Enhance Tempus/Innovus Experience with Tcl

Debugging and Exploration made easy !

July - 2018

Course Objectives

This course will help users to learn how to:

1. Create program flow control in Tempus / Innovus script
2. Browse Tempus data model through TCL
3. Dynamically create timing constraints or commands
4. Create repository of your own commands

Modules

1. TCL overview
   - ❑ Basic structure
   - ❑ Built-in commands
   - ❑ Control flow
2. Tempus/Innovus timing data model
   - ❑ Collection Data types
   - ❑ Commands to manipulate collection data types
3. Access Tempus/Innovus data-model through TCL
   - ❑ Browse object properties
4. Create your commands in Tempus/Innovus
   - ❑ Writing Tcl procedures
   - ❑ Adding Help message
   - ❑ Argument validation

Module 1 - Tcl Overview

Why bother TCL in Tempus / Innovus?

- ❑ Tempus and Innovus command interface (shell) speaks, listens and understands TCL
  - ❑ The nature of command syntax (including Tempus commands)
  - ❑ How variables are created and used later
  - ❑ How expressions are interpreted for evaluation
  - ❑ How scripts, control flow and procedures work
- ❑ TCL provides the necessary programing constructs for scripting in Tempus and Innovus
- ❑ TCL helps code designer context to get more specific results from Tempus and Innovus commands

NOTE: Tempus output used as example references. However this applies equally for Innovus.

How to run TCL code in Tempus

Type in TCL commands interactively in Tempus shell

```
tempus prompt > puts "Hi There!"
```

Save TCL commands in a file and source it in Tempus shell or nest inside another

```
tempus prompt > source design_info.tcl
```

Pass TCL commands in script in batch mode or as direct TCL commands

```
% tempus -files "globals.tcl design_init.tcl" \
   -execute "set TOP_MODULE soc_top; set SI_ANALYSIS true;
   set CPU 16; set TOOL_VERSION 17.10.000"
```

Nested sourcing of TCL scripts through another parent TCL script

```
report_timing -late -max_paths 100
report_timing -early -max_paths 100
```

```
read_lib slow.lb
read_verilog top.v
set_top_module top
update_timing -full
source custom_reports.tcl
```

TCL Command Syntax

- **Command** : Space separated one or more words, 1st word holds command name

    command_name arg1 arg2 ...

- **Comment :** Lines starting with #
- **Script :** Sequence of commands saved in a file, separated by new-lines or semi-colons

```
# This is a comment
cmd1 arg arg arg ...
cmd2 arg arg arg ...  # This is a bad  comment - Error
cmd3 arg arg arg ...; # This is a good comment

# Even a comment can be split across multiple \
lines!
```

TCL Variables

- A name composed any characters such as letters, digits, underscores

Command	Description
`set <varname> ?<value>?`	Creates a variable and assigns / retrieves value
`unset <varname>`	Deletes one or more variables
`info exist <varname>`	Checks pre-existence of variable

- Variables store **STRING** values or **Arbitrary** length

```
tempus 1> set a "Hello world!"
Hello world!
tempus 2> set a
Hello world!
tempus 3> set b
can’t read "b": no such variable
tempus 4> info exist a
1
tempus 5> unset a b
```

Variable Substitution (`$varName / ${varName}`)

- Each occurrence of `$varName / ${varName}` replaced with the corresponding variable value
  - Except where `$` is escaped (`\$`)
  - **Wildcard support :** Asterisk (*) and question mark (?) for pattern matching on objects such as variables and strings

    Asterisk (*) – match any sequence of characters in an object names
    Question mark (?) – match any single character in object names

```
tempus 1 > set d 10; set p 2
2
tempus 2> puts "Time = $d ns"
Time = 10 ns
tempus 3> get_ports scan_*
scan_in scan_enable scan_out
```

Backslash Substitution and Quoting

- Inserts special characters such as new line, tabs, into text
  - \n : inserts new line character
  - \t : inserts tab character

```
tempus 1 > puts “WNS: -0.3ns\nTNS: -10.4ns”
WNS: -0.3ns
TNS: -10.4ns
```

Command Substitution (`[]`)

- Each occurrence of `[<command>]` is replaced with the value returned from the *last* command executed in `<commands>`
  - Except where `[]` are escaped (`\[` and `\]`)
  - Nested command substitution allowed
  - Command name and arguments case sensitive (clk and CLK are different names)

Sample command	**Result**
`set b 8`	`8`
`set a [expr $b+2]`	`10`
`puts “Delay: [set a] ns”`	`Delay: 10 ns`

How to Output data on screen

- ❑ `Puts` **`?-nonewline?`** `String (Tempus Command)`
- ❑ `puts ?`**`-nonewline`**`? ?`*`file_id`*`?` *`String (TCL command)`*
  - `-nonewline` suppresses output of new-line character
  - `Puts` captures data on screen as well as tempus log file
  - *`file_id`* indicates file ID of the channel to which to send output

Sample command

```
set clock "Clock_20MHZ"
set latency "2ns"
puts -nonewline "Clock $clock: "
puts "Latency : $latency"
```

Result

```
Clock_20MHZ
 2ns
 Clock Clock_20MHZ:
 Clock Clock_20MHZ: Latency : 2ns
```

Quoting

- ❑ Double-quotes “” do not disable command / backslash/ variable substitution (weak quote)
- ❑ Curly braces { } disables all substitution (rigid quote)

```
tempus 1 > set a 5; set b 10
10
tempus 2> puts {[expr $b - $a]}
{[expr $b - $a]}
tempus 3> puts [expr $b - $a]
5
```

Data Types

- Strings
  - Sequence of characters
  - Command arguments as well as return values treated as strings lists
  - Ordered set of elements – string / another list
- Arrays
  - Each element is a variable with name/value pair
- Collection
  - Tempus data type to group design elements such as ports, nets, instances, pins, clocks, timing paths, timing arcs, registers etc.

Common string commands

- Most string manipulations can be done by Tcl `string` command
  - Syntax : `string option ?arg … ?`

```
tempus 1 > set first “clock1”
clock1
tempus 2> string compare $first “clock2”
-1
tempus 3> string match $first “clock2”
0
tempus 4> string length $first
6
tempus 5> string equal $first “clock2”
0
tempus 6> string cat "clock1" "->" "clock2"
clock1->clock2
```

A few other Tcl commands to use with strings

- ❑ Type “`man command_name`” for details about these commands

Command	Description
`format`	Formats a string
`regexp`	Matches regular expression within a string
`regsub`	Substitutes sub-strings based on regular expression

```
set tool_version “17.22-s086”
set info_type “cell_summary”
set cell_name “I1/FF1/D”
set lib_cell_name “SDFFHQ1X”
set cell_info [format “%s %s“ $cell_name $lib_cell_name]
redirect [format “%s_%s.rpt” $tool_version $info_type] { puts $cell_info }
```

List

- Create list by enclosing members in curly braces { } / double quotes “” / with Tcl list command

```
set data_pin_list “I1/FF1/D I2/FF2/D I3/FF3/D”
set async_pin_list { I1/FF1/RST I2/FF2/CD I3/FF3/SET}
set clock_pin_list [list I1/FF1/CLK I2/FF2/CLK I3/FF3/CLK]
```

- List members can be accessed / manipulated through special list commands

```
tempus 1> lindex $data_pin_list 0
I1/FF1/D
tempus 2> llength $async_pin_list
3
tempus 3> set new_list [concat $data_pin_list $async_pin_list]
I1/FF1/D I2/FF2/D I3/FF3/D I1/FF1/RST I2/FF2/CD I3/FF3/SET

tempus 3> set file_hier [split "/home/a/b/c/d/test.sdc" /]
{} home a b c d test.sdc
```

A few other Tcl commands to use with Lists

- ❑ Type “`man command_name`” for details about these commands

Command	Description
`join`	Joins list elements into a string
`lappend`	Appends elements to a list
`lsearch`	Searches for a match in a list for a regular expression

Arrays

Create array element using

```
set array_name(element_name) “string/list”
```

```
set report_file(early) timing_report_early.txt
set report_file(late)  timing_report_late.txt
```

Reference array element using `$array_name(element_name)`

```
tempus 1> Puts $report_file(early)
timing_report_early.txt
```

Use Tcl `array` command to query about array elements

```
tempus 1> array size report_file
2
tempus 2> array names report_file
early late
```

TCL Expressions

- ❑ Expressions are formed of Tcl operators and operands
- ❑ Expression could be logical (true/false), arithmetic, relational (greater than, equal), bit-wise
- ❑ For string comparison and pattern matching expressions, using string commands
- ❑ Use `expr` command to evaluate Tcl expressions
- ❑ Control flow commands can also evaluate expressions without requiring `expr` command

```
tempus 1> set period 10.0
tempus 2> set half_period [expr $period / 2.0]
tempus 3> create_generated_clock -name DIV_CLK \
            -source [get_ports CLK] \
             -period $half_period -divide_by 2
tempus 4 >  expr {“clockA” eq “clocka”}
0
tempus 5 >  expr {“clockA” ne “clocka”}
1
```

Examples: Tcl Operators

```
Tempus 1> set a 2
2
Tempus 2> expr    $a > 0 && $a <= 3
1
Tempus 3> expr !(($a == 1) || ($a == 2))
0
Tempus 4> expr $a || 0
1
```

non-zero evaluates to true

Always use parentheses () to ensure readability.

```
Tempus 1> set a 0x07
0x07
Tempus 2> expr $a & 0x04
4
Tempus 3> set a [expr $a | 0x08]
15
Tempus 4> set a_neg [expr ~$a + 1]
-15
```

```
bits  7 6 5 4  3 2 1 0
      0000 0111 (7)
AND   0000 0100 (4)
      ---------
      0000 0100 (4)

      0000 0111 (7)
OR    0000 1000 (8)
      ---------
      0000 1111 (15)

INV   0000 1111 (15)
      ---------
      1111 0000 (-16)
    + 0000 0001 (1)
      ---------
      1111 0001 (-15)
```

Mathematical Functions

`expr { math_function(arg1, arg2, …) }`

`acos(x)`	`exp(x)`	`abs(x)`
`asin(x)`	`log(x)`	`ceil(x)`
`atan(x)`	`log10(x)`	`floor(x)`
`atan2(x,y)`	`pow(x,y)`	`round(x)`
`cos(x)`	`sqrt(x)`	
`cosh(x)`	`isqrt(x)`	`int(x)`
`sin(x)`	`hypot(x,y)`	`double(x)`
`sinh(x)`	`fmod(x,y)`	`bool()`
`tan(x)`	`max(x1,x2,..`	
`tanh(x)`	`min(x1, x2, )`	`srand(x)`
	`wide(x)`	`rand()`

TCL Control Flow

```
if {expression1} {
   script1
} elseif {expression2} {
  script2
} elseif {expression3} {
  script3
…
} else {
  final_script
}
```

Evaluate `expression*` in the order top to bottom

Stop at the first expression that evaluates to non-zero/true value

Execute the script enclosed with that expression

If all `expression*==0,` execute `final_script`

```
for {initialization_expr} \
    {loop_terminate_expr} \
    {update_expr} {
   Script
}
```

First `initialization_expr` initializes iteration variable

Execute `Script` provided `loop_terminate_expr > 0`

After `Script` is execute, update iteration variable and re-evaluate `loop_terminate_expr` that depends on its new value

Execute the `Script` until **`loop_terminate_expr`** ==0

```
foreach  var $any_list
{
  Script
}
```

Execute script for each element in specified Tcl list

Use `break` command to exit out of the loop

Use `continue` command to skip current iteration

Example - TCL Control Flow

```
if {[file exists a.lib.2]}{
   read_lib a.lib.2
} elseif {[file exists
a.lib.1]} {
 read_lib a.lib.1
} else {
  puts “Error: valid version
of a.lib missing”
  exit
}
```

Checks for version 2 of a.lib first, if it is not found, falls back for version 1 of a.lib. If that is also not found, it exits with error message

```
for {set i 0} \
    {$i < 128} \
    {incr i} {

set_case_analysis 1 \
   [get_pins
block_$i/c_reg/SE]


}
```

Applies case analysis value of 1 on all of 128 pins of

```
block_0/c_reg/SE
block_1/c_reg/SE
…
block_127/c_reg/SE
```

```
foreach clock [list
clockA clockB]

{
  create_clock \
   -period 10.0 \
   -waveform {0 5.0} \
   [get_ports $clock]

}
```

Create clock dynamically for all clocks in the clock list

TCL Control Flow – cotd.

```
switch <options> -- $var {
  pattern1 {script1}
  pattern2 {script2}
  pattern3 {script3}
  …
  default {script_last }
}
```

`<options>: -exact |-glob|-nocase | -regexp [-matchvar var_name] [-indexvar var_name]`

`--` marks end of options

Tests the value of `$var *` against each pattern (pattern1, patter2, pattern3 etc.)

Executes `scriptX` corresponding to `patternX` matching the value of `$var`

`default` clause must be the last entry

```
while {expression} {
   Script
}
```

Executes `Script` is executed as long as `expression` evaluates to a non-zero (true) value

TCL Control Flow – contd.

```
Tempus 1> set num 5
5
Tempus 2> switch -- $num {
           1 – 3 – 5 – 7 - 9 {puts “$num is Odd"}
           2 – 3 – 5 - 7     {puts “$num is Prime"}
           0 – 2 – 4 – 6 – 8 {puts “$num is even”}
           default { puts “$num out of range !!” }
}

5 is Odd
```

```
Tempus 3> set x 0; while {$x<10} { puts “X=$x”; incr x  }
```

TCL – Basic file commands

Use `file` command for basic file operations on a file’s name or its attributes

A few file subcommands	Description of subcommand
`file exists fname`	Returns 1 if the file exits 0 otherwise
`file normalize fname`	Returns the absolute path of parent working directory
`file extension fname`	Returns characters after last dot in fname
`file join str1 str2 ...`	Joins str1, str2, … to form a file path adding file separator as per OS

Use `glob` command to generate list of filenames matching one or more patterns

```
glob pattern1 pattern2 pattern3 ...
```

```
tempus 1> set timing_libs [glob *.lib]
slow.lib fast.lib tsmc_40.lib …
```

Use `cd` and `pwd` commands to change directory and name of current working directory respectively

TCL – Basic file commands

Use `open` command to open a file and get a handle to it

```
open file_name ?access_mode?
```

access_mode

```
  r  : READ_ONLY          r+ : READ+WRITE
  w  : WRITE_ONLY         w+ : READ+WRITE
  a  : APPEND_TO_FILE     a+ : READ+APPEND_TO_FILE
```

Returns a file ID string to identify the file for further interaction with it
Use `close` command to cease interaction with the file

```
close $file_id
```

Example:

```
set fid [open “test.sdc” w+]
# write something to it and/or read from it
close $fid
```

TCL – reading from / writing to a file

Use `puts` command to write data into a file

```
puts $file_handle var
```

Use `gets` command to read a single line from a file

```
gets $file_handle var
```

`$file_handle` : file ID of the file obtained from `open` command

`var` : the variable that stores the line retrieved (for `gets`) or holds the data to be written (for `puts`)

Use `read` command to slurp-read all contents of a file

```
read $file_handle
```

TCL – reading from / writing to a file (template)

```
set fid [open “hold_report.txt” r]
while {[gets $fid line] >= 0} {

    # do what you like here

}
close $fid

# Slurp up the data file
set fp [open "somefile" r]
set file_data [read $fp]
close $fp

#  Process data file
set data [split $file_data "\n"]
foreach line $data {
     # do some line processing here
}
```

Creating Collection of design objects

- **`get_*`** / **`all_*`** commands create a collection (a set of design objects) in current session
- Empty string “” is equivalent to **Empty collection**
- A collection persists only you set the result of a collection command to a variable
- Wildcarding through * (any sequence of characters) and ? (any single character) allowed

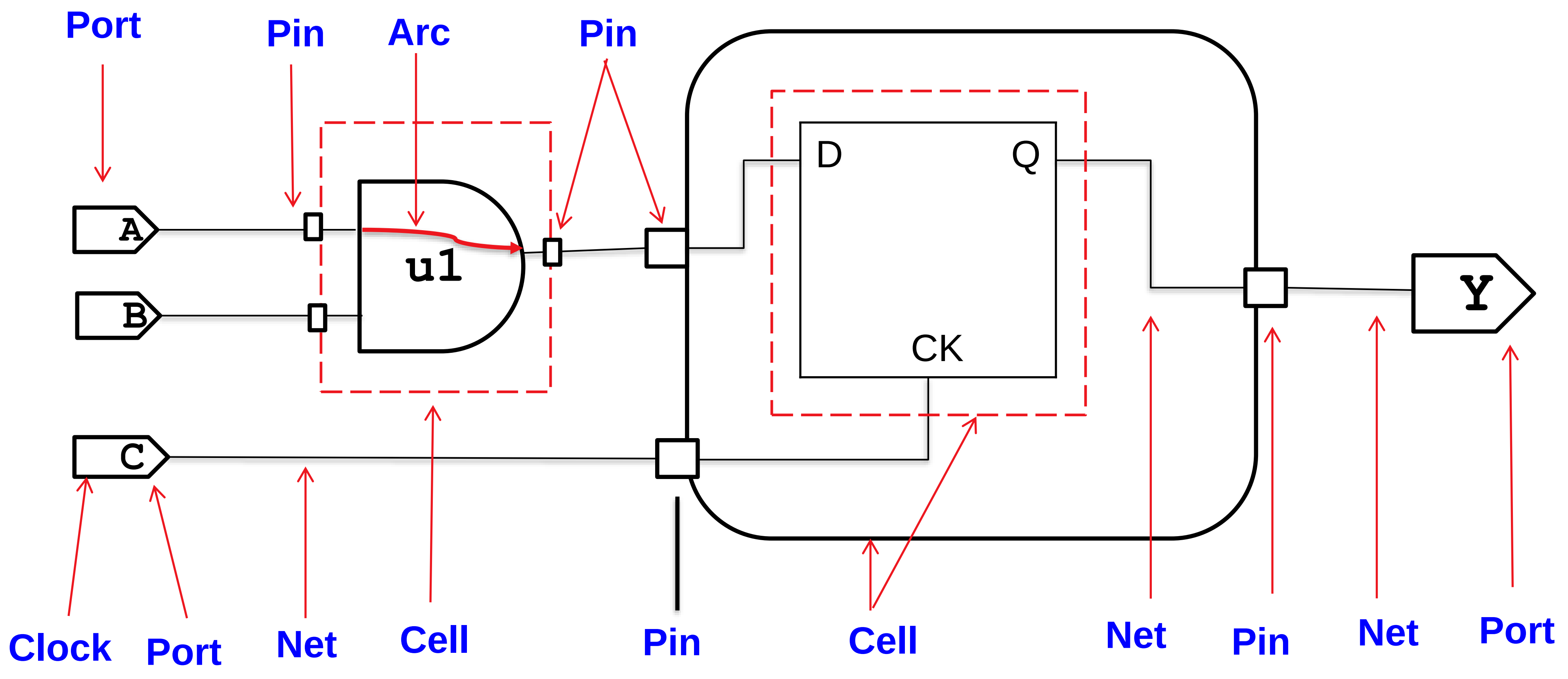
```
tempus 1> set data_ports [all_inputs -no_clock]
Data_in Data_out Reset Address Enable

tempus 2> set dft_ports [get_ports scan_*]
scan_in scan_out scan_mode
```

Module 2 – Tempus/Innovus Object Class

Tempus/Innovus Design Objects

Tempus builds an internal database of ***objects*** to represent the design and libraries

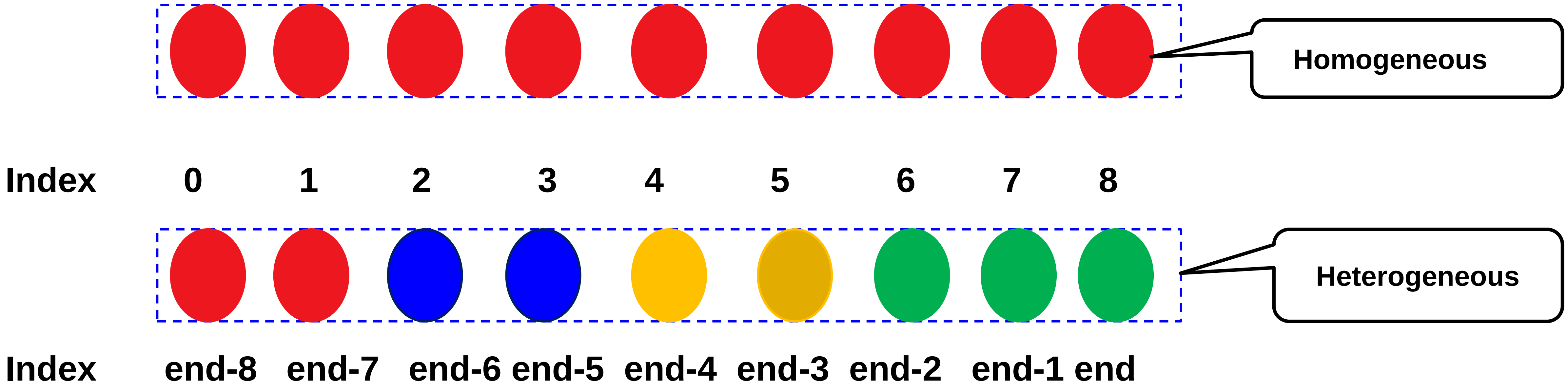


Internal database consists of several **classes** of objects such as **ports, cells, nets, pins, clocks**

Collection – The object container

Makes a group of objects available to Tcl interface for database access

- **Homogenous collection** contains only one type of objects
- **Heterogenous collection** contains more than one type of objects
- Commands that accept collection as input can accept both kinds
- Objects in a collection are indexed from **0** onwards till last object identified by index **end**



Tempus/Innovus Object Class Catalogue

Object Class	Description	Command(s) to create object collection
timing_path	Timing Path	`report_timing -collection`
timing_arc	Timing Arc	`get_arcs`

cell	Instance in the design (hierarchical / primitive li brary cells)	`get_cells`
		`all_instances`
		`all_registers`
		`all_fanin/all_fanout -only_cells`

clock	Clock in the design	`get_clocks / get_generated_clocks /all_clocks`
design	Design	`get_designs`
lib_timing_arc	Timing arc on a library cell	`get_lib_arcs`
lib_cell	Cell in a logic library	`get_lib_cells`
lib	Library	`get_libs`
net	Net in the current design	`get_nets` `all_connected`
path_group	Path groups	`get_path_groups`
pg_net	Power / Ground nets	`get_pg_nets`
pg_pin	Power / Ground pins	`get_pg_pins`
lib_pg_pin	Power/Ground pins in Library	`get_lib_pg_pins`

pin	Instance pin	`get_pins`
		`all_fanin`
		`all_fanout`
		`all_connected`

port	Input/Output/Bidir ports of current design	`get_ports / all_inputs / all_outputs / all_connected / all_fanin / all_fanout`

Displaying Objects in a collection

Internal representation provides reference to the actual object and access its attributes

String representation of collection can be displayed by :

`query_objects` *`collection [-limit <count>]`*

- ❑ Display objects in a collection. You can limit the display to *`<count>`* numbers with *`–limit`* option.

`get_object_name` *`collection`*

- ❑ Returns TCL list of strings holding object names in a collection
- ❑ The output can be saved in a variable or used implicitly with nested commands

```
tempus 1> query_objects [get_ports in*]
in0 in1 in2 in3 in4
tempus 2> get_object_name [get_ports scan_*]
scan_in scan_out scan_mode
```

Counting objects in a collection

```
size_of_collection collection
```

- ❑ Returns the number of objects in a collection
- ❑ Returns 0 for empty collection (empty string)
- ❑ More efficient that manually iteration and counting of elements

```
tempus 1> Puts “No of ports: [sizeof_collection [get_ports]]”
30
tempus 2> Puts “No of flops clocked by clockA : \
                [sizeof_collection \
                   [all_registers –flop –clock clockA ]]”
0
# No flop in the design is clocked by clockA
```

Selecting Objects in a collection

`range_collection` *`collection from_index to_index`*

- Returns a new collection extracting elements staring from *`from_index`* up to *`to_index`*
- *`from_index/to_index`* must be integer or *`end`* or *`end-integer`*
- 16.23 or later, returns objects in reverse order if *`from_index>to_index`*

`index_collection` *`base_collection index`*

- Returns a new collection containing only the single object at the index of *`base_collection`*

```
tempus 1> set collection1 [get_cells]
tempus 2> range_collection $collection1 end-5 end
TDSP_DS_CS_INST TDSP_MUX TEST_CONTROL_INST ULAW_LIN_CONV_INST PLLCLK
_INST ROM_512x16_0_INST
0x487
tempus 3> query_objects [index_collection $collection1 2]
FE_SIG_C663_port_pad_data_out_13_
```

Adding/Appending objects in a collection

`add_to_collection base_collection second_collection_or_list [-unique]`

- ❑ Creates a new collection by copying objects from base collection and then adding objects second collection / list of objects to base collection (assign it to a variable to use the collection later)
- ❑ Base collection could be an empty collection
- ❑ If `base_collection` is ***homogeneous***, only another collection of same type can be added to it

`append_to_collection collection`

- ❑ Updates an existing collection by appending objects from second collection or list
- ❑ Faster than `add_to_collection`
  NOTE: You can add multiple collections in a simple Tcl list also

```
tempus 1> set collection1 [get_ports]
tempus 2> set new_collection [add_to_collection $collection1 [get_cells]]
tempus 3> append_to_collection new_collection [get_clocks] -unique
```

Iterating over a collection

Collection is Not a Tcl List – `for/foreach/while` does not work on it !!

`foreach_in_collection` *`var`* `base_`*`collection`* `{` *`script`* `}`

- ❑ Iterator variable *`var`* is set to a collection of exactly one object at a time
- ❑ *`base_collection`* could be an implicit or explicit collection
- ❑ Commands in *`Script`* applied at each Iteration

```
tempus 1> foreach_in_collection cell [get_cells] {
               Puts “[get_object_name $cell]
          }
```

Removing objects in a collection

```
remove_from_collection base_collection ref_collection_or_list[-intersect]
```

- ❑ Creates a new collection from base collection by removing reference elements specified as Tcl list / one of the Tempus collections
- ❑ `-intersect` generates intersection of base collection elements with second collection of objects or list

```
tempus 1> set data_ports \
           [remove_from_collection [all_inputs] CLK]
           {"in1", "in2", "in3"}

tempus 2> set_false_path -from [get_clocks CLK1] \
            -to [remove_from_collection [all_clocks] [get_clocks CLK1]]
```

Filtering objects from a collection

`filter_collection` *`base_collection filter_expression [-nocase] [-regexp]`*

- ❑ Creates a new collection including only those objects from `base_collection` that match the expression specified by *`filter_expression`* or an empty collection in case of no match
- ❑ *`get_*`* commands that provide a –filter option filters out before including in the collection, so more efficient
- ❑ `-nocase` can be used only with `–regexp`

```
tempus 1> filter_collection [all_registers] "is_memory_cell==true"
RAM_128x16_TEST_INST/RAM_128x16_INST RAM_256x16_TEST_INST/RAM_256x16_INST

tempus 3> set leaf_cells [get_cells –hier * -filter "is_hierarchical == f
alse“]
         {“U1/i1", “U2/i2“}
```

Syntax of filter_expression

- ❑ Use curly brace { } or double quotes to " " enclose the expression
  `{filter_expression} or "filter_expression"`

  If it is a string value compared using logical comparison, quotation not mandatory

- ❑ TCL conditional expression contrasts on one or more object properties of the given type design-object against its value or its existence or non-existence in one of these forms:

  property-name ***RELATIONAL_OPERATOR*** *VALUE_TO_MATCH (String/Number/Boolean)*
  : Returns 1 for a match 0 in case of no match

  defined*(property-name)*
  :Returns 1 if the property is defined for the objects

  undefined*(property-name)* **:** opposite of *defined()*

- ❑ Club multiple conditions using logical AND (AND and &&) and logical OR (OR and ||) operators
  `"is_hierarchical == true AND area <=6"`

Supported Operators

- ❑ Supported Relational operators

== (Equal)	**=~** (Matches Patterns with * and ?)
!= (Not Equal)	**!~** (Does not match pattern with * and ?)
> (Greater Than)	**&&** (Logical AND)
< (Less than)	**AND** (Logical AND)
>= (Greater or Equal)	**||** (Logical OR)
<= (Less or Equal)	**OR** (Logical OR)

- ❑ With **`-regexp`** option **=~** and **!~** you can use any kind of Tcl regular expressions

- ❑ Supported existence operators

  defined **undefined**

Relational Rules for filter_expression

- String property can be compared with any operator
- Numerical property CAN NOT be compare with pattern match operators
- Boolean property can be compare ONLY with == and != against a value true/false
- Existence operator can be applied on any valid property
- Use parenthesis to group expressions to enforce order – else parsed left to right
- Regular expression match assumes the pattern anchored
  Prefix/Suffix the pattern with .* (dot star) to widen the search
- Use `–nocase` with `–regexp` to make case-neutral pattern match

```
tempus 1> set nomoded_arcs [filter_collection \
              [get_arcs –of_objects [get_cells *]] \
              "undefined(mode)"]
```

Sorting objects in a collection

```
sort_collection base_collection list_of_attributes [-dictionary]
                [-dictionary]
```

- Base collection must be homogenous for sorting
- One or multiple attributes can be used as sort keys

```
tempus 1> sort_collection [get_ports *] {direction full_name}
         {"in1", "in2", "out1", "out2"}
```

Comparing two collections

```
compare_collections collection1 collection2 [-order_dependent]
```

- ❑ Compares two collections (optionally considering object order in addition)
- ❑ Returns 0 when same objects are present in both collections, else return non-zero value
- ❑ `-order_dependent` additionally requires the object order to be same

```
tempus 1> set c1 [get_cells {u1 u2}]
{u1 u2}
tempus 2> set c2 [get_cells {u2 u1}]
{u2 u1}
Tempus 3> compare_collections $c1 $c2
0
tempus 4> compare_collections $c1 $c2 -order_dependent
-1
```

Creating a duplicate collection

```
copy_collection collection1 collection2
```

- Copies all objects of `collection1` to `collection2` in the same order
- Modifying or killing master collection does not affect the copied collection

```
Tempus 1> set c1 [get_cells "U1*"]
{U1 U10 U11 U12}
Tempus 2> set c2 [copy_collection $c1]
{U1 U10 U11 U12}
Tempus 3> unset c1
Tempus 4> query_objects $c2
{U1 U10 U11 U12}
```

Properties of a Design Object

- ❑ Design Object details stored in terms of a list of properties as ***Property⬄Value*** table
- ❑ Data type of the ***Value*** field determines how it can be queried or passed to other commands

port

Property	Value	Data type
name	DIN	string
direction	input	string
arrival_max	0.17	float

net

Property	Value	Data type
name	n1	string
driver_pins	{I1/Z}	collection
num_load_pins	10	integer

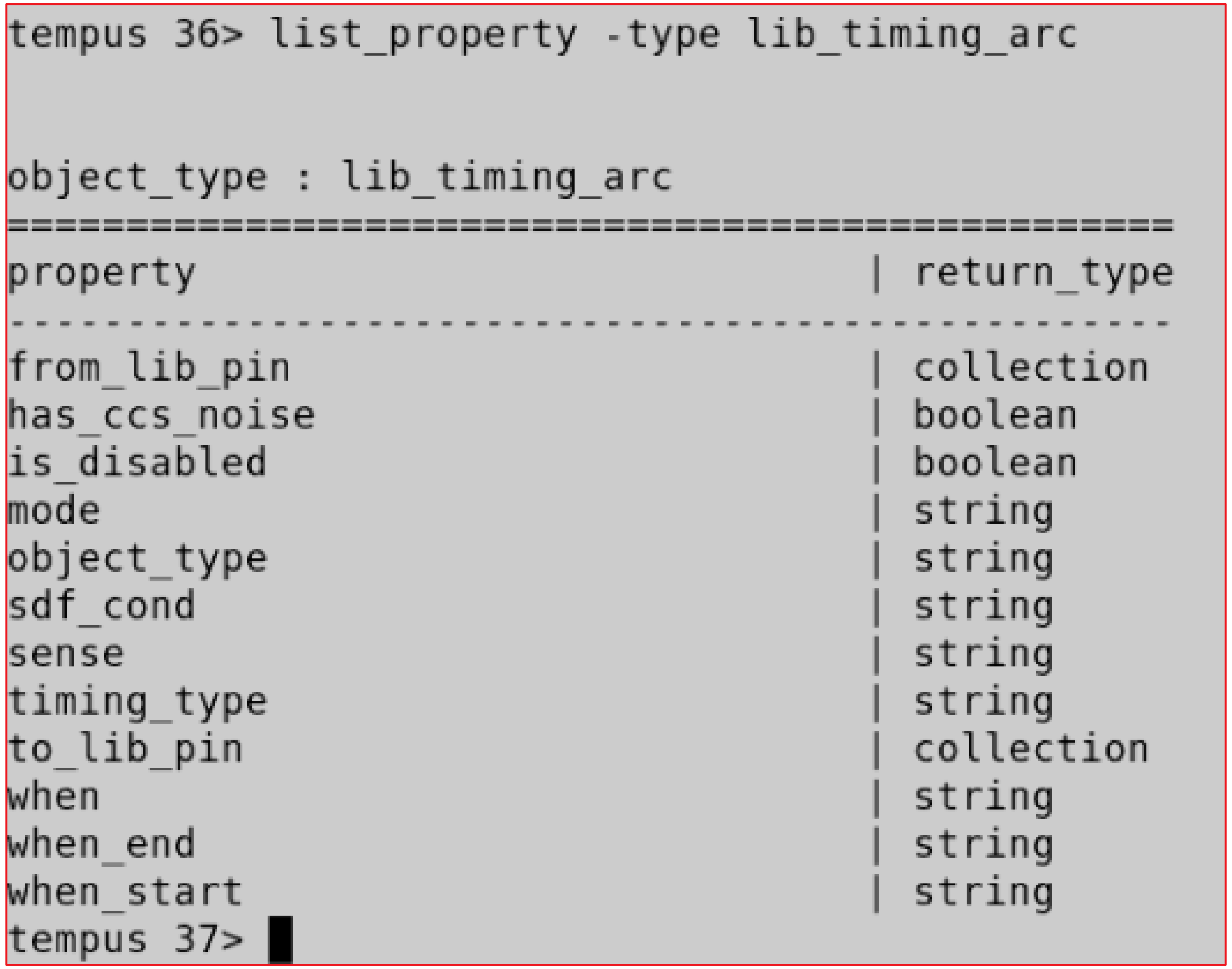
timing_arc

Property	Value	Data type
arc_type	combinational	string
delay_max	0.34	float
sink_pin	{I1/Z}	collection

Property names and the types

`list_property [-type <object_type>]` returns the valid list of properties and its data type

```
object_type
pin
port
cell
net
clock
lib_cell
lib_pin
design
lib
timing_path
timing_point
timing_arc
path_group
lib_timing_arc
si_victim
si_attacker
pg_pin
pg_net
```



```
tempus 36> list_property -type lib_timing_arc


object_type : lib_timing_arc
==================================================
property                                  | return_type
--------------------------------------------------
from_lib_pin                              | collection
has_ccs_noise                             | boolean
is_disabled                               | boolean
mode                                      | string
object_type                               | string
sdf_cond                                  | string
sense                                     | string
timing_type                               | string
to_lib_pin                                | collection
when                                      | string
when_end                                  | string
when_start                                | string
tempus 37> 
```

Report the property value pair of a Design Object

```
report_property [-property_list <property list>] \
                {<collection> | <list_of_collections>}
```

- `report_property` returns all properties if `–property_list` not mentioned
- Property value may be empty for some objects

```
tempus 61> report_property [get_cells clock_cell_1] -property_list {ref_lib_cell_name area}
property                                | value
-------------------------------------------------
ref_lib_cell_name                       | MX4X1
area                                    | 32.251
```

Retrieving property value of an Object

`get_property` *`collection property`* `[-clock` *`clock_name`*`] [-quiet] [-view` *`view_name`* `]`

- ❑ If the `collection` contains more than one objects, the property value is returned for all of them
- ❑ `-quiet` suppresses error/warning from `get_property` command
- ❑ To capture output of `get_property` command in log, use `Puts` command to print it
- ❑ Query `object_type` property to know Tempus data type (as a string) of a design object

```
tempus 1> get_property [get_clocks gen_clock1] sources
PLL_INST/ckout_1
0x443

tempus 2> get_property [get_property [get_clocks gen_clock1] sources] object_type
pin

tempus 3> set worst_path [report_timing -collection]
0x4b9
tempus 4> get_object_name [get_property $worst_path capturing_clock]
m_rcc_clk
tempus 5> get_property [get_property $worst_path capturing_clock] object_type
clock
```

Module 3 - Working with Procedures

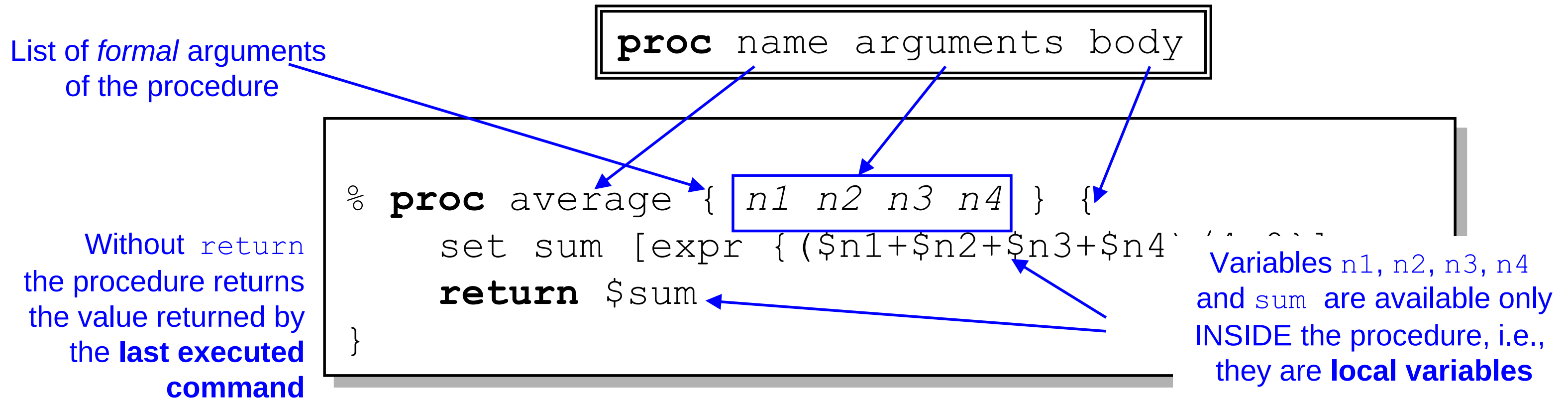
Creating Procedures in Tempus/Innovus

Procedure is a named block of Tcl and Tempus commands to perform a particular task or function

Use `proc` command to create new Tcl commands.

- New commands **look** just like built-in commands.
- There is a single global scope for procedure name
  - Usable/Visible everywhere (no local declaration)
  - New creation overrides existing with the same name without checking

Argument list is usually enclo
sed within curly braces but th
is is not mandatory.



```
proc name arguments body
```

```
% proc average { n1 n2 n3 n4 } {
    set sum [expr {($n1+$n2+$n3+$n4
    return $sum
}
```

List of *formal* arguments of the procedure
Without `return` the procedure returns the value returned by the **last executed command**
Variables `n1`, `n2`, `n3`, `n4` and `sum` are available only INSIDE the procedure, i.e., they are **local variables**

Invoking procedures

- Invoke procedure by specifying procedure name followed by the arguments
- The number of arguments passed must match the count of formal arguments
- Checking of data types of arguments is not automatically done by Tcl
- To save the procedure result, assign it to a variable, like other Tcl and Tempus commands
- To print the procedure result, use `puts / Puts` command

```
tempus 1> average  1 2 3 4
2.5
tempus 2> average -10 10 -50 3
-11.75
tempus 3> puts "average:\t[average -10 10 -50 3]"
average: -11.75
tempus 4> set mean [average 1 2 3 4]
2.5
```

Procedure with default values

Procedures can have *optional* arguments.

- When arguments are lists of two items:
  - Argument name
  - Default value
  - Arguments without default must be specified before arguments with defaults

```
proc print_log { log {fid stdout} {prefix LOG:} } {
   foreach line $log {
        puts $fid "$prefix $line“
    }
}
```

Using procedures with *optional* arguments

```
% print_log $text
LOG: ...
% print_log $text $open_fid
% print_log $text $open_fid SYNTH:
```

Use optional procedure argument s to improve Tcl code re-use, read ability and maintainability.

Procedure with variable number of arguments

Procedures can also have *variable* number of arguments.

- Use special argument specifier `args` to absorb all argument values not matched by the preceding arguments.
- `args` is an ordinary Tcl list and must be the last argument

```
proc print_log_args  { log {fid stdout} args} {
    set prefix "LOG:"; set header "";
    foreach {option value} $args {
        switch -- $option {
            -prefix         {set prefix $value}
            -header         {set header $value}
            default         {error "ERROR: Unknown option \"$option\"."; retur
n}
        }
    }
    if {$header != ""} {puts $fid $header}
    foreach line $log {puts $fid "$prefix $line"}
}
```

Handling Arrays in Procedures

Make the array a global variable and access inside the procedure

Use array set to convert a Tcl list into an array
Use array get to extract values from the array

```
tempus 1 > proc foo { arg_list } {
   # arg_list was an array in the main code
   array set my_array $arg_list;
   # manipulate my_array
   return [array get my_array];
}
tempus 2> set cell(one) {I1/U1};
tempus 3> set cell(two) {I1/U2};
tempus 4> array set new_cell [foo [array get cell]]
```

Module 4 - Enhance Procedure Usability

Extend procedure in Tempus/Innovus environment

- ❑ Add help message
- ❑ Add Argument Validation
- ❑ Hide procedure code (`info body proc_name` reveals code)

Add help message to procedures

```
define_proc_arguments proc_name \
  [-info info_text ] \
  [-define_args arg_defs ] [-hide]
```

- ❑ Multiple `define_proc_arguments` overwrites the previous one
- ❑ `-define_args` argument is a list of lists, where each list element has the following format:

```
arg_name option_help value_help data_type attributes
```

Components of -define_args

Argument	Description	Supported Values	Mandatory
`arg_name`	Name of argument	`-<string>`	Yes
`option_help`	Short description of argument	`“<text message>”`	No
`value_help`	Argument name for positional arguments		No
`data_type`	Data type of an argument	**`string`**`/list/Boolean/int/float/one_of_string`	No

`attributes`	Additional attributes for an argument
	Must for Argument Validation
	
	`value_help`
	lists valid values for `one_of_string`
	
	`mutual_exclusive_group` includes the current option in the *group_name* group
		`reuired|`**`optional`**`|internal`
		`value_help`
		`values {allowed values}`
		`merge_duplicates`
		`bind_option` *`other_arg_name`*
		`mutual_exclusive_group` *`group_name`*
			No

Argument validation of procedures

```
parse_proc_arguments -args arg_list result_array
```

- ❑ Parses arguments defined with `define_proc_arguments` command
- ❑ Stores arguments in the array `result_array` with array-keys as defined in `define_proc_arguments`

```
proc plot_waveform {args} {
   parse_proc_arguments -args $args results
   foreach argname [array names results] {
      puts " $argname = $results($argname)“
   }
   .....
}

define_proc_arguments plot_waveform \
  -info “plot graphically the waveforms as reported by report_delay_calculation” \
  -define_args {\
     {-file "" "" string required} \
     {-gnuplot_file "" "" string optional} \
     {-gnuplot_cmd_file "" "" string optional} }
```

Example of argument parsing and help message

```
tempus 1> proc sum {a b} {
  parse_proc_arguments -args $args result
  return [expr $result(-a) + $result(-b)]
}

tempus 2> define_proc_arguments sum \
-info "Add two numbers" \
-define_args {{"-a" "first addend" "operend1" double required} \
              {"-b" "second addend" "operend2" double required}}

tempus 3> sum –help
Description:
Add two numbers

Usage: sum [-help] -a <operend1> -b <operend2>

-help                # Prints out the command usage
-a <operend1>        # first addend (float, required)
-b <operend2>        # second addend (float, required)

tempus 29> sum  -a 10.9 -b 20.5

31.4
```

cādence®