

Tcl's Syntax Parsing Overview

Command and Word Separation

sequences of **white space** (space or horizontal tab) separate words

line endings and semicolons (;) separate commands

Variable and Return Value Substitution

variable content (aka. its “value”) gets inserted for a dollar (\$) followed by a valid variable name

return value of a command gets inserted for parts of a command line enclosed in square brackets

Command Execution

H
O
N
O
R

Q
U
O
T
I
N
G

A backslash (\) at the very end of a line is special, as it concatenates a line with the next one, **removing all leading white space**

Backslash-Substitutions – to insert non-graphic characters (e.g. \n, \t, \a ... - like in C/C++)

Backslash-Quoting – to remove a special meaning from the directly following character (e.g. \; denotes a literal semicolon, not a *Command Separator*)

Quoting with Double Quotes (" ... ") – to remove the special meaning from *Separators* and *Curly Brace Quoting*, while still allowing *Variable and Command Substitution*.

Quoting with Curly Braces ({ ... }) – to remove the special meaning from Everything enclosed.

For a full description in manual style see XILINX documentation ug835.

Applies **recursively** for nested command substitution, i.e. for **square brackets inside square brackets**.

From the syntax viewpoint only the **first word** is relevant – all the following words are evaluated by the command itself!

array name must be followed by index in parentheses to access an element

unusual (“funny”) names may need to be enclosed in curly braces

Read The Fine Manual

Similar for commands string, clock, file ...

list-related but creates proper list from its arguments

commonality: first letter ell

as a “matter of taste and style” and for economy because of memory limitations in the early 1990s, when development of Tcl started

	Tcl (native)		Vivado
	Scarce – some hints in error messages when command execution fails ... so: RTFM	Syntax Help	All commands have a -help option, providing extensive explanations (close to the full manual page)
	Rather weak – stronger for (historically) newer commands compared to the “good ol’ & ancient” commands	Uniformity	very strong (especially in comparison native Tcl commands – which need to be used nevertheless)
	Command Collections used for grouping, e.g. <ul style="list-style-type: none">• array set ... (get all elements of an array)• array get ... (get all elements of an array)• array size ... (get number of array elements)• array names ... (get list of array indices)• ...	Related Commands	Similar commands lexically grouped by common prefix, e.g.: <ul style="list-style-type: none">• get_xyz (access xyz element of design model)• set_xyz (modify xyz element of design model)• report_xyz (generate xyz report)• ...
	Prominent exception: commands for <i>list</i> -processing, e.g.: <ul style="list-style-type: none">• list• ...• llength ...• lindex ...• lsort ...		Similar command typically grouped under the same name, with an option to change their behaviour, e.g.: <ul style="list-style-type: none">• create_file_set ...• create_fileset -constrset ...• create_fileset -simset ...
	<i>Minimalistic</i> approach (for economy), i.e.: <ul style="list-style-type: none">• most commands with a “result” produce it as <i>return value</i>• so, to “print” it to the console use: puts [...]• and to save it in a file open a channel<ul style="list-style-type: none">• store return value of open: set chan [open ...]• use it as channel argument: puts \$chan [...]	Other (General)	<i>Regular</i> approach (for convenience), e.g. all report_xyz commands <ul style="list-style-type: none">• by default produce their output on the console (window)• have a -file option to store the report in a <i>file</i>• have a -return_string option to deliver the report via their return value

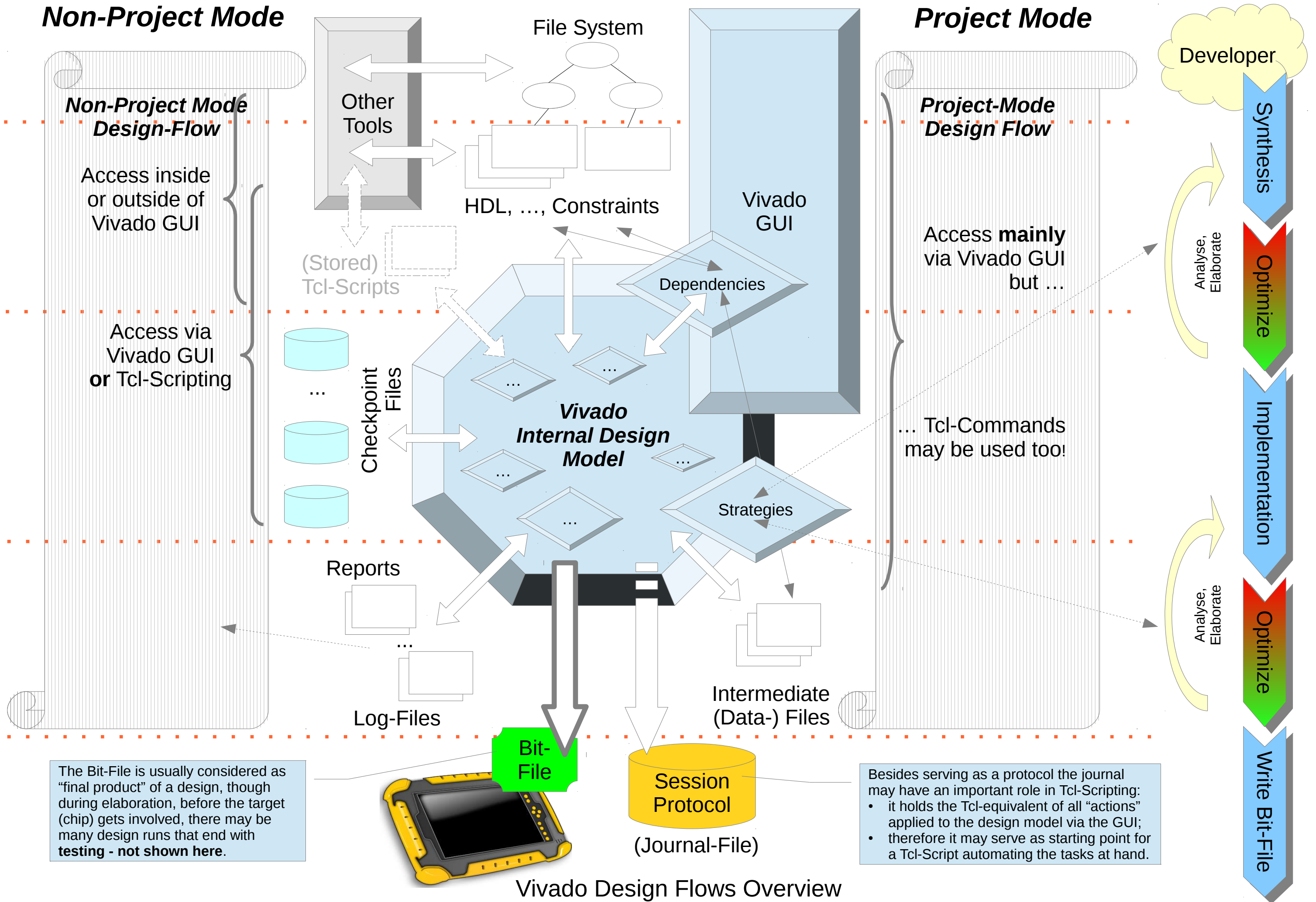
Vivado vs. Tcl Syntax

(cc) BY-SA: Technische Beratung für EDV, Dipl.-Ing. Martin Weitzel, Germany, <http://tbfe.de>

largely bloats command look-up table (but not any more an issue considering CPU power and main memory size available today)

Non-Project Mode

Project Mode



“Non-Project Mode”
is sometimes
confused with using
the Vivado Tcl-
Commands only.

“Project Mode” is
sometimes confused
with using the Vivado
GUI only.

It is well
possible to
mix Vivado
Tcl-Commands
and using the
GUI.

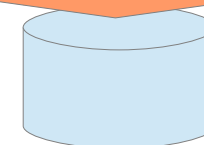
It is well
possible to mix
using the
Vivado GUI
and Tcl-
Commands.

Use the Vivado
GUI for any non-
repetitive or mostly
explorative
elaboration and as
a convenient way
to view reports.

Use Vivado Tcl-
Commands to
automate anything
that is systematic
and repetitive.

Non-Project Mode	Activity	Project Mode
Explicitely get HDL-Sources into Internal Model <ul style="list-style-type: none">• read_vhdl• read_verilog• read_xdc	Preparing the Design Model	Manage file dependencies automatically via File Sets: <ul style="list-style-type: none">• create_project• add_files
		• current_fileset
		• import_...
		<ul style="list-style-type: none">• open_project• current_project• close_project
Elaborating the Design		
There are too many command to list exhaustively, so just some typical examples are given: <ul style="list-style-type: none">• get_cells (select and query design objects)• set_property (modify design objects)• report_... (generate various kinds of reports and summaries)• ...		
Run synthesis tools (various options allow to specify details) <ul style="list-style-type: none">• synth_design	Synthesis	Select and run tools depending on changes made (various options, including some for load-sharing on compile servers): <ul style="list-style-type: none">• launch_runs• wait_on_run
Run implementation tools (rich on options, especially for optimization) <ul style="list-style-type: none">• opt_design• place_design• phys_opt_design• route_design	Implementation	
Generating Target File		
<ul style="list-style-type: none">• write_bitstream (various options to select different formats and encoding)• write_verilog• write_vhdl• write_xdc		

Creating the **Bitstream File** is typically the overall goal when using Vivado.

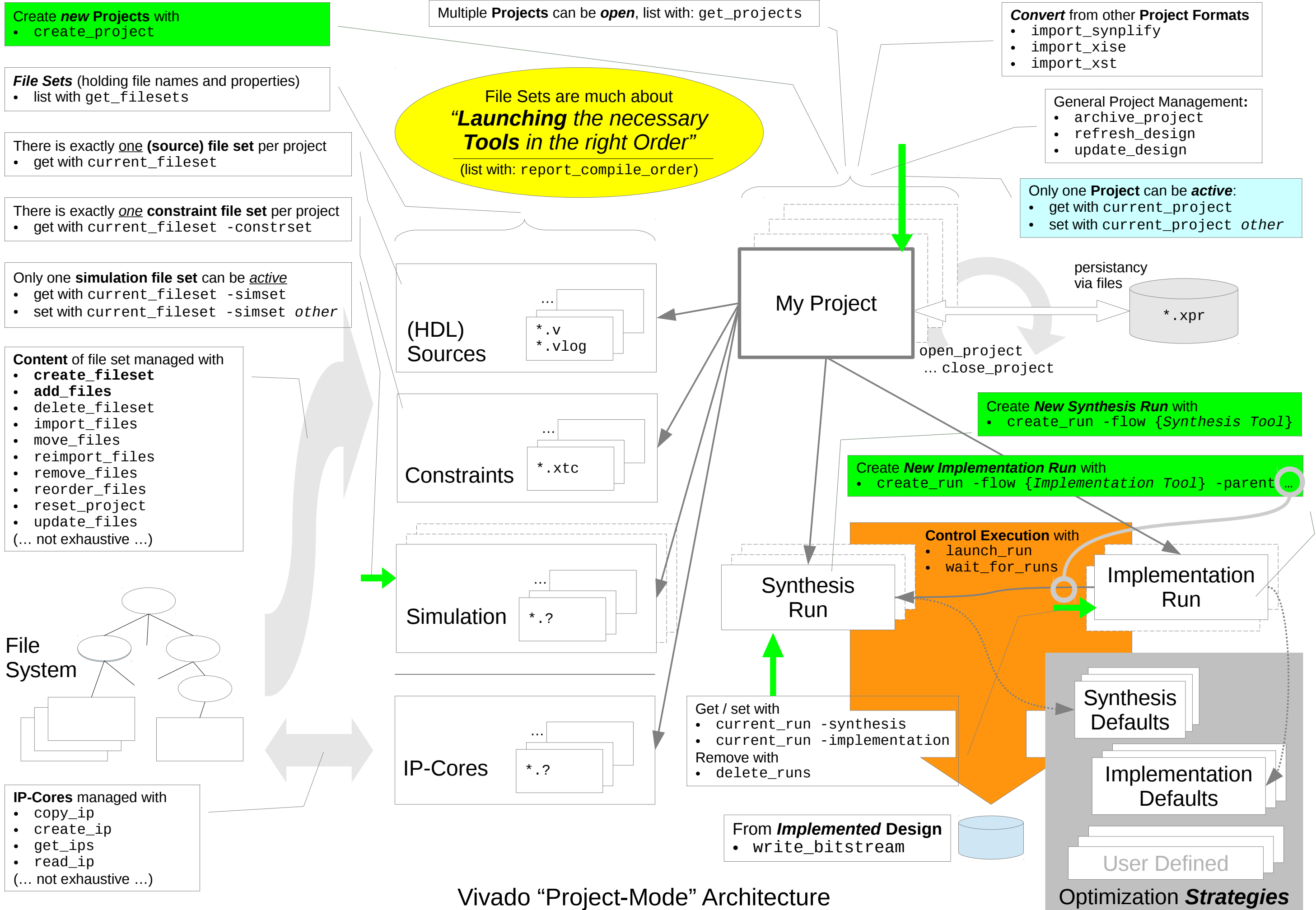


Vivado Modes and Tcl Commands

(cc) BY-SA: Technische Beratung für EDV, Dipl.-Ing. Martin Weitzel, Germany, <http://tbfe.de>

Usually these phases overlap and are repeated, depending on reports and other kinds of checks and verifications, including simulation (not shown here). If files – especially Verilog or VHDL – are modified, they need to be read again explicitly.

Usually these phases overlap and are repeated, depending on reports and other kinds of checks and verifications. If files Verilog or VHDL Files are modified, Vivado knows about dependencies and will launch the tools as required in the right order.



Vivado “Project-Mode” Architecture

(cc) BY-SA: Technische Beratung für EDV, Dipl.-Ing. Martin Weitzel, Germany, <http://tbfe.de>