, simulation files, and run settings are imported from the ISE project and recreated in the current project.

This command should be run on a new empty project. Since source files, constraints, and run settings are imported from the ISE project, any existing source files or constraints may be overwritten.

### Arguments

**-copy\_sources** - Copy source files in the ISE project to the local project directory structure rather than referenced from their current location. The default is to reference source files from their current location.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*file* - The name of the ISE project file (.XISE) to be imported into the current project.

## Examples

The following example creates a new project called importISE, and then imports the ISE project file (first\_use.xise) into the new project.

```
create_project importISE C:/Data/importISE import_xise \  
C:/Data/FPGA_design/ise_designs/drp_des/first_use.xise
```

**Note** This example does not specify the **-copy\_sources** argument, so all source files in the ISE project will be added to the current project by reference.

## See Also

[create\\_project](#)

## import\_xst

Imports the given XST project file.

### Syntax

```
import_xst [-copy_sources] [-quiet] [-verbose] file
```

### Returns

List of files object that were imported from the XST file

### Usage

Name	Description
<i>[-copy_sources]</i>	Copy all the sources from xst project file into the created project
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>file</i>	Name of the XST project file to be imported

### Categories

[Project](#)

### Description

Imports XST synthesis project files into the current project, including the various source files used in the XST run.

### Arguments

**-copy\_sources** - (Optional) Copy XST project source files to the local project directory structure rather than referenced from their current location. The default is to reference source files from their current location.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*file* - The name of the XST project file from which to import the source files.

### Examples

The following example creates a new project called `xst_test`, and imports the `drp_des.xst` file:

```
create_project xst_test C:/Data/FPGA_Design/xst_test
import_xst C:/Data/ise_designs/drp_des.xst
```

The source files specified in the specified XST project file are imported into the `xst_test` project.

## See Also

[create\\_project](#)

## infer\_diff\_pairs

Infer differential pairs, for ports just imported from a CSV, UCF, or XDC file.

### Syntax

```
infer_diff_pairs -file_type arg [-quiet] [-verbose] file
```

### Returns

Nothing

### Usage

Name	Description
<b>-file_type</b>	Input file type: 'csv', 'ucf', or 'xdc'
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>file</i>	Pin Planning CSV file, UCF file, or XDC file

### Categories

[FileIO](#)

## launch\_chipscope\_analyzer

Launch ChipScope Analyzer tool for a run.

### Syntax

```
launch_chipscope_analyzer [-run arg] [-csproject arg] [-quiet]
[-verbose]
```

### Returns

Nothing

### Usage

Name	Description
<code>[-run]</code>	Implemented run to launch ChipScope Analyzer with
<code>[-csproject]</code>	ChipScope project
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

### Categories

[ToolLaunch](#), [ChipScope](#)

### Description

Launches the ChipScope Pro Analyzer tool for the active run, or a specified Implemented Design run. You can setup a Netlist Design for use with ChipScope prior to implementation, using the `create_debug_core`, `create_debug_port`, and `connect_debug_port` commands.

The Implemented Design must also have a bitstream file generated by BitGen for `launch_chipscope_analyzer` to run. If BitGen has not been run, an error will be returned.

**Note** It is not enough to use the `write_bitstream` command to create a bitstream file. You must follow the steps outlined below in the second example

### Arguments

**-run** *arg* - The run name to use when launching the ChipScope Pro Analyzer. The specified run must be implemented and have a bitstream (.bit) file generated. ChipScope will use the bitstream file and the `debug_nets.cdc` file from the specified run.

**-csproject** *arg* - The name of the project to open in ChipScope Pro Analyzer. If you do not specify the project name, the default project name of `csdefaultproj.cpj` will be used. When you specify the project name, you should also specify the `.cpj` extension.

**Note** The project is created in the `project/project.data/sources_1/cs` folder.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns `TCL_OK` regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

## Examples

The following example launches ChipScope Pro Analyzer, specifying the implementation run to use and the name of the ChipScope project to create:

```
launch_chipscope_analyzer -run impl_3 -csproject impl_3_cs_project
```

The following example sets the **add\_step Bitgen** property for the **impl\_4** run, launches the **impl\_4** run, and then launches the ChipScope Pro Analyzer on the specified run:

```
set_property add_step Bitgen [get_runs impl_4]  
launch_runs impl_4 -jobs 2  
launch_chipscope_analyzer -run impl_4
```

**Note** In this example the ChipScope project will be called `csdefaultproj.cpj`.

## See Also

- [connect\\_debug\\_port](#)
- [create\\_debug\\_core](#)
- [create\\_debug\\_port](#)
- [launch\\_runs](#)
- [set\\_property](#)
- [write\\_bitstream](#)

## launch\_fpga\_editor

Launch FPGAEEditor tool for a Run.

### Syntax

```
launch_fpga_editor [-run arg] [-more_options arg] [-mapped_ncd]
[-quiet] [-verbose]
```

### Returns

Nothing

### Usage

Name	Description
<code>[-run]</code>	Implemented run to launch FED with
<code>[-more_options]</code>	More command line options
<code>[-mapped_ncd]</code>	Use Mapped NCD
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

### Categories

[ToolLaunch](#)

### Description

Launches the Xilinx FPGA Editor tool for the active run, or a specified run. When the design is open in the FPGA Editor, you can place critical elements of the design and examine critical timing pathways. You can also cross-probe between the PlanAhead tool and the FPGA Editor using the **crossprobe\_fed** command to quickly locate elements of your design in the two editors.

### Arguments

**-run** *arg* - The run name to use when launching the FPGA Editor. The specified run must have a mapped NCD or routed NCD file to launch the FPGA Editor.

**-more\_options** *arg* - Additional options to use when invoking the FPGA Editor software. For more information, see the online Help provided with FPGA Editor.

**-mapped\_ncd** - Use the mapped NCD file as input for FPGA Editor. This allows you to view the NCD file output from MAP, without the placement and routing data from PAR. You can use this to perform some preplacement and routing of critical components if necessary.

**Note** By default the **launch\_fpga\_editor** command loads the *design\_routed.ncd* file, unless this option is specified.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

## Examples

The following example launches FPGA Editor, specifying the implementation run to use and to open the mapped NCD instead of the placed and routed NCD:

```
launch_fpga_editor -run impl_4 -mapped_ncd
```

## See Also

[crossprobe\\_fed](#)

## launch\_impact

Launch iMPACT configuration tool for a run.

### Syntax

```
launch_impact [-run arg] [-ipf arg] [-quiet] [-verbose]
```

### Returns

Nothing

### Usage

Name	Description
<i>[-run]</i>	Implemented run to launch iMPACT with
<i>[-ipf]</i>	Project for iMPACT
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution

### Categories

[ToolLaunch](#)

## launch\_isim

Launch simulation using ISim simulator.

### Syntax

```
launch_isim [-simset arg] [-mode arg] [-quiet] [-verbose]
```

### Returns

Nothing

### Usage

Name	Description
<code>[-simset]</code>	Name of the simulation fileset
<code>[-mode]</code>	Simulation mode. Values: behavioral, timing Default: behavioral
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

### Categories

[ToolLaunch](#), [Simulation](#)

## launch\_modelsim

Launch simulation using ModelSim simulator.

### Syntax

```
launch_modelsim [-simset arg] [-mode arg] [-quiet] [-verbose]
```

### Returns

Nothing

### Usage

Name	Description
<code>[-simset]</code>	Name of the simulation fileset
<code>[-mode]</code>	Simulation mode. Values: behavioral, timing Default: behavioral
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

### Categories

[ToolLaunch](#), [Simulation](#)

## launch\_runs

Launch a set of runs.

### Syntax

```
launch_runs [-jobs arg] [-scripts_only] [-all_placement]
[-dir arg] [-to_step arg] [-next_step] [-host args]
[-remote_cmd arg] [-email_to args] [-email_all]
[-pre_launch_script arg] [-post_launch_script arg]
[-force] [-quiet] [-verbose] runs ...
```

### Returns

Nothing

### Usage

Name	Description
<code>[-jobs]</code>	Number of jobs Default: 1
<code>[-scripts_only]</code>	Only generate scripts
<code>[-all_placement]</code>	Export all fixed and non-fixed placement to ISE (by default only fixed placement will be exported)
<code>[-dir]</code>	Launch directory
<code>[-to_step]</code>	Last Step to run. Ignored when launching multiple runs. Not valid with <code>-next_step</code>
<code>[-next_step]</code>	Run next step. Ignored when launching multiple runs. Not valid with <code>-to_step</code> .
<code>[-host]</code>	Launch on specified remote host with a specified number of jobs. Example: <code>-host {machine1 2} -host {machine2 4}</code>
<code>[-remote_cmd]</code>	Command to log in to remote hosts Default: <code>ssh -q -o BatchMode=yes</code>
<code>[-email_to]</code>	List of email addresses to notify when jobs complete
<code>[-email_all]</code>	Send email after each job completes
<code>[-pre_launch_script]</code>	Script to run before launching each job
<code>[-post_launch_script]</code>	Script to run after each job completes
<code>[-force]</code>	Run the command, even if there are pending constraint changes, which will be lost (in a Partial Reconfig design)
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>runs</code>	Runs to launch

### Categories

Project



**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*runs* - The names of synthesis and implementation runs to launch.

## Examples

The following command launches three different synthesis runs with two parallel jobs:

```
launch_runs synth_1 synth_2 synth_4 -jobs 2
```

**Note** The results for each run will be written to a separate folder *synth\_1*, *synth\_2*, and *synth\_4* inside of the *project.runs* directory.

The following example creates a results directory to write run results. In this case a separate folder named *impl\_3*, *impl\_4*, and *synth\_3* will be written to the specified directory. In addition, the **-scripts\_only** argument tells PlanAhead to write *runme.bat* scripts to each of these folders but not to launch the runs at this time.

```
launch_runs impl_3 impl_4 synth_3 -dir C:/Data/FPGA_Design/results -scripts_only
```

## See Also

- [create\\_run](#)
- [get\\_runs](#)
- [opt\\_design](#)
- [place\\_design](#)
- [route\\_design](#)
- [set\\_property](#)
- [synth\\_design](#)

## launch\_sdk

Launch Xilinx Software Development Kit (SDK).

### Syntax

```
launch_sdk [-bit arg] [-bmm arg] [-workspace arg] [-hwspec arg]  
[-quiet] [-verbose]
```

### Returns

Nothing

### Usage

Name	Description
<i>[-bit]</i>	Specify the bitstream file for FPGA programming
<i>[-bmm]</i>	Specify the BMM file for BRAM initialization
<i>[-workspace]</i>	Specify the workspace directory for SDK projects
<i>[-hwspec]</i>	Specify the hardware platform specification file (system.xml)
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution

### Categories

[ToolLaunch](#), [XPS](#)

## launch\_xpa

Launch XPower Analyzer tool.

### Syntax

```
launch_xpa [-run arg] [-more_options arg] [-mapped_ncd] [-quiet]  
[-verbose]
```

### Returns

Nothing

### Usage

Name	Description
<i>[-run]</i>	Implemented run to launch XPA with
<i>[-more_options]</i>	More command line options
<i>[-mapped_ncd]</i>	Use Mapped NCD
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution

### Categories

[ToolLaunch](#)

## link\_design

Open a netlist design.

### Syntax

```
link_design [-name arg] [-part arg] [-constrset arg] [-top arg]
[-quiet] [-verbose]
```

### Returns

Design object

### Usage

Name	Description
<code>[-name]</code>	Design name
<code>[-part]</code>	Target part
<code>[-constrset]</code>	Constraint fileset to use
<code>[-top]</code>	Specify the top module name when the structural netlist is Verilog
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

### Categories

Tools

### Description

Opens a new or existing Netlist design, linking the netlists and constraints with the target part to create the design. This can also be accomplished with the **open\_run** command.

The design\_mode property for the current source fileset must be defined as GateLvl in order to open a Netlist design. If not, you will get the following error:

```
ERROR: The design mode of 'sources_1' must be GateLvl.
```

### Arguments

**-name arg** - The name of a new or existing Netlist design.

**-part arg** - The Xilinx device to use when creating a new design. If the part is not specified the default part will be used.

**-constrset arg** - The name of the constraint fileset to use when opening the design.

**Note** The **-constrset** argument must refer to a constraint fileset that exists. It cannot be used to create a new fileset. Use create\_filesset for that purpose.

**-top arg** - The top module of the design hierarchy of the netlist.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

## Examples

The following creates a new Netlist design called Net1:

```
link_design -name Net1
```

**Note** The default source set, constraint set, and part will be used in this example.

The following example opens a Netlist design called Net1, and specifies the constraint set to be used:

```
link_design -name Net1 -constrset con1
```

## See Also

[open\\_run](#)

## list\_param

Get all parameter names.

### Syntax

```
list_param [-quiet] [-verbose]
```

### Returns

List

### Usage

Name	Description
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	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, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

### Examples

The following example returns a list of all user-definable parameters:

```
list_param
```

### See Also

- [get\\_param](#)
- [report\\_param](#)
- [reset\\_param](#)
- [set\\_param](#)

## list\_property

List properties of object.

### Syntax

```
list_property [-class arg] [-quiet] [-verbose] [object]
```

### Returns

List of property names

### Usage

Name	Description
<i>[-class]</i>	Object type to query for properties. Ignored if object is specified.
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ object ]</i>	Object to query for properties

### Categories

[Object](#), [PropertyAndParameter](#)

### Description

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

**Note** `report_property` also returns a list of properties on an object, but includes the property type and property value.

### Arguments

**-class** *arg* - The class of object for which to list the properties.

**Note** When both **-class** and *object* are specified, the properties of the specific object are returned.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the `set_msg_limit` command.

*object* - The single object on which to report properties.

**Note** If you specify multiple objects you will get an error.

## Examples

The following example returns all properties of the specified object:

```
list_property [get_cells cpuEngine]
```

## See Also

- [create\\_property](#)
- [get\\_cells](#)
- [get\\_property](#)
- [list\\_property\\_value](#)
- [report\\_property](#)
- [reset\\_property](#)
- [set\\_property](#)

## list\_property\_value

List legal property values of object.

### Syntax

```
list_property_value [-class arg] [-quiet] [-verbose] name
[object]
```

### Returns

List of property values

### Usage

Name	Description
<i>[-class]</i>	Object type to query for legal property values. Ignored if object is specified.
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>name</i>	Name of property whose legal values is to be retrieved
<i>[ object ]</i>	Object to query for legal properties values

### Categories

[Object](#), [PropertyAndParameter](#)

### Description

Gets a list of valid values for an enumerated type property of either a class of objects or a specific object.

**Note** The command cannot be used to return valid values for properties other than enum properties. The **report\_property** command will return the type of property to help you identify enum properties.

### Arguments

**-class** *arg* - The class of object to query. The class of object can be used in place of an actual object.

*name* - The name of the property to be queried. Only properties with an enumerated value, or a predefined value set, can be queried with this command. All valid values of the specified property will be returned.

*object* - An object to query. An actual object can be used in place of the -class argument to specify the type of object to query.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

## Examples

The following example returns the list of valid values for the KEEP\_HIERARCHY property from cell objects:

```
list_property_value KEEP_HIERARCHY -class cell
```

The following example returns the same result, but uses an actual cell object in place of the general cell class:

```
list_property_value KEEP_HIERARCHY [get_cells cpuEngine]
```

## See Also

- [create\\_property](#)
- [get\\_cells](#)
- [get\\_property](#)
- [list\\_property](#)
- [report\\_property](#)
- [reset\\_property](#)
- [set\\_property](#)

## list\_targets

List applicable targets for the specified source.

### Syntax

```
list_targets [-quiet] [-verbose] files
```

### Returns

List of targets

### Usage

Name	Description
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>files</i>	Source file for which the targets needs to be listed

### Categories

[Project](#), [XPS](#)

## load\_reconfig\_modules

Load specific Reconfigurable Modules or all modules from a given run.

### Syntax

```
load_reconfig_modules [-force] [-run arg] [-quiet] [-verbose]  
[reconfig_modules]
```

### Returns

Nothing

### Usage

Name	Description
<i>[-force]</i>	Run the command, even if there are pending constraint changes, which will be lost
<i>[-run]</i>	Run to load Reconfigurable Modules from
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ reconfig_modules ]</i>	Reconfigurable Module(s) to load

### Categories

[PartialReconfiguration](#)

## make\_diff\_pair\_ports

Make differential pair for 2 ports.

### Syntax

```
make_diff_pair_ports [-quiet] [-verbose] ports ...
```

### Returns

Nothing

### Usage

Name	Description
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>ports</i>	Ports to join

### Categories

[PinPlanning](#)

### Description

Joins two existing ports to create a differential pair.

**Note** This command operates silently and does not return direct feedback of its operation.

### Arguments

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*ports* - Two port objects to join as a differential pair.

### Examples

The following example joins the two specified ports to create a differential pair:

```
make_diff_pair_ports PORT_N PORT_P
```

### See Also

- [create\\_interface](#)
- [create\\_port](#)

## make\_wrapper

Generate HDL wrapper for the specified source.

### Syntax

```
make_wrapper [-top] [-testbench] [-inst_template] [-fileset arg]
[-import] [-quiet] [-verbose] files
```

### Returns

Nothing

### Usage

Name	Description
<i>[-top]</i>	Create a top-level wrapper for the specified source
<i>[-testbench]</i>	Create a testbench for the specified source
<i>[-inst_template]</i>	Create an instantiation template for the specified source. The template will not be added to the project and will be generated for reference purposes only.
<i>[-fileset]</i>	Fileset name
<i>[-import]</i>	Import generated wrapper to the project
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>files</i>	Source file for which the wrapper needs to be generated

### Categories

[Project](#), [XPS](#), [SysGen](#)

## mark\_objects

Mark objects in GUI.

### Syntax

```
mark_objects [-rgb args] [-color arg] [-quiet]
             [-verbose] objects
```

### Returns

Nothing

### Usage

Name	Description
<i>[-rgb]</i>	RGB color index list
<i>[-color]</i>	Valid values are red green blue magenta yellow cyan and orange
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>objects</i>	Objects to mark

### Categories

[GUIControl](#)

### Description

Marks specified objects in GUI mode. This command places an iconic mark to aid in the location of the specified object or objects. The mark is displayed in a color as determined by one of the color options.

Objects can be unmarked with the **unmark\_objects** command.

**Note** Use only one color option. If both color options are specified, **-rgb** takes precedence over **-color**

### Arguments

**-rgb** *args* - The color to use in the form of an RGB code specified as {R G B}. For instance, {255 255 0} specifies the color yellow, while {0 255 0} specifies green.

**-color** *arg* - The color to use for marking the specified object or objects. Supported colors are: red, green, blue, magenta, yellow, cyan, and orange.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*objects* - One or more objects to be marked.

## Examples

The following example adds a red icon to mark the currently selected objects:

```
mark_objects -color red [get_selected_objects]
```

## See Also

- [get\\_selected\\_objects](#)
- [unmark\\_objects](#)

## open\_example\_project

Open the example project for the indicated IP.

### Syntax

```
open_example_project [-dir arg] [-force] [-in_process] [-quiet]
[-verbose] objects ...
```

### Returns

The Project that was opened

### Usage

Name	Description
<i>[-dir]</i>	Path to directory where example project will be created
<i>[-force]</i>	Overwrite an example project if it exists
<i>[-in_process]</i>	Open the example project in the same process
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>objects</i>	The objects whose example projects will be opened

### Categories

[Project](#), [IPFlow](#)

### Description

Open an example project for the specified IP cores. The example project can be used to explore the features of the IP core in a stand-alone project, instead of integrated into the current project.

### Arguments

**-dir** *arg* - (Optional) Specifies the path to the directory where the example project will be written.

**-force** - (Optional) Force the opening of a new example project, overwriting an existing example project at the specified path.

**-in\_process** - (Optional) Open the example project in the same tool process as the current project. As a default, without this argument, a new process instance of the tool will be launched for the example project.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*objects* - The IP cores to open example projects for.

## Examples

The following generates the target data for the example project, then opens the example project for the specified IP core:

```
generate_target {example} [get_ips blk_mem*]  
open_example_project -force [get_ips blk_mem*]
```

**Note** The Example target data must be generated prior to using the **open\_example\_project** command. This will create a Tcl script to open and configure the specified IP core

## See Also

- [create\\_ip](#)
- [generate\\_target](#)
- [get\\_ips](#)
- [import\\_ip](#)

## open\_io\_design

Open an IO design.

### Syntax

```
open_io_design [-name arg] [-part arg] [-constrset arg] [-quiet]  
[-verbose]
```

### Returns

Design object

### Usage

Name	Description
<code>[-name]</code>	Design name
<code>[-part]</code>	Target part
<code>[-constrset]</code>	Constraint fileset to use
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

### Categories

[Project](#)

### Description

Opens a new or existing I/O Pin Planning design.

**Note** The `design_mode` property for the current source fileset must be defined as `PinPlanning` in order to open an I/O design. If not, you will get the following error:

```
ERROR: The design mode of 'sources_1' must be PinPlanning
```

### Arguments

**-name** *arg* - The name of a new or existing I/O Pin Planning design.

**-part** *arg* - The Xilinx device to use when creating a new design. If the part is not specified the default part will be used.

**-constrset** *arg* - The name of the constraint fileset to use when opening an I/O design.

**Note** The `-constrset` argument must refer to a constraint fileset that exists. It cannot be used to create a new fileset. Use `create_fileset` for that purpose.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns `TCL_OK` regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the `set_msg_limit` command.

## Examples

The following creates a new I/O design called myIO:

```
open_io_design -name myIO
```

**Note** The default source set, constraint set, and part will be used in this case.

The following example opens an existing I/O design called myIO, and specifies the constraint set to be used:

```
open_io_design -name myIO -constrset topCon
```

## See Also

[create\\_project](#)

## open\_project

Open a PlanAhead project file (.ppr).

### Syntax

```
open_project [-read_only] [-quiet] [-verbose] file
```

### Returns

Opened project object

### Usage

Name	Description
<i>[-read_only]</i>	Open the project in read-only mode
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>file</i>	Project file to be read

### Categories

[Project](#)

### Description

Opens a project file (.ppr) for editing the design source files and hierarchy, for performing I/O pin planning and floorplanning, and to synthesize and implement the device.

### Arguments

**-read\_only** - Open the project in read only mode. You will not be able to save any modifications to the project unless you use the **save\_project\_as** command to save the project to a new editable project.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*file* - The project file to open. You must include both the path to the file and the .ppr file extension.

### Examples

The following example opens the project named `my_project1` located in the Designs directory.

```
open_project C:/Designs/my_project1.ppr
```

**Note** The project must be specified with the .ppr extension for the PlanAhead tool to recognize it as a project file. The path to the file must be specified along with the project file name or the PlanAhead tool will return an error that it cannot find the specified file.

## See Also

- [create\\_project](#)
- [current\\_project](#)

## open\_rtl\_design

Open an rtl design.

### Syntax

```
open_rtl_design [-name arg] [-part arg] [-constrset arg]  
[-top arg] [-quiet] [-verbose]
```

### Returns

Design object

### Usage

Name	Description
<code>[-name]</code>	Design name
<code>[-part]</code>	Target part
<code>[-constrset]</code>	Constraint fileset to use
<code>[-top]</code>	Specify the top module name
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

### Categories

[Project](#)

### Description

Opens a new or existing RTL source design.

**Note** Verilog or VHDL source files must be added to the source fileset, and a top module identified, before opening an RTL design.

### Arguments

**-name** *arg* - The name of a new or existing RTL design.

**-part** *arg* - The Xilinx device to use when creating a new design. If the part is not specified the default part will be used.

**-constrset** *arg* - The name of the constraint fileset to use when opening the design.

**Note** The **-constrset** argument must refer to a constraint fileset that exists. It cannot be used to create a new fileset. Use `create_fileset` for that purpose.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the `set_msg_limit` command.

## Examples

The following creates a new RTL design called RTL1:

```
open_rtl_design -name RTL1
```

**Note** The default source set, constraint set, and part will be used in this case.

The following example opens an existing RTL design called RTL1, and specifies the constraint set to be used:

```
open_rtl_design -name RTL1 -constrset top
```

**Note** The default source set and part will be used in this case.

## See Also

- [config\\_run](#)
- [launch\\_runs](#)
- [open\\_impl\\_design](#)
- [open\\_netlist\\_design](#)

## open\_run

Open a run into a netlist or implementation design.

### Syntax

```
open_run [-name arg] [-quiet] [-verbose] run
```

### Returns

Design object

### Usage

Name	Description
<i>[-name]</i>	Design name
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>run</i>	Run to open into the design

### Categories

[Project](#)

### Description

Opens the specified synthesis run into a Netlist Design or implementation run into an Implemented Design. The run properties defining the target part and constraint set are combined with the synthesis or implementation results to create the design view in the tool.

### Arguments

**-name** - (Optional) The name of the design to open.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*run* - Specifies the run name of the synthesis or implementation run to open. The run must have completed synthesis or implementation before it can be opened as a design.

**Note** If you attempt to open a run that has not been launched the tool will return an error.

## Examples

The following command opens the specified synthesis run into a Netlist Design named synthPass1 :

```
open_run -name synthPass1 synth_1
```

The following opens an Implemented Design for impl\_1:

```
open_run impl_1
```

## See Also

[launch\\_runs](#)

## place\_cell

Move or place one or more instances to new locations. Sites and cells are required to be listed in the right order and there should be same number of sites as number of cells.

### Syntax

```
place_cell [-quiet] [-verbose] cell_site_list ...
```

### Returns

Nothing

### Usage

Name	Description
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>cell_site_list</i>	a list of cells and sites in the interleaved order

### Categories

[Floorplan](#)

### Description

Places cells onto device resources of the target part. Cells can be placed onto specific BEL sites (i.e. **SLICE\_X49Y60/A6LUT**), or into available SLICE resources (i.e. **SLICE\_X49Y60**). If you specify the SLICE but not the BEL the tool will determine an appropriate BEL within the specified SLICE if one is available.

When placing a cell onto a specified site, the site must not be currently occupied, or an error will be returned: "Cannot set site and bel property of instances. Site SLICE\_X49Y61 is already occupied."

You can test if a site is occupied by querying the IS\_OCCUPIED property of a BEL site:

```
get_property IS_OCCUPIED [get_bels SLICE_X48Y60/D6LUT]
```

**Note** The IS\_OCCUPIED property of a SLICE only tells you if some of the BELs within the SLICE are occupied; not whether or not the SLICE is fully occupied.

This command can be used to place cells, or to move placed cells from one site on the device to another site. The command syntax is the same for placing an unplaced cell, or moving a placed cell.

When moving a placed cell, if you specify only the SLICE for the site, the tool will attempt to place the cell onto the same BEL site in the new SLICE as it currently is placed. For instance moving a cell from the B6LUT, by specifying a new SLICE, will cause the tool to attempt to place the cell onto the B6LUT in the new SLICE. If this BEL site is currently occupied, an error is returned.

**Note** This command operates silently and does not return direct feedback of its operation.

## Arguments

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*cell\_site\_list* - Specifies a list of cells and sites as {*cell\_name site*}. The cell name is listed first, followed the BEL site or SLICE to place the cell onto. If the site is specified as a SLICE, the tool will select an available BEL within the SLICE. Multiple cells can be placed onto multiple sites by repeating the cell/site pair multiple times as needed:

{*cell\_name1 site1 cell\_name2 site2 cell\_name3 site3 ... cell\_nameN siteN*}.

## Examples

The following example places the specified cell onto the specified BEL site:

```
place_cell div_cntr_reg_inferredi_4810_15889 SLICE_X49Y60/D6LUT
```

The following example places the specified cell into the specified SLICE:

```
place_cell div_cntr_reg_inferredi_4810_15889 SLICE_X49Y61
```

**Note** The tool will select an appropriate BEL site if one is available. If no BEL is available, an error will be returned

The following example places multiple cells onto multiple sites:

```
place_cell { \
cpuEngine/cpu_iwb_adr_o/buffer_fifo/i_4810_17734 SLICE_X49Y60/A6LUT \
cpuEngine/or1200_cpu/or1200_mult_mac/i_4775_15857 SLICE_X49Y60/B6LUT \
cpuEngine/cpu_iwb_adr_o/buffer_fifo/xlnx_opt_LUT_i_4810_18807_2 SLICE_X49Y60/C6LUT }
```

## See Also

- [create\\_cell](#)
- [remove\\_cell](#)
- [unplace\\_cell](#)

## place\_pblocks

Run the pblocks Placer.

### Syntax

```
place_pblocks [-effort arg] [-utilization arg] [-quiet]
[-verbose] pblocks ...
```

### Returns

Nothing

### Usage

Name	Description
<i>[-effort]</i>	Placer effort level (per pblock) Values: LOW, MEDIUM, HIGH Default: HIGH
<i>[-utilization]</i>	Placer utilization (per pblock)
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>pblocks</i>	List of pblocks to place

### Categories

[Floorplan](#)

### Description

Places Pblocks onto the fabric of the FPGA. Pblocks must be created using the `create_pblock` command, and should be populated with assigned logic using the `add_cells_to_pblock` command.

**Note** An empty Pblock will be placed as directed, but results in a Pblock covering a single CLB tile (two slices).

### Arguments

**-effort** *arg* - Effort level that the Pblock placer should use in placing each Pblock onto the fabric. Valid values are LOW, MEDIUM, HIGH, with the default being HIGH.

**-utilization** *arg* - Percentage of device resources that should be consumed by the logic elements assigned to a Pblock when it is placed onto the FPGA. For instance, a utilization rate of 50% means that half of the resources should be allocated to the logic in the Pblock, and half should be left for other design elements to be intermingled. A high utilization rate makes the Pblock smaller but more difficult to place, while a smaller utilization makes the Pblock larger.

**Note** Pblock utilization is post-synthesis estimation. Actual results may be different, and may require you to resize the Pblock using the **resize\_pblock** command.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*pblocks* - One or more Pblocks to be placed onto the fabric of the FPGA.

## Examples

The following example places the specified Pblocks with a utilization of 75%:

```
place_pblocks -effort LOW -utilization 75 block1 block2 block3 block4 block5
```

## See Also

- [add\\_cells\\_to\\_pblock](#)
- [create\\_pblock](#)
- [resize\\_pblock](#)

## place\_ports

Automatically place a set of ports.

### Syntax

```
place_ports [-skip_unconnected_ports] [-check_only] [-quiet]  
[-verbose] [ports ...]
```

### Returns

Nothing

### Usage

Name	Description
<code>[-skip_unconnected_ports]</code>	Do not place unconnected ports
<code>[-check_only]</code>	Only check IO/Clock placement DRCs
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>[ ports ]</code>	Ports to place (if omitted, all ports will be placed)

### Categories

[PinPlanning](#)

### Description

Automatically places ports on an available I/O or clocking site.

**Note** This command operates silently and does not return direct feedback of its operation.

### Arguments

**-skip\_unconnected\_ports** - (Optional) Do not place unconnected ports

**-check\_only** - (Optional) Check only I/O or clocking design rule checks (DRCs) during placement.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

**ports** - (Optional) The names of the ports to be placed. If no *ports* are specified, all ports will be placed.

## Examples

The following example automatically places all ports in the design, and only checks I/O and clock DRCs:

```
place_ports -check_only
```

## See Also

- [create\\_port](#)
- [delete\\_port](#)
- [create\\_interface](#)
- [make\\_diff\\_pair\\_ports](#)

## promote\_run

Promote previously implemented Partitions to make them available to import and reuse in this or other runs.

### Syntax

```
promote_run [-partition_names args] [-promote_dir arg]
[-description arg] [-no_state_update] [-quiet] [-verbose] run
```

### Returns

Nothing

### Usage

Name	Description
<i>[-partition_names]</i>	List of Partitions to be promoted
<i>[-promote_dir]</i>	Promote path
<i>[-description]</i>	Promote description
<i>[-no_state_update]</i>	Do not automatically update the state of promoted Partitions to Import
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>run</i>	Implemented run to be promoted

### Categories

[PartialReconfiguration](#), [Partition](#)

### Description

Promotes previously implemented partitions to make them available for import and reuse in other designs.

### Arguments

**-partition\_names** *arg* - (Optional) The partition or partitions to be promoted.

**-promote\_dir** *arg* - (Optional) The path and directory into which to write the partition data.

**Note** If the specified directory already exists, the **promote\_run** command will overwrite it without warning

**Note** If the path is not specified as part of the directory name, the tool will create a directory in your home directory.

- For Windows: %APPDATA%/Xilinx/PlanAhead
- For Linux: \$HOME/.Xilinx/PlanAhead

**-description** *{text description}* - (Optional) A description of the partition being promoted. Enclose *text* in braces {}. This description can be displayed or queried when using the partition in another design.

**-no\_state\_update** - (Optional) Do not automatically update partition states. By default the tool will notify you of any updates to a promoted partition. After promotion, the partition state will change to import for promoted partitions.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*run* - The implemented run to be promoted.

## Examples

The following example promotes the synthesis run for the usbEngine0 and usbEngine1 partitions, and promotes the implementation run for usbEngine0:

```
promote_run -run synth_1 -partition_names {usbEngine0 usbEngine1} -promote_dir \  
C:/Data/partition/DP_RTL.promote/Xsynth_1  
promote_run -run impl_1 -partition_names usbEngine0 -promote_dir \  
C:/Data/partition/DP_RTL.promote/Ximpl_1 -description {Implementation of USB0}
```

## See Also

- [config\\_partition](#)
- [load\\_reconfig\\_modules](#)

## read\_chipscope\_cdc

Import ChipScope Core Inserter CDC file.

### Syntax

```
read_chipscope_cdc [-quiet] [-verbose] file
```

### Returns

Nothing

### Usage

Name	Description
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>file</i>	ChipScope CDC file name

### Categories

[FileIO](#), [ChipScope](#)

### Description

Reads a ChipScope Definition and Connection (CDC) file to associate with a Netlist Design in the current project. This file stores information about core parameters, and core settings for ChipScope ILA debug cores, and can also be used as input to the ChipScope Pro Analyzer to import signal names.

The ChipScope CDC file can come from an ISE project or from another PlanAhead project through the **write\_chipscope\_cdc** command.

If certain parameters of the CDC file are not acceptable to the current project then those parameters will not be imported. For instance, if signals specified for connection to ports do not exist in the current netlist, those signals will be ignored.

### Arguments

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*file* - The path and filename of the CDC file to read into. The CDC file defines the debug core to insert, the signals to probe, and the clock domains for those signals.

### Examples

The following example reads the specified CDC file:

```
read_chipscope_cdc C:/Data/FPGA_Design/bft.cdc
```

## See Also

- [create\\_debug\\_core](#)
- [get\\_debug\\_cores](#)
- [write\\_chipscope\\_cdc](#)

## read\_csv

Import package pin and port placement information.

### Syntax

```
read_csv [-verbose] file
```

### Returns

Nothing

### Usage

Name	Description
<i>[-verbose]</i>	Suspend message limits during command execution
<i>file</i>	Pin Planning CSV file

### Categories

[FileIO](#)

### Description

Imports package pin and port placement information from a comma separated value (CSV) file.

The CSV file can only be imported into an IO Pin Planning project. In all other projects the pin definitions are imported with the source design data.

The specific format and requirements of the CSV file are described in the *PlanAhead Users Guide* (ug632.pdf).

### Arguments

*file* - The file name of the CSV file to be imported.

**Note** If the path is not specified as part of the file name, the tool will search for the specified file in the current working directory and then in the directory from which the tool was launched.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

### Examples

The following example imports a CSV file into an open project:

```
read_csv C/Data/pinList.csv
```

The following example sets up a new IO Pin Planning project, and then imports the specified CSV file into it:

```
create_project myPinPlan C:/Data/myPinPlan -part xc7v285tffg1157-1
set_property design_mode PinPlanning [current_filesset]
open_io_design -name io_1
read_csv C:/Data/import.csv
```

**Note** The `design_mode` property on the source fileset is what determines the nature of the project.

## See Also

- [create\\_project](#)
- [open\\_io\\_design](#)
- [set\\_property](#)
- [write\\_csv](#)

## read\_edif

Read one or more EDIF files.

### Syntax

```
read_edif [-quiet] [-verbose] files
```

### Returns

List of file objects that were added

### Usage

Name	Description
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>files</i>	EDIF file name(s)

### Categories

[FileIO](#)

### Description

Imports an EDIF netlist file into the Design Source fileset of the current project.

### Arguments

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*files* - The name of the EDIF files to be imported.

**Note** If the path is not specified as part of the file name, the tool will search for the specified file in the current working directory and then in the directory from which the tool was launched.

### Examples

The following example imports an EDIF file into the open project:

```
read_edif C/Data/bft_top.edf
```

### See Also

[write\\_edif](#)

## read\_pxml

Import Partition definitions from a PXML file.

### Syntax

```
read_pxml [-quiet] [-verbose] file
```

### Returns

Nothing

### Usage

Name	Description
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>file</i>	PXML file name

### Categories

[FileIO](#)

### Description

Imports a partition XML file into the current design. The PXML file contains partition information related to hierarchical design. A PXML file is created by the PlanAhead tool during synthesis or implementation. You can also create one by hand or by using an XML template provided in the PlanAhead tool installation directory. Refer to the *Hierarchical Design Methodology Guide* (ug748) for more information on partitioning designs and creating a PXML file.

A partition must be defined to implement, or to import in the RTL design in order to be properly handled in synthesis and implementation. Therefore read\_pxml must be used on an open RTL design.

### Arguments

*file* - The name of the PXML file. The file must be named xpartition.pxml, or you will get an error when trying to read the file.

**Note** If the path is not specified as part of the file name, the tool will search for the specified file in the current working directory and then in the directory from which the tool was launched.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

## Examples

The following example reads the specified PXML file for partition information related to the hierarchy of the design:

```
read_pxml C:/Data/FPGA_Design/pxmlTest.pxml
```

## See Also

[config\\_partition](#)

## read\_twx

Read timing results from Trace STA tool.

### Syntax

```
read_twx [-cell arg] [-pblock arg] [-quiet] [-verbose] name file
```

### Returns

Nothing

### Usage

Name	Description
<i>[-cell]</i>	Interpret names in the report file as relative to the specified cell
<i>[-pblock]</i>	Interpret names in the report file as relative to the specified pblock
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>name</i>	Name for the set of results
<i>file</i>	Name of the Trace import file

### Categories

[FileIO](#)

### Description

Imports timing results in the TWX format timing report files generated by the Xilinx Timing Reporter And Circuit Evaluator (TRACE) tool. The TWX file can be imported at the top-level, which is the default, or at a specific cell-level or relative to a specific Pblock.

After the TWX files are imported, the timing results display in the Timing Results view in GUI mode.

### Arguments

**-cell** *arg* - (Optional) Specify The name of a hierarchical cell in the current design to import the TWX file into. The timing paths will be applied to the specified cell.

**-pblock** *arg* - (Optional) The name of a Pblock in the current design. The timing paths will be imported relative to the specified block.

*name* - The name of the Timing Results view to create when importing the timing paths in the TWX file.

**Note** Both *name* and *file* are required positional arguments. The *name* argument must be provided first.

*file* - The file name of the TWX file to be imported.

**Note** If the path is not specified as part of the file name, the tool will search for the specified file in the current working directory and then in the directory from which the tool was launched.

## Examples

The following example reads the specified TWX file into the top-level of the design:

```
read_twx C:/Data/timing_files/bft.twx
```

## See Also

[report\\_timing](#)

## read\_ucf

Import physical constraints from a file.

### Syntax

```
read_ucf [-cells args] [-ref arg] [-verbose] file
```

### Returns

List of added files

### Usage

Name	Description
<i>[-cells]</i>	Import constraints for these cells
<i>[-ref]</i>	Import constraints for this ref
<i>[-verbose]</i>	Suspend message limits during command execution
<i>file</i>	Input UCF file name

### Categories

[FileIO](#)

### Description

Imports physical constraints from a user constraint file (UCF). The UCF can be imported at the top-level, which is the default, or at a specific cell-level. When imported at the top-level, the specified UCF file is added to the active constraint fileset.

**Note** Constraints from the UCF file will overwrite any current constraints of the same name. Therefore, exercise some caution when reading a UCF file to be sure you will not overwrite important constraints.

This command is similar to the **add\_files** command in that the UCF file is added by reference rather than imported into the local project directory.

### Arguments

*file* - The file name of the UCF file to be imported.

**Note** If the path is not specified as part of the file name, the tool will search for the specified file in the current working directory and then in the directory from which the tool was launched.

**-cell** *arg* - The name of a hierarchical cell in the current design to import the UCF file into. The constraints will be applied to the specified block, and the imported UCF file will not be added to the active constraint fileset.

**Note** A design must be open when specifying the **-cell** option.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

## Examples

The following example reads the specified UCF file into the top-level of the design:

```
read_ucf C:/Data/FPGA_Design/top1.ucf
```

## See Also

- [add\\_files](#)
- [write\\_ucf](#)
- [save\\_design](#)

## read\_verilog

Read one or more Verilog files.

### Syntax

```
read_verilog [-library arg] [-sv] [-quiet] [-verbose] files ...
```

### Returns

List of file objects that were added

### Usage

Name	Description
<i>[-library]</i>	Library name Default: work
<i>[-sv]</i>	Enable system verilog compilation
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>files</i>	Verilog file name(s)

### Categories

[FileIO](#)

### Description

Reads Verilog or SystemVerilog source files. This command is similar to the **add\_files** command. The Verilog file is added to the source fileset as it is read. If the **-library** argument is specified, the file is added with the Library property defined appropriately.

Because SystemVerilog is a superset of the Verilog language, the **read\_verilog** command can read both file types. However, for SystemVerilog files, the **-sv** option needs to be specified for **read\_verilog** to enable compilation in the SystemVerilog mode. In this mode, the PlanAhead tool recognizes and honors the SystemVerilog keywords and constructs.

You can have a mixture of both Verilog files (.v files), and SystemVerilog files (.sv files), as well as VHDL (using **read\_vhdl**). When the PlanAhead tool compiles these files for synthesis, it creates separate "compilation units" for each file type. All files of the same type are compiled together.

### Arguments

**-library** *arg* - The library the Verilog file should reference. The default Verilog library is work.

**-sv** - Read the files as a SystemVerilog compilation group.

**Note** Since Verilog is a subset of SystemVerilog, unless a Verilog source has user-defined names that collide with reserved SystemVerilog keywords, reading Verilog files with the **-sv** switch enables SystemVerilog compilation mode for those files. However, adding a SystemVerilog file in a Verilog compilation unit (without **-sv**) will not work.

*files* - The name of one or more Verilog files to be read.

**Note** If the path is not specified as part of the file name, the tool will search for the specified file in the current working directory and then in the directory from which the tool was launched.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the `set_msg_limit` command.

## Examples

The following example reads the specified Verilog file and adds it to the source fileset:

```
read_verilog C:/Data/FPGA_Design/new_module.v
```

The following example creates two compilation units, one for SystemVerilog files and one for Verilog files:

```
read_verilog -sv { file1.sv file2.sv file3.sv }  
read_verilog { file1.v file2.v file3.v }
```

## See Also

- [add\\_files](#)
- [read\\_vhdl](#)

## read\_vhdl

Read one or more VHDL files.

### Syntax

```
read_vhdl [-library arg] [-quiet] [-verbose] files
```

### Returns

List of file objects that were added

### Usage

Name	Description
<i>[-library]</i>	VHDL library Default: work
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>files</i>	VHDL file name(s)

### Categories

[FileIO](#)

### Description

Reads a VHDL source file. This command is similar to the **add\_files** command. The VHDL file is added to the source fileset as it is read. If the **-library** argument is specified, the file is added with the Library property defined appropriately.

### Arguments

**-library** *arg* - The library the VHDL file should reference. The default VHDL library is work.

*file* - Filename of the VHDL file to be read.

**Note** If the path is not specified as part of the file name, the tool will search for the specified file in the current working directory and then in the directory from which the tool was launched.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

### Examples

The following example reads the specified VHDL file and adds it to the source fileset:

```
read_vhdl C:/Data/FPGA_Design/new_module.vhdl
```

## See Also

[add\\_files](#)

## read\_xdl

Import placement information from a file.

### Syntax

```
read_xdl [-pblock arg] [-cell arg] [-quiet] [-verbose] file
```

### Returns

Nothing

### Usage

Name	Description
<i>[-pblock]</i>	Import placement for this pblock
<i>[-cell]</i>	Import placement for this cell
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>file</i>	Placement file name

### Categories

[FileIO](#)

### Description

Import ISE placement results using XDL format data. XDL data is created automatically when implementation runs are launched. You can also create an XDL format file from an NCD file using the XDL command-line tool.

You can create XDL files and import placement for the entire design, for individual modules, or relative to specific Pblocks.

### Arguments

**-pblock *arg*** - (Optional) The name of a Pblock in the current design. The data in the XDL file will be imported relative to the specified block.

**-cell *arg*** - (Optional) The name of a hierarchical cell in the current design to import the XDL file into. The data in the XDL file will be imported relative to the specified cell.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*file* - The file name of the XDL file to be imported.

**Note** If the path is not specified as part of the file name, the tool will search for the specified file in the current working directory and then in the directory from which the tool was launched.

## Examples

The following example reads the specified XDL file into the top-level of the design:

```
read_xdl C:/Data/FPGA_Designs/bft.xdl
```

## redo

Re-do previous command.

### Syntax

```
redo [-list] [-quiet] [-verbose]
```

### Returns

With -list, the list of redoable tasks

### Usage

Name	Description
<code>[-list]</code>	Show a list of redoable tasks
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

### Categories

[GUIControl](#)

### Description

Redo a command that has been previously undone. This command can be used repeatedly to redo a series of commands.

If a command group has been created using the **startgroup** and **endgroup** commands, the redo command will redo the group of commands as a sequence.

### Arguments

**-list** - Get the list of commands that can be redone. When you use the **undo** command, the tool will step backward through a list of commands. The **redo** command can then be used to redo those commands.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

### Examples

The following example returns a list of commands that can be redone:

```
redo -list
```

## See Also

- [undo](#)
- [startgroup](#)
- [endgroup](#)

## refresh\_design

Refresh the current design.

### Syntax

```
refresh_design [-part arg] [-quiet] [-verbose]
```

### Returns

Nothing

### Usage

Name	Description
<code>[-part]</code>	Target part
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

### Categories

[Project](#)

### Description

Reloads the current design from the project data on the hard drive. This overwrites the in-memory view of the design to undo any recent design changes.

### Arguments

**-part** *arg* - (Optional) The new target part for the design when it is reloaded. This overrides the constraint file part specified in the project data on the hard drive.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the `set_msg_limit` command.

### Examples

The following command reloads the current design from the project data on hard disk. This will overwrite the unsaved changes of the design which are in memory.

```
refresh_design
```

**Note** You can use the command to undo a series of changes to the design and revert to the previously saved design.

The following example refreshes the current design using the specified V6 part as the target device. The second command is required to make the selected part the target device for the active implementation run.

```
refresh_design -part xc6vcx75tff784-1  
set_property part xc6vcx75tff784-1 [get_runs impl_6]
```

**Note** The second command is not required if the target part is not changed.

## See Also

[set\\_property](#)

## register\_drc\_rule

Register a user defined drc rule.

### Syntax

```
register_drc_rule [-category arg] -name arg [-desc arg]
[-msg arg] [-checker arg] [-quiet] [-verbose]
```

### Returns

Nothing

### Usage

Name	Description
<i>[-category]</i>	Specify the category for this rule. This is the major grouping for your rule. It is optional and will default to User Defined. Default: User Defined
<b>-name</b>	Specify the name for this rule. This is the typically a 4-6 letter specification for your rule.
<i>[-desc]</i>	Specify the short description for this rule. It is optional and will default to User rule - default description . Default: User rule - default description
<i>[-msg]</i>	Specify the full description for this rule. Including the substitutions. Values are: %MSG_STRING %NETLIST_ELEMENT %SITE_GROUP %CLOCK_REGION %BANK.
<i>[-checker]</i>	The string name of the Tcl proc - the rule name. Also referred to as a checker.
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution

### Categories

[Report](#)

### Description

Register a new user-defined DRC rule into the violation library. This allows you to define a rule to use for checking designs against.

There is a significant level of understanding required to create and register user-defined DRC rules. You must understand the details of the DRC requirements in order to be successful in this effort.

### Arguments

**-category** *arg* - (Optional) Defines a grouping for the rule. The default is "User Defined". This is used as the first level of hierarchy in the GUI when listing DRC rules.

**-name** *arg* - The unique abbreviation for the rule in the violation library. This should match the abbreviation given to the violation in the **create\_violation** command.

**-desc** *arg* - (Optional) A brief description of the rule. The default is "User Rule". This is displayed in the GUI when listing DRC rules. The description is also used in the DRC report and summary.

**-msg** *arg* - (Optional) This is the message displayed when a violation of the rule is found. The message can include dynamic substitutions with design elements found in violation of the rule when the DRC report is generated. Valid substitutions keys are:

- %MSG\_STRING - String.
- %NETLIST\_ELEMENT - Netlist elements including cells, pins, ports, and nets.
- %SITE\_GROUP - Device site.
- %CLOCK\_REGION - Clock region.
- %BANK - Package IO bank.

**-checker** *arg* - (Optional) This is the name of the Tcl procedure which defines the rule checking functionality. The Tcl proc must be separately defined and can be loaded with the **source** command. The Tcl checker procedure can create a violation object that defines the message to be returned by the DRC check, and also contains the design elements which are the subject of a DRC rule violation. The violation object is created by the **create\_violation** command, that maps the violation to the DRC rule registered by this command.

**Note** A single DRC abbreviation (**-name**) can have multiple rules, or checkers

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

## Examples

The following example registers the **cell\_check** Tcl procedure as a checker for the **CLCHK** user-defined DRC rule, providing a category for the check, with a brief description and long message for the rule:

```
register_drc_rule -name CLCHK -category SHARED RULES \
  -desc {Rule CLCHK: Cell check.} -msg {Violation %MSG_STRING. \
    For %NETLIST_ELEMENT cell. Again %NETLIST_ELEMENT cell. } \
  -checker cell_check
```

In the following example the rule named **NETV** is registered with the default category (**User Defined**), with a message, and the **net\_check** Tcl rule checker procedure:

```
register_drc_rule -name NETV -msg {%Msg_String. For %NETLIST_ELEMENT net.} \
  -checker net_check
```

The following is a sample Tcl checker proc definition, incorporating the **create\_violation** command:

```
proc cell_check {} {  
    puts "Executing DRC rule cell_check."  
    set mycell3 [lindex [get_cells] 3]  
    set mycell5 [lindex [get_cells] 5]  
    puts $mycell3  
    set vio_1 [create_violation -abbrev CLCHK -msg "Rule 1: failed." \  
        -severity ERROR $mycell3 $mycell5]  
    set vio_2 [create_violation -abbrev CLCHK -msg "Rule 1: failed again." \  
        -severity ERROR $mycell5 $mycell3]  
    set x [list $vio_1 $vio_2]  
    return -code error $x  
}
```

## See Also

- [report\\_drc](#)
- [report\\_user\\_drc\\_rule](#)
- [reset\\_drc](#)
- [create\\_violation](#)

## reimport\_files

Reimport files when they are found out-of-date.

### Syntax

```
reimport_files [-force] [-quiet] [-verbose] [files ...]
```

### Returns

List of file objects that were imported

### Usage

Name	Description
<i>[-force]</i>	Force a reimport to happen even when the local files may be newer
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ files ]</i>	List of files to reimport. If no files are specified, all files in the project that are out-of-date, will be reimported

### Categories

Project

### Description

Reimports project files. This updates the local project files from the original referenced source files.

### Arguments

**-force** - (Optional) Reimport files even when the local project files may be newer than their referenced source files.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*files* - (Optional) List of files to reimport. If no files are specified, all files in the project that are out-of-date, will be reimported. If you use **-force** and specify no files, all files in the project will be reimported.

## Examples

The following example reimports all project files regardless of whether they are out of date, or the local files are newer than the referenced source file:

```
reimport_files -force
```

**Note** No warnings will be issued for newer local files that will be overwritten.

The following example reimports the specified files to the project, but only if the original source file is newer than the local project file:

```
reimport_files C:/Data/FPGA_Design/source1.v C:/Data/FPGA_Design/source2.vhdl
```

## See Also

- [add\\_files](#)
- [import\\_files](#)

## remove\_cells\_from\_pblock

Remove cells from a Pblock.

### Syntax

```
remove_cells_from_pblock [-quiet] [-verbose] pblock cells ...
```

### Returns

Nothing

### Usage

Name	Description
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>pblock</i>	Pblock to remove cells from
<i>cells</i>	Cells to remove

### Categories

[Floorplan](#), [XDC](#)

### Description

Removes the specified logic instances from a Pblock. Cells are added to a Pblock with the **add\_cells\_to\_pblock** command.

**Note** Cells that have been placed will not be unplaced as they are removed from a Pblock. Any current LOC assignments are left intact.

### Arguments

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*pblock* - The name of the Pblock from which to remove the specified instances.

*cells* - One or more cell objects to remove from the specified Pblock.

### Examples

The following example removes the specified cells from the pb\_cpuEngine Pblock:

```
remove_cells_from_pblock pb_cpuEngine [get_cells cpuEngine/cpu_dwb_dat_o/*]
```

## See Also

[add\\_cells\\_to\\_pblock](#)

## remove\_files

Remove files or directories from a fileset.

### Syntax

```
remove_files [-fileset arg] [-quiet] [-verbose] [files ...]
```

### Returns

List of files that were removed

### Usage

Name	Description
<i>[-fileset]</i>	Fileset name
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>[ files ]</i>	Name of the files and/or directories to remove

### Categories

[Project](#), [Simulation](#)

### Description

Removes files from the specified fileset. Files must be specified with the full path to the file.

### Arguments

**-fileset** *arg* - The name of the fileset from which to remove files. The default is the `sources_1` source fileset.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns `TCL_OK` regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the `set_msg_limit` command.

### Examples

The following example removes the file named `C:/Design/top.ucf` from the constraint set `constrs_1`:

```
remove_files -fileset constrs_1 C:/Design/top.ucf
```

Multiple files can be specified as follows:

```
remove_files -fileset sim_1 top_tbt1.vhdl top_tbt2.vhdl
```

## See Also

[add\\_files](#)

## remove\_pin

Remove pins from the current design.

### Syntax

```
remove_pin [-quiet] [-verbose] pins ...
```

### Returns

Nothing

### Usage

Name	Description
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>pins</i>	List of pins to remove

### Categories

[Netlist](#)

### Description

Remove pins from the current netlist in either an open Synthesized or Implemented design.

Netlist editing changes the in-memory view of the netlist in the current design. It does not change the files in the source fileset, or change the persistent design on the disk. Changes made to the netlist may be saved to a design checkpoint using the **write\_checkpoint** command, or may be exported to a netlist file such as Verilog, VHDL, or EDIF, using the appropriate **write\_\*** command.

**Note** Netlist editing is not allowed on an RTL design.

### Arguments

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*pins* - List of pins to remove from the netlist. The pins must be specified hierarchically by the cell instance the pin is found on.

## Examples

The following example removes the fftEngine from the in-memory netlist of the current design:

```
remove_cell fftEngine
```

## See Also

- [create\\_cell](#)
- [write\\_checkpoint](#)
- [write\\_edif](#)
- [write\\_verilog](#)
- [write\\_vhdl](#)

## reorder\_files

Change the order of source files in the active fileset.

### Syntax

```
reorder_files [-fileset arg] [-before arg] [-after arg] [-front]
[-back] [-auto] [-disable_unused] [-quiet] [-verbose] files ...
```

### Returns

Nothing

### Usage

Name	Description
<i>[-fileset]</i>	Fileset to reorder
<i>[-before]</i>	Move the listed files before this file
<i>[-after]</i>	Move the listed files after this file
<i>[-front]</i>	Move the listed files to the front (default)
<i>[-back]</i>	Move the listed files to the back
<i>[-auto]</i>	Automatically re-orders the given fileset
<i>[-disable_unused]</i>	Disables all files not associated with the TOP design unit
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>files</i>	Files to move

### Categories

[Project](#)

### Description

Reorders source files in the specified fileset. Takes the files indicated and places them at the front of, the back of, or before or after other files within the fileset. This command also has an auto reorder feature that reorders the files based on the requirements of the current top module in the design.

### Arguments

**-fileset** *arg* - The fileset in which to reorder files. The default is the sources\_1 source fileset.

**-before** *arg* - Place the specified files before this file in the fileset. The file must be specified with the full path name in the fileset.

**-after** *arg* - Place the specified files after this file in the fileset. The file must be specified with the full path name in the fileset.

**-front** - Place the specified files at the front of the list of files in the fileset.

- back** - Place the specified files at the back of the list of files in the fileset.
  - auto** - Enable automatic reordering based on the hierarchy requirements of the current top-module in the project. Often used after changing the top module with the **"set\_property top"** command.
  - disable\_unused** - Disable any files not currently used by the hierarchy based on the top-module. Often used after changing the top module with the **"set\_property top"** command.
  - quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.
  - verbose** - (Optional) Temporarily override any message limits and return all messages from this command.
- Note** Message limits can be defined with the **set\_msg\_limit** command.
- files* - One or more files to relocate in the fileset. Files must be specified by their full path name in the fileset, and are reordered in the order they are specified.

## Examples

The following example takes the specified files and moves them to the front of the source fileset:

```
reorder_files -front {C:/Data/FPGA/file1.vhdl C:/Data/FPGA/file2.vhdl}
```

**Note** The default source fileset is used in the preceding example since the **-fileset** argument is not specified.

The following example sets a new top\_module in the design, and then automatically reorders and disables unused files based on the hierarchy of the new top-module:

```
set_property top block1 [current_fileset]
reorder_files -auto -disable_unused
```

## See Also

- [add\\_files](#)
- [create\\_fileset](#)
- [current\\_fileset](#)
- [remove\\_files](#)

## report\_carry\_chains

Report carry chains.

### Syntax

```
report_carry_chains [-file arg] [-return_string]  
[-max_chains arg] [-quiet] [-verbose]
```

### Returns

Report

### Usage

Name	Description
<i>[-file]</i>	Filename to output results to. (send output to console if -file is not used)
<i>[-return_string]</i>	return report as string
<i>[-max_chains]</i>	Number of chains for which report is to be generated Default: 1
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution

### Categories

[Report](#)

## report\_clock\_interaction

Report on inter clock timing paths and unlocked registers.

### Syntax

```
report_clock_interaction [-delay_type arg] [-setup] [-hold]
[-significant_digits arg] [-file arg] [-append] [-name arg]
[-return_string] [-quiet] [-verbose]
```

### Returns

Nothing

### Usage

Name	Description
<i>[-delay_type]</i>	Type of path delay: Values: max, min, min_max Default: max
<i>[-setup]</i>	Consider max delay timing paths (equivalent to -delay_type max)
<i>[-hold]</i>	Consider min delay timing paths (equivalent to -delay_type min)
<i>[-significant_digits]</i>	Number of digits to display: Range: 0 to 13 Default: 2
<i>[-file]</i>	Filename to output results to. (send output to console if -file is not used)
<i>[-append]</i>	Append the results to file, don't overwrite the results file
<i>[-name]</i>	Output the results to GUI panel with this name
<i>[-return_string]</i>	Return report as string
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution

### Categories

[Report](#)

### Description

Reports clock interactions and signals that cross clock domains to identify potential problems such a metastability or data loss or incoherency some visibility into the paths that cross clock domains is beneficial. This command requires an open synthesized or implemented design.

**Note** By default the report is written to the Tcl console or STD output. However, the results can also be written to a file or returned as a string if desired.

## Arguments

**-delay\_type** *arg* - Specifies the type of delay to analyze when running the clock interaction report. The valid values are min, max, and min\_max. The default setting for **-delay\_type** is max.

**-setup** - Check for setup violations. This is the same as specifying **-delay\_type max**.

**-hold** - Check for hold violations. This is the same as specifying **-delay\_type min**.

**Note** **-setup** and **-hold** can be specified together, which is the same as specifying **-delay\_type min\_max**

**-significant\_digits** *arg* - The number of significant digits in the output results. The valid range is 0 to 13. The default setting is 2 significant digits.

**-file** *arg* - (Optional) Write the report into the specified file.

**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** - 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

**-name** *arg* - The name of the Clock Interaction Report view to display in the PlanAhead tool GUI mode. If the name has already been used in an open Report view, that view will be closed and a new report opened.

**-return\_string** - 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, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

## Examples

The following example sets the model for interconnect delay, selects a device speed grade, and then runs **report\_clock\_interaction**:

```
set_delay_model -interconnect none
set_speed_grade -3
report_clock_interaction -delay_type min_max -significant_digits 3 -name "results_1"
```

The following example returns the clock interactions, writing the report to the GUI, to the specified file, and returns a string which is assigned to the specified variable:

```
set clk_int [report_clock_interaction -file clk_int.txt -name clk_int1 \
-return_string]
```

## See Also

- [create\\_clock](#)
- [create\\_generated\\_clock](#)
- [report\\_clocks](#)
- [set\\_delay\\_model](#)
- [set\\_speed\\_grade](#)











**-cells** *args* - (Optional) Report control sets for the specified cell objects.

**-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, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

**-verbose** - Provide a detailed report listing all control sets of the specified objects. Temporarily override any message limits and return all output from the command.

## Examples

The following example reports the control sets of the current design, sorted by the clk and clkEn signals:

```
report_control_sets -verbose -sort_by {clk clkEn}
```

The following example reports the control sets of the specified cells, sorted by clk and set:

```
report_control_sets -verbose -sort_by {clk set} -cells [get_cells usb*]
```



## Examples

The following example writes the debug core report to the specified file name at the specified location:

```
report_debug_core -file C:/Data/FPGA_Design/project_1_cores.txt
```

## See Also

- [create\\_debug\\_core](#)
- [read\\_chipscope\\_cdc](#)







**-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, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

## Examples

The following example returns the available list of DRC rules to run against the current design:

```
report_drc -list_rules
```

The following example runs the specified DRC rules against the current design, and writes the output to the specified file:

```
report_drc -rules {IOCNT-1 IOPCPR-1 IOPCMGT-1 IOCTMGT-1 IODIR-1} \  
-file C:/Data/DRC_Rpt1.txt
```

## See Also

- [register\\_drc\\_rule](#)
- [report\\_user\\_drc\\_rule](#)
- [reset\\_drc](#)
- [create\\_violation](#)





## Examples

The following example reports the IO blocks of the current design:

```
report_io
```

## See Also

[report\\_route\\_status](#)





## Examples

Specify an industrial temperature grade device with an ambient temperature of 75 degrees C and then write those settings to a file on disk.

```
set_operating_conditions -grade industrial -junction_temp 75
report_operating_conditions -grade -junction_temp -return_string -file \
~/conditions.txt
```

## See Also

[set\\_operating\\_conditions](#)



## See Also

- [get\\_param](#)
- [list\\_param](#)
- [reset\\_param](#)
- [set\\_param](#)



## Arguments

**-no\_propagation** - (Optional) For all undefined nodes power analysis uses a vectorless propagation engine to estimate activity. This argument disables the propagation engine for a faster analysis of the design.

**-file** *arg* - (Optional) Write the report into the specified file.

**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.

**-name** *arg* - (Optional) Specifies the name of the results set to report the results to.

**-xpe** *arg* - (Optional) Output the results to an XML file for importing into XPE.

**-return\_string** - 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, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

## Examples

The following example performs power analysis, without net propagation, and writes the results to an XML file for use in XPE:

```
report_power -no_propagation -xpe C:/Data/design1.txt
```

## See Also

- [read\\_saif](#)
- [read\\_vcd](#)
- [set\\_default\\_switching\\_activity](#)
- [set\\_operating\\_conditions](#)



## Examples

The following example returns all properties of the specified object:

```
report_property -all [get_cells cpuEngine]
```

To determine what properties are available for the different design objects supported by the tool, you can use multiple **report\_property** commands in sequence. The following example returns all properties of the specified current objects:

```
report_property -all [current_project]  
report_property -all [current_fileset]  
report_property -all [current_design]  
report_property -all [current_instance]  
report_property -all [current_run]
```

## See Also

- [create\\_property](#)
- [get\\_cells](#)
- [get\\_property](#)
- [list\\_property](#)
- [list\\_property\\_value](#)
- [reset\\_property](#)
- [set\\_property](#)



**-file** *arg* - (Optional) Write the resource report into the specified file.

**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.

**-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.

**-format [ table | xml ]** - (Optional) Specifies the format of the output as either a table or XML. The default output is table.

**Note** The format applies when **-file** is specified, but is otherwise ignored.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

## Examples

The following example reports the resources of the design hierarchy to two levels, and writes the results to the specified file:

```
report_resources -hierarchy -levels 2 -file C:/Data/resources_2.txt
```

## See Also

[report\\_utilization](#)



**-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.

**Note** This argument cannot be used with the **-file** option.

**-file *arg*** - (Optional) Write the SSN report into the specified file.

**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.

**-format [ csv | html ]** - (Optional) Specifies the format of the output as either comma-separated values (CSV), or HTML. The default output is CSV.

**Note** The format applies when **-file** is specified, but is otherwise ignored.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

## Examples

The following example performs an SSN analysis on the current synthesis or implemented design, formats the output as HTML, and writes the output to the specified file:

```
report_ssn -format html -file C:/Data/devSSN.html
```

The following example performs an SSN analysis on the current synthesis or implemented design, and returns the output to a string which is stored in the specified variable:

```
set devSSN [report_ssn -format html -return_string]
```

**Note** The **-format** argument is ignored in the absence of **-file**.

## See Also

- [report\\_sso](#)
- [reset\\_ssn](#)
- [reset\\_sso](#)





## Examples

The following example performs an SSO analysis on the current synthesis or implemented design, using the default settings for board and device, and writes the output to the specified file:

```
report_sso -file C:/Data/devSSO.txt
```

The following example performs an SSO analysis on the current synthesis or implemented design, using the specified values to define the system-level PCB:

```
report_sso -name sso_1 -file C:/Data/mySSO1.txt \  
-board_thickness 55 -via_diameter 9 -pad_to_via_breakout_length 75 \  
-breakout_width 9 -other_pcb_inductance 37 -socket_inductance 19 \  
-ground_bounce 537 -output_cap 12
```

## See Also

- [report\\_ssn](#)
- [reset\\_ssn](#)
- [reset\\_sso](#)

## report\_stats

Report Statistics.

### Syntax

```
report_stats [-file arg] [-cell arg] [-pblock arg]
[-clock_region arg] [-format arg] [-level arg] [-all]
[-tables args] [-quiet] [-verbose]
```

### Returns

Report

### Usage

Name	Description
<i>[-file]</i>	Filename to which results will be written. Writes console if not specified
<i>[-cell]</i>	Write statistics for the specified cell
<i>[-pblock]</i>	Write statistics for the specified Pblock
<i>[-clock_region]</i>	Write statistics for the specified clock region
<i>[-format]</i>	Report format Values: TABLE, CSV, XML Default: TABLE
<i>[-level]</i>	Report level (used with '-cell' or '-pblock') Default: 1
<i>[-all]</i>	Report all levels (used with '-cell' or '-pblock')
<i>[-tables]</i>	List of table types Values: rtlMacro, rtlPrimitive, rtlHierarchy, rtlMemory, rtlPower, rtlPower2, primitive, netBoundary, carryChain, physicalResource, io, RPM, clock, PRModule, PRModule, pblockOverlap, ioBank
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution

### Categories

[Report](#)



**-return\_string** - (Optional) Returns the data as a text string for assignment to a Tcl variable.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*objects* - Specifies the list of objects to return switching activity information on.

## Examples

The following example reports the signal\_rate and static probability value on all output ports in the design:

```
report_switching_activity -signal_rate -static_probability [all_outputs]
```

## See Also

- [power\\_opt\\_design](#)
- [report\\_default\\_switching\\_activity](#)
- [report\\_power](#)
- [reset\\_default\\_switching\\_activity](#)
- [reset\\_switching\\_activity](#)
- [set\\_default\\_switching\\_activity](#)
- [set\\_switching\\_activity](#)

## report\_timing

Report timing paths.

### Syntax

```
report_timing [-from args] [-rise_from args] [-fall_from args]
[-to args] [-rise_to args] [-fall_to args] [-through args]
[-rise_through args] [-fall_through args] [-delay_type arg]
[-setup] [-hold] [-max_paths arg] [-nworst arg]
[-path_type arg] [-input_pins] [-nets] [-slack_lesser_than arg]
[-slack_greater_than arg] [-group args] [-sort_by arg]
[-no_report_unconstrained] [-match_style arg] [-of_objects args]
[-significant_digits arg] [-file arg] [-append] [-name arg]
[-return_string] [-quiet] [-verbose]
```

### Returns

Nothing

### Usage

Name	Description
<code>[-from]</code>	From pins, ports, cells or clocks
<code>[-rise_from]</code>	Rising from pins, ports, cells or clocks
<code>[-fall_from]</code>	Falling from pins, ports, cells or clocks
<code>[-to]</code>	To pins, ports, cells or clocks
<code>[-rise_to]</code>	Rising to pins, ports, cells or clocks
<code>[-fall_to]</code>	Falling to pins, ports, cells or clocks
<code>[-through]</code>	Through pins, ports, cells or nets
<code>[-rise_through]</code>	Rising through pins, ports, cells or nets
<code>[-fall_through]</code>	Falling through pins, ports, cells or nets
<code>[-delay_type]</code>	Type of path delay: Values: max, min, min_max, max_rise, max_fall, min_rise, min_fall Default: max
<code>[-setup]</code>	Report max delay timing paths (equivalent to <code>-delay_type max</code> )
<code>[-hold]</code>	Report min delay timing paths (equivalent to <code>-delay_type min</code> )
<code>[-max_paths]</code>	Maximum number of paths to output when sorted by slack, or per path group when sorted by group: Value =1 Default: 1
<code>[-nworst]</code>	List up to N worst paths to endpoint: Value =1 Default: 1
<code>[-path_type]</code>	Format for path report: Values: end summary short full full_clock full_clock_expanded Default: full_clock_expanded
<code>[-input_pins]</code>	Show input pins in path
<code>[-nets]</code>	List net names

Name	Description
<code>[-slack_lesser_than]</code>	Display paths with slack less than this Default: 1e+30
<code>[-slack_greater_than]</code>	Display paths with slack greater than this Default: -1e+30
<code>[-group]</code>	Limit report to paths in this group(s)
<code>[-sort_by]</code>	Sorting order of paths: Values: group, slack Default: slack
<code>[-no_report_unconstrained]</code>	Do not report unconstrained paths
<code>[-match_style]</code>	Style of pattern matching, valid values are ucf, sdc Default: ucf
<code>[-of_objects]</code>	Report timing for these paths
<code>[-significant_digits]</code>	Number of digits to display: Range: 0 to 13 Default: 3
<code>[-file]</code>	Filename to output results to. (send output to console if -file is not used)
<code>[-append]</code>	Append the results to file, don't overwrite the results file
<code>[-name]</code>	Output the results to GUI panel with this name
<code>[-return_string]</code>	return report as string
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

## Categories

Report, Timing

## Description

This command performs timing analysis on the specified timing paths of the current Synthesized or Implemented Design. By default the tool reports the timing path with the worst calculated slack within each path group. However, you can optionally increase the number of paths and delays reported with the use of the **-nworst** or **-max\_paths** arguments.

The timing engine runs in quad timing mode, analyzing min and max delays for both slow and fast corners. You can configure the type of analysis performed by the **config\_timing\_corners** command. However, it is not recommended to change the default because this reduces the timing analysis coverage.

**Note** By default the report is written to the Tcl console or STD output. However, the results can also be written to a file or returned as a string if desired.

## Arguments

**-from** *args* - (Optional) The starting points of the timing paths to be analyzed. Ports, pins, or cells can be specified as timing path startpoints. You can also specify a clock object, and all startpoints clocked by the named clock will be analyzed.





**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

## Examples

The following example reports the timing for the 5 worst paths in the design, reporting the full timing path, including input pins, with timing values:

```
report_timing -nworst 5 -path_type full -input_pins
```

The following example shows the use of the multiple through points to define both a specific path (through state\_reg1) and alternate paths (through count\_3 or count\_4), and writes the timing results to the specified file:

```
report_timing -from go -through {state_reg1} -through { count_3 count_4 } \  
-to done -path_type summary -file C:/Data/timing1.txt
```

## See Also

- [get\\_path\\_groups](#)
- [get\\_timing\\_paths](#)
- [group\\_path](#)
- [report\\_timing\\_summary](#)
- [set\\_msg\\_limit](#)

## report\_transformed\_primitives

Report details of Unisim primitive transformations.

### Syntax

```
report_transformed_primitives [-file arg] [-return_string]  
[-quiet] [-verbose]
```

### Returns

Nothing

### Usage

Name	Description
<i>[-file]</i>	Output file
<i>[-return_string]</i>	return report as string
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution

### Categories

[Report](#)



**-rules** *args* - The list of rules to enable or disable. The **-rules** argument is used with the **-action** argument to specify the rules to enable or disable for the specified abbreviation. The specified rules must be associated with the abbreviation, or an error will be returned. Rules are associated with abbreviations using the **register\_drc\_rule** command.

**-list\_rules** [ **all** | **enabled** ] - List the rules associated with the specified abbreviation. List **all** rules, or only those that are currently **enabled**. Use **-list\_rules** in place of the **-action** and **-rules** arguments to list the rules currently registered with the specified DRC abbreviation.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

## Examples

The following example lists all rules currently registered with the specified abbreviation:

```
report_user_drc_rule -abbrev ABCD -list_rules all
```

The following example disables both rule1 and rule2 for the specified abbreviation:

```
report_user_drc_rule -abbrev ABCD -enable unregister -rules {rule1 rule2}
```

The following example enables rule1 for the specified abbreviation:

```
report_user_drc_rule -abbrev ABCD -enable register -rules {rule1}
```

## See Also

- [register\\_drc\\_rule](#)
- [report\\_drc](#)
- [reset\\_drc](#)
- [create\\_violation](#)

## report\_utilization

Compute utilization of device and display report.

### Syntax

```
report_utilization [-file arg] [-append] [-pblocks args]
[-cells args] [-return_string] [-packthru] [-name arg]
[-no_primitives] [-omit_locs] [-quiet] [-verbose]
```

### Returns

Report

### Usage

Name	Description
<code>[-file]</code>	Filename to output results to. (send output to console if -file is not used)
<code>[-append]</code>	Append the results to file, don't overwrite the results file
<code>[-pblocks]</code>	Report utilization of given list of pblocks
<code>[-cells]</code>	Report utilization of given list of cells
<code>[-return_string]</code>	Return report as string
<code>[-packthru]</code>	Reports LUTs used exclusively as pack-thru
<code>[-name]</code>	Output the results to GUI panel with this name
<code>[-no_primitives]</code>	Removes "Primitives Section" from report_utilization o/p.
<code>[-omit_locs]</code>	Removes "Loced" column from report_utilization o/p.
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

### Categories

Report

### Description

Report the utilization of resources on the target part by the current synthesized or implemented design. The report is returned to the standard output, unless the **-file**, **-return\_string**, or **-name** arguments are specified.

Though resource utilization can be reported early in the design process, the report will be more accurate as the design progresses from synthesis through implementation.





## Examples

The following example resets the toggle rate and static probability value on all design output ports:

```
reset_default_switching_activity -toggle_rate -static_probability all
```

## See Also

- [power\\_opt\\_design](#)
- [report\\_default\\_switching\\_activity](#)
- [report\\_power](#)
- [report\\_default\\_switching\\_activity](#)
- [report\\_switching\\_activity](#)
- [reset\\_switching\\_activity](#)
- [set\\_default\\_switching\\_activity](#)
- [set\\_switching\\_activity](#)



## reset\_msg\_count

Reset message count.

### Syntax

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

### Returns

New message count

### Usage

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

### Categories

[Report](#)

### Description

Reset the message count for the specified message ID to 0. This restarts the message counter toward the specified message limit. This can be used to reset the count of specific messages that may be reaching the limit, or reset the count of all messages returned by the tool.

You can get the current message count for a specific message ID using the **get\_msg\_count** command.

### Arguments

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

**-id arg** - Specifies the message ID to reset the count to 0. Specify \* to reset the count of all messages to 0.

### Examples

The following example resets the message count for all messages:

```
reset_msg_count -id *
```

## See Also

- [get\\_msg\\_count](#)
- [set\\_msg\\_limit](#)

## reset\_msg\_limit

Reset message limit.

### Syntax

```
reset_msg_limit [-severity arg] [-id arg] [-quiet] [-verbose]
```

### Returns

New message limit

### Usage

Name	Description
<i>[-severity]</i>	Message severity to be reset (not valid with -id,) e.g. "ERROR" or "CRITICAL WARNING" Default: ALL
<i>[-id]</i>	Unique message Id to be reset (not valid with -severity,) e.g Common-99
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution

### Categories

[Report](#)

### Description

Restores the default message limit. The command can be used to restore the default limit for a specific message ID, for a specific message severity, or for all messages returned. The current default limit for all messages returned is 4,294,967,295.

By default this command returns the message limit for all messages. You can also get the limit of a specific severity of message, or for a specific message ID. For instance the following are two messages returned under different circumstances:

```
INFO: [common-99] This is an example INFO message
CRITICAL WARNING: [Netlist-1129] This message is a CRITICAL WARNING and
requires user attention
```

**Note** You can change the severity of a specific message ID with the `set_msg_severity` command.

### Arguments

**-id value** - The ID of a specific message for which to change the message limit. Each message returned contains its own ID. For instance, the message IDs above are common-99 and Netlist-1129.

**-severity** *value* - The severity of the message. There are five message severities:

- **ERROR** - An ERROR condition implies an issue has been encountered which will render design results unusable and cannot be resolved without user intervention.
- **{CRITICAL WARNING}** - A CRITICAL WARNING message indicates that certain input/constraints will either not be applied or are outside the best practices for a FPGA family. User action is strongly recommended.

**Note** Since this is a two word value, it must be enclosed in {}.

- **WARNING** - A WARNING message indicates that design results may be sub-optimal because constraints or specifications may not be applied as intended. User action may be taken or may be reserved.
- **INFO** - An INFO message is the same as a STATUS message, but includes a severity and message ID tag. An INFO message includes a message ID to allow further investigation through answer records if needed.
- **STATUS** - A STATUS message communicates general status of the process and feedback to the user regarding design processing. A STATUS message does not include a message ID.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

## Examples

The following example resets the default limit for all messages:

```
reset_msg_limit
```

**Note** The default limit is 4,294,967,295.

The following example resets the message limit of the specified message ID:

```
reset_msg_limit -id Netlist-1129
```

## See Also

- [get\\_msg\\_limit](#)
- [set\\_msg\\_limit](#)
- [set\\_msg\\_severity](#)



The following example restores the severity of message ID **Netlist-1129** to its original severity:

```
reset_msg_severity Netlist-1129
```

## See Also

[set\\_msg\\_severity](#)













































































































## Examples

The following example swaps the instances assigned to the two specified device sites:

```
swap_locs [get_sites {OLOGIC_X2Y1}] [get_sites {OLOGIC_X2Y0}]
```

## See Also

- [get\\_cells](#)
- [get\\_ports](#)
- [get\\_sites](#)



## See Also

- [redo](#)
- [startgroup](#)
- [endgroup](#)







## Examples

The following example unmarks the selected objects:

```
unmark_objects [get_selected_objects]
```

The following example unmarks all objects currently marked in the color yellow:

```
unmark_objects -color yellow
```

## See Also

- [get\\_selected\\_objects](#)
- [mark\\_objects](#)



## See Also

- [create\\_cell](#)
- [place\\_cell](#)
- [remove\\_cell](#)



The following example unselects all currently selected objects:

```
unselect_objects
```

## See Also

- [get\\_selected\\_objects](#)
- [select\\_objects](#)

## update\_files

Update files in the project with same named files from the files or directories specified.

### Syntax

```
update_files [-from_files args] [-norecurse] [-to_files args]
[-filesets args] [-force] [-report_only] [-quiet] [-verbose]
```

### Returns

List of the files updated

### Usage

Name	Description
<i>[-from_files]</i>	New files and directories to use for updating
<i>[-norecurse]</i>	Recursively search in specified directories
<i>[-to_files]</i>	Existing project files and directories to limit updates to
<i>[-filesets]</i>	Fileset name
<i>[-force]</i>	Overwrite imported files in the project, even if read-only, if possible
<i>[-report_only]</i>	Do no actual file updates, but report on updates that otherwise would have been made
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution

### Categories

[Project](#)

## update\_ip\_catalog

Update the IP Catalog. Before executing this command optionally use the following to set repository paths: 'set\_property ip\_repo\_paths repo\_path\_list [current\_fileset]'.

### Syntax

```
update_ip_catalog [-rebuild] [-add_ip arg] [-delete_ip arg]
[-repo_path arg] [-quiet] [-verbose]
```

### Returns

True for success

### Usage

Name	Description
<code>[-rebuild]</code>	Trigger a rebuild of the specified repository's index file or rebuild all repositories if none specified
<code>[-add_ip]</code>	Add the specified IP into the specified repository Values: Either a path to the IP's component.xml or to a zip file containing the IP
<code>[-delete_ip]</code>	Remove the specified IP from the specified repository Values: Either a path to the IP to be removed or its VLNV
<code>[-repo_path]</code>	Used in conjunction with rebuild, add_ip or delete_ip to specify the path of the repository on which to operate
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

### Categories

IPFlow

### Description

Update the IP Catalog associated with the current design.

The default IP catalog is found in the installation hierarchy of the tool. However, you can also add new IP repository paths to the IP catalog using the **set\_property** command to define the IP\_REPO\_PATHS property of the current Source fileset.

The Xilinx IP catalog, or repository, is located in the installation hierarchy of the software release being used. You can also add custom IP to the repository by using the **set\_property** command to set the IP\_REPO\_PATHS property on the source fileset to point to the locations of custom IP, as shown in the example below.

### Arguments

**-rebuild** - (Optional) Rebuild the complete IP Catalog index, or just rebuild the index for the IP repository specified by the **-repo\_path**.

**-add\_ip** *arg* - Add an individual IP core to the specified IP repository. This argument requires the **-repo\_path** argument to also be specified. The IP must be in the form of a component .xml or a zipped file.

**-delete\_ip** *arg* - Remove an IP core IP from the specified IP repository. This argument requires the **-repo\_path** argument to also be specified.

**-repo\_path** *arg* - Specify the directory name of an IP repository to add to or delete from, or to rebuild the index for.

**Note** The IP repository must have been previously added to the current Source fileset using the **set\_property** command. See the example below

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

## Examples

The following example sets the IP\_REPO\_PATHS property of the current Source fileset, to add an IP repository, then rebuilds the IP catalog index for the whole IP catalog:

```
set_property ip_repo_paths C:/Data/IP_LIB [current_fileset]
update_ip_catalog -rebuild
```

## See Also

- [create\\_ip](#)
- [import\\_ip](#)
- [validate\\_ip](#)

## update\_timing

Update timing.

### Syntax

```
update_timing [-full] [-quiet] [-verbose]
```

### Returns

Nothing

### Usage

Name	Description
<i>[-full]</i>	Perform a full timing update instead of an incremental one
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution

### Categories

[Timing](#)

## upgrade\_ip

Upgrade a configurable IP to a later version.

### Syntax

```
upgrade_ip [-srcset arg] [-latest arg] [-ips args] [-quiet]
[-verbose]
```

### Returns

List of files that were upgraded

### Usage

Name	Description
<code>[-srcset]</code>	Source set name
<code>[-latest]</code>	Upgrade the IP to the latest version
<code>[-ips]</code>	IP to be upgraded
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution

### Categories

[IPFlow](#)

### Description

This command upgrades the specified IP cores from an older version to the latest version in the IP catalog.

You can only upgrade legacy IP that explicitly supports upgrading. The `UPGRADE_VERSIONS` property on the `ipdef` object indicates if there are upgrade version for an IP core.

**Note** Not all legacy IP support upgrading, and native IP cannot be upgraded

### Arguments

**-ips** *arg* - (Optional) Specifies which legacy IP cores should be upgraded.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns `TCL_OK` regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the `set_msg_limit` command.

### Examples

The following example upgrades all IP cores in the current project to the latest version:

```
upgrade_ip -ips *
```

## See Also

- [create\\_ip](#)
- [import\\_ip](#)

## verify\_config

Analyze implemented runs to ensure they follow rules required for partial reconfiguration.

### Syntax

```
verify_config [-file arg] [-quiet] [-verbose] runs ...
```

### Returns

Pr report

### Usage

Name	Description
<i>[-file]</i>	Output report file name
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>runs</i>	List of implemented Runs to be verified

### Categories

[PartialReconfiguration](#)

### Description

Analyzes the implemented configurations of a design to ensure they follow rules required for partial reconfiguration.

### Arguments

**-file** *arg* - (Optional) Output the results of this command to the specified log file.

**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.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*runs* - The list of two or more implemented configuration runs to verify. These specify the runs used to implement the different configurations of the design.

### Examples

The following example checks two runs, **config\_2** and **config\_3**, for any errors and writes the log into the specified log file:

```
verify_config -runs {config_2 config_3} -file pr_verify.log
```

## See Also

[create\\_reconfig\\_module](#)

## version

Returns the build for Vivado and the build date.

### Syntax

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

### Returns

Vivado version

### Usage

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

### Categories

[Report](#)

### Description

Returns the version number of the Xilinx tool. This includes the software version number, build number and date, and copyright information.

### Arguments

**-short** - (Optional) Returns the version number of the software only.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

### Examples

The following example returns only the version number for the software:

```
version -short
```

## wait\_on\_run

Block execution of further Tcl commands until the specified run completes.

### Syntax

```
wait_on_run [-timeout arg] [-quiet] [-verbose] run
```

### Returns

Nothing

### Usage

Name	Description
<i>[-timeout]</i>	Maximum time to wait for the run to complete (in minutes) Default: -1
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>run</i>	Run to wait on

### Categories

[Project](#)

### Description

Blocks the execution of Tcl commands until the specified run completes, or until the specified amount of time has elapsed.

**Note** This command is used for running the tool in batch mode or from Tcl scripts. It is ignored when running interactively from the GUI.

### Arguments

**-timeout** *arg* - The time in minutes that the **wait\_on\_run** command should wait until the run finishes. This allows you to define a period of time beyond which the PlanAhead tool should resume executing Tcl commands even if the specified run has not finished execution. The default value of -1 is used if timeout is not specified, meaning that there is no limit to the amount of time the PlanAhead tool will wait for the run to complete.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*run* - The name of the run to wait on. If the specified run is not actively running when the **wait\_on\_run** command is used, you will get an error.

## Examples

The following example launches the impl\_1 run, and then waits for the specified run to complete, or to wait for one hour, whichever occurs first:

```
launch_runs impl_1  
wait_on_run -timeout 60 impl_1
```

## See Also

[launch\\_runs](#)

## write\_chipscope\_cdc

Export nets that are connected to debug ports.

### Syntax

```
write_chipscope_cdc [-quiet] [-verbose] file
```

### Returns

Name of the output file

### Usage

Name	Description
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>file</i>	ChipScope CDC file name

### Categories

[FileIO](#), [ChipScope](#)

### Description

Writes a ChipScope Definition and Connection (CDC) file containing the debug cores, ports, and signals defined in the current project.

ChipScope debug cores are added to a project through the use of the **create\_debug\_core** command. The CDC file stores information about source files, destination files, core parameters, and core settings for the ChipScope Pro Analyzer.

You can import this CDC file into the ChipScope Analyzer to automatically set up the net names on the ILA core data and trigger ports. The written CDC file can also be used as input to other projects by using the **read\_chipscope\_cdc** command.

### Arguments

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*file* - The CDC file name to be written.

**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.

## Examples

The following example writes a CDC file called bft.cdc:

```
write_chipscope_cdc bft.cdc
```

The written CDC file will include signals to be debugged by ChipScope as well as the clock domain for the signals, and other settings appropriate for use in ChipScope.

## See Also

- [create\\_debug\\_core](#)
- [read\\_chipscope\\_cdc](#)

## write\_csv

Export package pin and port placement information.

### Syntax

```
write_csv [-force] [-quiet] [-verbose] file
```

### Returns

Name of the output file

### Usage

Name	Description
<i>[-force]</i>	Overwrite existing file
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>file</i>	Pin Planning CSV file

### Categories

[FileIO](#)

### Description

Writes package pin and port placement information into a comma separated value (CSV) file.

The specific format and requirements of the CSV file are described in the *PlanAhead Users Guide* (ug632.pdf).

### Arguments

**-force** - Overwrite the CSV file if it already exists.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*file* - The filename of the CSV file to write.

**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.

### Examples

The following example exports a CSV file from the current project:

```
write_csv C:/Data/pinList.csv
```

## See Also

[read\\_csv](#)

## write\_edif

Export the current netlist as an EDIF file.

### Syntax

```
write_edif [-pblocks args] [-cell arg] [-force]
[-security_mode arg] [-quiet] [-verbose] file
```

### Returns

The name of the output file or directory

### Usage

Name	Description
<i>[-pblocks]</i>	Export netlist for these pblocks (not valid with -cell)
<i>[-cell]</i>	Export netlist for this cell (not valid with -pblocks)
<i>[-force]</i>	Overwrite existing file
<i>[-security_mode]</i>	If set to 'all', and some of design needs encryption then whole of design will be written to a single encrypted file Default: multifile
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>file</i>	Output file (directory with -pblocks or -cell)

### Categories

[FileIO](#)

### Description

Writes the current netlist as an EDIF file, or outputs the contents of specific Pblocks or hierarchical cells as EDIF netlist files.

In the case of either the -pblock or -cell argument being used, this argument specifies a directory name where the EDIF netlist files for each Pblock or cell will be written. The EDIF netlist file will be named after the Pblock or cell. If the directory specified does not exist, the PlanAhead tool will return an error.

### Arguments

**-pblocks *arg*** - Instructs the tool to output the contents of the specified Pblocks as EDIF netlist files. The contents of each Pblock will be written to a separate EDIF file.

**-cell *arg*** - Instructs the tool to output the contents of the specified hierarchical cell as EDIF netlist files. Only one cell can be specified for output.

**Note** The -pblock and -cell arguments are mutually exclusive. Although they are optional arguments, only one may be specified at one time.

**-force** - Overwrite the EDIF file if it already exists.

**-security\_mode [ multifile | all ]** - Write a multiple EDIF files when encrypted IP is found in the design, or write a single file.

- **multifile** - This is the default setting. By default the command writes out the full design netlist to the specified file. However, if the design contains secured IP, it creates an encrypted file containing the contents of the secured module. This will result in the output of multiple EDIF files, containing secured and unsecured elements of the design.
- **all** - Write both encrypted and unencrypted cells to a single specified file.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*file* - The filename of the EDIF file to write. The default file extension for an EDIF netlist is .edn.

**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.

## Examples

The following example writes an EDIF netlist file for the whole design to the specified file name:

```
write_edif C:/Data/edifOut.edn
```

The following example outputs an EDIF netlist for all Pblocks in the design. The files will be written to the specified directory.

```
write_edif -pblocks [get_pblocks] C:/Data/FPGA_Design/
```

## See Also

[read\\_edif](#)

## write\_ibis

Write IBIS models for current floorplan.

### Syntax

```
write_ibis [-allmodels] [-nopin] [-truncate arg] [-ibs arg]
[-pkg arg] [-quiet] [-verbose] file
```

### Returns

Name of the output file

### Usage

Name	Description
<i>[-allmodels]</i>	Include all available buffer models for this architecture. By default, only buffer models used by the floorplan are included.

### Categories

[FileIO](#)

### Description

Writes the IBIS models for the target device in the current design. The netlist and implementation details from the design are combined with the per-pin parasitic package information to create a custom IBIS model for the design.

Because the write\_ibis command incorporates design information into the IBIS Model, you must have an RTL, Netlist, or Implemented Design open when running this command.

### Arguments

**-allmodels** - Export all buffer models for the target device. By default the tool will only write buffer models used by the design.

**-nopin** - Disable per-pin modeling of the path from the die pad to the package pin. The IBIS model will include a single RLC transmission line model representation for all pins in the [Package] section. By default the file will include per-pin modeling of the package as RLC matrices in the [Define Package Model] section if this data is available.

**-truncate *arg*** - The maximum length for a signal name in the output file. Names longer than this will be truncated. Valid values are 20, 40, or 0 (unlimited). By default the signal names are truncated to 40 characters in accordance with the IBIS version 4.2 specification.

**-ibs *arg*** - Specify an updated generic IBIS models file. This is used to override the IBIS models found in the tool installation under the parts directory. This argument is required for any parts that do not have generic models in the installation directory.

**-pkg *arg*** - Specify an updated per pin parasitic package data file. This is used to override the parasitic package file found in the the tool installation hierarchy under the parts directory. This argument is required for any parts that do not have generic models in the installation directory.

*file* - The filename of the IBIS file to write.

**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.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

## Examples

The following example exports all buffer models for the target device, eliminates truncation of signal names, and specifies the file name and path to write:

```
write_ibis -allmodels -truncate 0 C:/Data/FPGA_Design/ibisOut.txt
```

## write\_timing

Export a set of timing results to file.

### Syntax

```
write_timing [-force] [-quiet] [-verbose] name file
```

### Returns

Nothing

### Usage

Name	Description
<i>[-force]</i>	Overwrite existing file
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>name</i>	Name for the set of results
<i>file</i>	Name of the file to write the results to

### Categories

[FileIO](#)

### Description

Write the results of timing analysis to the specified file. This command writes the results from timing analysis previously created by the **report\_timing** command; it does not actually run timing analysis.

**write\_timing** produces a legacy timing path report. The format of this file is different from the output of the **report\_timing -file** command.

### Arguments

**-force** - Overwrite the timing file if it already exists.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*name* - The name of the results set created by the **report\_timing** command. These are the timing results that will be output to the specified file.

*file* - The filename of the timing file to write.

**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.

## Examples

The following example runs timing analysis and creates a timing results set called `time_1`, then writes the named timing results to the specified file:

```
report_timing -name time_1  
write_timing time_1 C:/Data/FPGA_Design/bft_time_1.txt
```

## See Also

[report\\_timing](#)

## write\_ucf

Export UCF information to a file or directory.

### Syntax

```
write_ucf [-no_fixed_only] [-constraints arg] [-pblocks args]
[-cell arg] [-quiet] [-verbose] file
```

### Returns

Name of the output file or directory

### Usage

Name	Description
<i>[-no_fixed_only]</i>	Export fixed and non-fixed placement (by default only fixed placement will be exported)
<i>[-constraints]</i>	Include constraints that are flagged invalid Values: valid, invalid, all Default: valid
<i>[-pblocks]</i>	Export placement for these pblocks (not valid with -cell)
<i>[-cell]</i>	Export placement for this cell (not valid with -pblocks)
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>file</i>	Output file (directory with -pblocks, -cell)

### Categories

FileIO

### Description

Writes physical constraints to a user constraint file (UCF). Constraints can be written from the top-level, which is the default, from specific Pblocks, or from a specific hierarchical cell.

This command will write the constraints from all UCF files of the active constraint fileset. Constraints from multiple files will be included in the specified UCF file.

### Arguments

**-no\_fixed\_only** - Write both fixed and unfixed placement LOCs to the constraint file being written. By default only the fixed LOCs will be written to the UCF file. Fixed LOCs are associated with user-assigned placements, while unfixed LOCs are associated with tool assigned placements.

**-constraints *arg*** - Write constraints that are flagged valid, invalid, or all constraints (both valid and invalid). The default behavior is to export only valid constraints to the UCF file. However, you can specify **-constraints** values of VALID, INVALID, or ALL.

**-pblocks *arg*** - One or more Pblocks from which to write the constraints.

**-cell** *arg* - The name of a hierarchical cell in the current design to export the constraints from. The constraints will be written to the specified UCF file relative to the specified cell.

**Note** A design must be open when using this option.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*file* - The filename of the UCF file to write.

**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.

**Note** When you use **-pblock** or **-cell**, this argument specifies a directory name where the UCF files for each Pblock or cell will be written. Each UCF file will be named after a Pblock or cell. If the specified directory does not exist you will get an error.

## Examples

The following example writes the valid and invalid constraints, including both fixed and unfixed LOCs, for all Pblocks found in the design to the specified directory:

```
write_ucf -no_fixed_only -constraints all -pblocks [get_pblocks] C:/Data/FPGA_Design
```

## See Also

- [read\\_ucf](#)
- [save\\_design](#)
- [save\\_design\\_as](#)

## write\_verilog

Export the current netlist in Verilog format.

### Syntax

```
write_verilog [-cell arg] [-mode arg] [-lib]
[-port_diff_buffers] [-write_all_overrides] [-rename_top arg]
[-sdf_anno arg] [-sdf_file arg] [-force] [-include_xilinx_libs]
[-quiet] [-verbose] file
```

### Returns

The name of the output file or directory

### Usage

Name	Description
<i>[-cell]</i>	Root of the design to write. eg. des.subblk.cpu Default: whole design
<i>[-mode]</i>	Values: design, port, sta, funcsim, timesim Default: design
<i>[-lib]</i>	Write each library into a separate file
<i>[-port_diff_buffers]</i>	Output differential buffers when writing in -port mode
<i>[-write_all_overrides]</i>	Write parameter overrides on Xilinx primitives even if the override value is the same as the default value
<i>[-rename_top]</i>	Replace top module name with custom name e.g. netlist Default: new top module name
<i>[-sdf_anno]</i>	Specify if sdf_annotate system task statement is generated
<i>[-sdf_file]</i>	Full path to sdf file location Default: file .sdf
<i>[-force]</i>	Overwrite existing file
<i>[-include_xilinx_libs]</i>	Include simulation models directly in netlist instead of linking to library
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>file</i>	Which file to write

### Categories

[FileIO](#), [Simulation](#)

### Description

Write a Verilog netlist of the current design or from a specific cell of the design to the specified file or directory. The output is a IEEE 1364-2001 compliant Verilog HDL file that contains netlist information obtained from the input design files.

You can output a complete netlist of the design or specific cell, or output a port list for the design, or a Verilog netlist for simulation or static timing analysis.

## Arguments

**-cell *arg*** - Write the Verilog netlist from a specified cell or block level of the design hierarchy. The output Verilog file or files will only include information contained within the specified cell or module.

**-mode *arg*** - The mode to use when writing the Verilog file. By default, the Verilog netlist is written for the whole design. Valid mode values are:

- **design** - Output a Verilog netlist for the whole design. This acts as a snapshot of the design, including all post placement, implementation, and routing information in the netlist.
- **port** - Outputs only the I/O ports for the top-level of the design.
- **sta** - Output a Verilog netlist to be used for static timing analysis (STA).
- **funcsim** - Output a Verilog netlist to be used for functional simulation. The output netlist is not suitable for synthesis.
- **timesim** - Output a Verilog netlist to be used for timing simulation. The output netlist is not suitable for synthesis.

**-lib** - Create a separate Verilog file for each library used by the design.

**Note** This option is the opposite of, and replaces the **-nolib** option from prior releases. Previously the default behavior of **write\_verilog** was to output a separate Verilog file for each library used in the design, unless **-nolib** was specified. Now you must specify the **-lib** option to output separate Verilog files for each library

**-port\_diff\_buffers** - Add the differential pair buffers and internal wires associated with those buffers into the output ports list. This argument is only valid when **-mode port** is specified.

**-write\_all\_overrides [ true | false ]** - Write parameter overrides, in the design to the Verilog output even if the value of the parameter is the same as the defined primitive default value. If the option is false then parameter values which are equivalent to the primitive defaults are not output to the Verilog file. Setting this option to true will not change the result but makes the output Verilog more verbose.

**-rename\_top *arg*** - Rename the top module in the output as specified. This option only works with **-mode funcsim** or **-mode timesim** to allow the Verilog netlist to plug into top-level simulation test benches.

**-sdf\_anno [ true | false ]** - Add the **\$sdf\_annotate** statement to the Verilog netlist. Valid values are true (or 1) and false (or 0). This option only works with **-mode sim**, and is set to false by default.

**-sdf\_file *arg*** - The path and filename of the SDF file to use when writing the **\$sdf\_annotate** statement into the output Verilog file. When not specified, the SDF file is assumed to have the same name and path as the Verilog output specified by *<file>*, with a file extension of .sdf. The SDF file must be separately written to the specified file path and name using the **write\_sdf** command.

**-force** - Overwrite the Verilog files if they already exists.

**-include\_xilinx\_libs** - This option writes the simulation models directly in the output netlist file rather than pointing to the libraries by reference.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*file* - The filename of the Verilog file to write. If the file name does not have either a `.v` or `.ver` file extension then the name is assumed to refer to a directory, and the Verilog file is named after the top module, and is output to the specified directory.

**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.

## Examples

The following example writes a Verilog simulation netlist file for the whole design to the specified file and path:

```
write_verilog C:/Data/my_verilog.v
```

In the following example, the specified file does not have a `.v` or `.ver` file extension, and so is treated as a directory name. A Verilog file is created in the specified directory and is named after the top module in the design. In addition, because the **-mode timesim** and **-sdf\_anno** options are specified, the **\$sdf\_annotate** statement will be added to the Verilog netlist. However, since the **-sdf\_file** option is not specified, the SDF file is assumed to have the same name as the Verilog output file, with an `.sdf` file extension:

```
write_verilog C:/Data/my_verilog.net -mode timesim -sdf_anno true
```

**Note** The Verilog file name in this example will be `top.v` and will be written to the `my_verilog.net` directory. The SDF file written to the **\$sdf\_annotate** statement is assumed to be in the same directory and named `top.sdf`

## See Also

- [write\\_sdf](#)
- [write\\_vhdl](#)

## write\_vhdl

Export the current netlist in VHDL format.

### Syntax

```
write_vhdl [-cell arg] [-mode arg] [-lib] [-port_diff_buffers]
[-write_all_overrides] [-rename_top arg] [-arch_only] [-force]
[-include_xilinx_libs] [-quiet] [-verbose] file
```

### Returns

The name of the output file or directory

### Usage

Name	Description
<code>[-cell]</code>	Root of the design to write. eg. des.subblk.cpu Default: whole design
<code>[-mode]</code>	Output mode. Valid values: funcsim, port Default: funcsim
<code>[-lib]</code>	Write each library into a separate file
<code>[-port_diff_buffers]</code>	Output differential buffers when writing in -port mode
<code>[-write_all_overrides]</code>	Write parameter overrides on Xilinx primitives even if the same as the default value
<code>[-rename_top]</code>	Replace top module name with custom name e.g. netlist Default: new top module name
<code>[-arch_only]</code>	Write only the architecture, not the entity declaration for the top cell
<code>[-force]</code>	Overwrite existing file
<code>[-include_xilinx_libs]</code>	Include simulation models directly in netlist instead of linking to library
<code>[-quiet]</code>	Ignore command errors
<code>[-verbose]</code>	Suspend message limits during command execution
<code>file</code>	Which file to write

### Categories

[FileIO](#), [Simulation](#)

### Description

Write a VHDL netlist of the current design or from a specific cell of the design to the specified file or directory.

The output of this command is a VHDL IEEE 1076.4 VITAL-2000 compliant VHDL file that contains netlist information obtained from the input design files. You can output a complete netlist of the design or specific cell, or output a port list for the design.

## Arguments

**-cell** *arg* - Write the VHDL netlist from a specified cell or block level of the design hierarchy. The output VHDL file or files will only include information contained within the specified cell or module.

**-mode** *arg* - The mode to use when writing the VHDL file. By default, the simulation netlist is written for the whole design. Valid mode values are:

- **funcsim** - Output the VHDL netlist to be used as a functional simulation model. The output netlist is not suitable for synthesis. This is the default setting.
- **port** - Outputs only the I/O ports in the entity declaration for the top module.

**-lib** - Create a separate VHDL file for each library used by the design.

**Note** This option is the opposite of, and replaces the **-nolib** option from prior releases. Previously the default behavior of **write\_vhdl** was to output a separate VHDL file for each library used in the design, unless **-nolib** was specified. Now you must specify the **-lib** option to output separate files for each library

**-port\_diff\_buffers** - Add the differential pair buffers and internal wires associated with those buffers into the output ports list. This argument is only valid when **-mode port** is specified.

**-write\_all\_overrides** [ true | false ] - Write parameter overrides in the design to the VHDL output even if the value of the parameter is the same as the defined primitive default value. If the option is false then parameter values which are equivalent to the primitive defaults are not output to the VHDL file. Setting this option to true will not change the result but makes the output netlist more verbose.

**-rename\_top** *arg* - Rename the top module in the output as specified. This option only works with **-mode funcsim** to allow the VHDL netlist to plug into top-level simulation test benches.

**-arch\_only** - Suppress the entity definition of the top module, and outputs the architecture only. This simplifies the use of the output VHDL netlist with a separate test bench.

**-include\_xilinx\_libs** - Write the simulation models directly in the output netlist file rather than pointing to the libraries by reference.

**-force** - Overwrite the VHDL files if they already exists.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*file* - The filename of the VHDL file to write. If the file name does not have either a .vhd or .vhd1 file extension then the name is assumed to be a directory, and the VHDL file is named after the top module, and is output to the specified directory.

**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.

## Examples

The following example writes a VHDL simulation netlist file for the whole design to the specified file and path:

```
write_vhdl C:/Data/bft_top.vhd
```

In the following example the entity definition of the top-level module is not output to the VHDL netlist:

```
write_vhdl C:/Data/vhdl_arch_only.vhd -arch_only
```

## See Also

[write\\_verilog](#)

## write\_xdc

Specifies the name of the XDC file to be created. The default file extension for a XDC file is .xdc. If you called this command with -pblocks or -cell The file argument will specify the output directory.

### Syntax

```
write_xdc [-no_fixed_only] [-constraints arg] [-pblocks args]
[-cell arg] [-sdc] [-no_tool_comments] [-force] [-quiet]
[-verbose] file
```

### Returns

Name of the output file or directory

### Usage

Name	Description
<i>[-no_fixed_only]</i>	Export fixed and non-fixed placement (by default only fixed placement will be exported)
<i>[-constraints]</i>	Include constraints that are flagged invalid Values: valid, invalid, all Default: valid
<i>[-pblocks]</i>	Export placement for these pblocks (not valid with -cell)
<i>[-cell]</i>	Export placement for this cell (not valid with -pblocks)
<i>[-sdc]</i>	Export all timing constraints
<i>[-no_tool_comments]</i>	Don't write verbose tool generated comments to the xdc when translating from ucf
<i>[-force]</i>	Overwrite existing file
<i>[-quiet]</i>	Ignore command errors
<i>[-verbose]</i>	Suspend message limits during command execution
<i>file</i>	Output file/directory path. (path is a directory if -pblocks or -cell used)

### Categories

[FileIO](#)

### Description

Writes constraints to a Xilinx Design Constraints file (XDC). The XDC can be exported from the top-level, which is the default, from specific Pblocks, or from a specific hierarchical cell.

This command can be used to create an XDC file from a design with UCF files. All constraints from the active constraint fileset will be exported to the XDC, even if they come from multiple files.

## Arguments

**-no\_fixed\_only** - Export both fixed and unfixed placement LOCs to the constraint file being written. By default only the fixed LOCs will be written to the XDC file. Fixed LOCs are associated with user-assigned placements, while unfixed LOCs are associated with tool assigned placements.

**-constraints *arg*** - Export constraints that are flagged valid, invalid, or all constraints (both valid and invalid). The default behavior is to export only valid constraints to the XDC file. Valid values are VALID, INVALID, or ALL.

**-pblocks *arg*** - One or more Pblocks to export the constraints from.

**-cell *arg*** - The name of a hierarchical cell in the current design to export the constraints from. The constraints will be written to the specified XDC file relative to the specified cell.

**Note** A design must be open when using this option.

**-sdc** - Export only the timing constraints in an SDC format from the current design. Does not export any other defined constraints.

**-no\_tool\_comments** - Do not generate tool comments when writing the XDC file. Without this argument, comments and warnings related to the creation of the XDC file will be added.

**-force** - Overwrite the XDC file if it already exists.

**-quiet** - (Optional) Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns TCL\_OK regardless of any errors encountered during execution.

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

**Note** Message limits can be defined with the **set\_msg\_limit** command.

*file* - The filename of the XDC file to write.

**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.

**Note** When you use **-pblock** or **-cell**, this argument specifies a directory name where the XDC files for each Pblock or cell will be written. Each XDC file will be named after a Pblock or cell. If the specified directory does not exist you will get an error.

## Examples

The following example writes the valid and invalid constraints, including both fixed and unfixed LOCs, for all Pblocks found in the design to the specified directory:

```
write_xdc -no_fixed_only -constraints all -pblocks [get_pblocks] C:/Data/FPGA_Design
```

## See Also

[read\\_xdc](#)

## Additional Resources

---

The [PlanAhead User Guide \(UG632\)](#) provides detailed information on usage of the PlanAhead tool. The [Hierarchical Design Methodology Guide \(UG748\)](#) provides information on working with hierarchical designs for Xilinx FPGA devices.

Other Xilinx resources on the Web include:

- Xilinx Glossary - <http://www.xilinx.com/company/terms.htm>
- Xilinx Support and Documentation - <http://www.xilinx.com/support>

### Tcl Developer Xchange

Tcl reference material is available on the Internet. Xilinx recommends the Tcl Developer Xchange, which maintains the open source code base for Tcl, and is located at:

<http://www.tcl.tk>

An introductory tutorial is available at:

<http://www.tcl.tk/man/tcl/tutorial/tcltutorial.html>

### About SDC

Synopsys Design Constraints (SDC) is an accepted industry standard for communicating design intent to tools, particularly for timing analysis. A reference copy of the SDC specification is available from Synopsys by registering for the TAP-in program at:

<http://www.synopsys.com/Community/Interoperability/Pages/TapinSDC.aspx>