



DC Tcl-me Tutorial



©1996 Children's Television Workshop (CTW). Sesame Street
Muppets©1996 Jim Henson Productions, Inc. All rights reserved.



Agenda



Introduce DC Tcl

- **Setup file changes**
- **DC Tcl Shell Basics**
- **DC Extensions to the Tcl language**
- **Writing Procedures**
- **Differences between DCSH and DC Tcl**
- **Converting DCSH scripts to DC Tcl**



About DC Tcl Shell

- **Native Implementation of the Industry Standard Tcl Scripting Language**
- **DC-Tcl is NOT a wrapper around dc_shell!**
- **A DCSH to DC-Tcl translator is provided**
- **Both DCSH and DC-Tcl interfaces are available**



About DC Tcl Shell

- **Either shell is specified when dc_shell is invoked**
 - Existing dc_shell is called DCSH mode
 - New Tcl dc_shell is called DC-Tcl mode
- **Once DC-Tcl mode is invoked it is not possible to switch back to DCSH mode in the same dc_shell session**
- **For Tcl version information, type: `info tclversion`**



Tcl support in dc_shell does not imply support for commands and attributes from PrimeTime



Agenda

- **Introduce DC Tcl**
- ➔ **Setup file changes**
 - **DC Tcl Shell Basics**
 - **DC Extensions to the Tcl language**
 - **Writing Procedures!**
 - **Differences between DCSH and DC Tcl**
 - **Converting DCSH scripts to DC Tcl**



New .synopsys_dc.setup file

- **Beginning with 1999.05 the format of the root .synopsys_dc.setup file has changed**
- **The root .synopsys_dc.setup and supplemental files that are shipped with the release are now in a Tcl Subset (Tcl-s)**
- **Common syntax for compatibility with DCSH**
- **A utility is provided that can translate your DCSH scripts and setup files**



New .synopsys_dc.setup file (cont)

- **Commands supported in Tcl-s are:**
 - alias
 - annotate
 - define_name_rules
 - get_unix_variable
 - getenv
 - group_variable
 - if
 - set
 - set_layer
 - set_unix_variable
 - setenv
 - source



New .synopsys_dc.setup file (cont)

- For DCSH users, the home and local setup files can be in either Tcl-s or DCSH format
- For DCSH users, if the local setup file is in Tcl-s, then the home setup file must be in Tcl-s as well
- Valid Setup Configurations

dc_shell mode	Root setup	Home setup	Local setup
DCSH	Tcl-s	Tcl-s	Tcl-s
DCSH	Tcl-s	Tcl-s	DCSH
DCSH	Tcl-s	DCSH	DCSH
Tcl	Tcl-s	Tcl	Tcl



Agenda

- **Introduce DC Tcl**
- **Setup file changes**
- ➔ **DC Tcl Shell Basics**
 - **DC Extensions to the Tcl language**
 - **Writing Procedures**
 - **Differences between DCSH and DC Tcl**
 - **Converting DCSH scripts to DC Tcl**



DC Tcl Shell Basics

To Run DC Tcl Shell from the UNIX command line:

```
% dc_shell-t  
OR  
% dc_shell -tcl_mode
```

```
DC Professional (TM)  
DC Expert (TM)  
Version 1999.05  
Copyright (c) 1988-1998 by Synopsys, Inc.  
ALL RIGHTS RESERVED
```

```
This program is proprietary and confidential information of Synopsys, Inc.  
and may be used and disclosed only as authorized in a license agreement  
controlling such use and disclosure.
```

```
Initializing...  
dc_shell-t> _
```

To Exit DC Tcl:

```
dc_shell-t> exit  
  
Memory usage for this session: 2091 Kbytes  
CPU usage for this session 3 seconds.  
  
Thank you...  
%_
```



Getting Help

- To get a basic summary of all DC Tcl commands:

```
dc_shell-t> help
Procedures:
    (user-defined procedures)
Builtins:
    (a very long list...)
Default Command Group:
    (another very long list...)
```

- Use a wildcard to find a command:

```
dc_shell-t> help *clock
clock                # Builtin
create_clock         # create_clock
create_test_clock    # create_test_clock
remove_clock         # remove_clock
remove_propagated_clock # remove_propagated_clock
report_clock         # report_clock
set_propagated_clock # set_propagated_clock
```



More Help!

- Use `help -verbose` for command syntax info:

```
dc_shell-t> help -verbose set_input_delay
set_input_delay      # Define arrival time relative to clock
  [-clock clock_name] (relative clock)
  [-clock_fall]       (delay is relative to falling edge of clock)
  [-level_sensitive]  (delay is from level-sensitive latch)
  [-rise]             (specifies rising delay)
  [-fall]             (specifies falling delay)
  [-max]              (specifies maximum delay)
  [-min]              (specifies minimum delay)
  [-add_delay]        (don't remove existing input delay)
  delay_value         (path delay)
  port_pin_list       (list of ports and/or pins)
```

- Use `man command` for complete man page info
- Also use `command -help`

Comments in DC Tcl Scripts

- Comment a line in a Tcl script using the '#' character:

```
# This is an example of a comment in a DC Tcl script
set period 15

# If you wish to comment on the same line, be sure to use
# a semicolon before the comment:
set header_str "Output Header"; # Same line comment
```



This semicolon is required!

✘ **NOTE:** There is no `"/ * */`, start/end comment in Tcl, just `'#'` which comments to end of line.



Using Variables

- Variables are assigned using the Tcl **set** command. To assist in compatibility, DC Tcl also supports the “=” construct (in a limited way)
- Variables need to be de-referenced using the “\$” character before the variable name (similar to the UNIXtm cshell)
- Variables are removed using **unset varname**
- Print variables using **printvar varname**



TCL Variable Examples

```
dc_shell-t> set clock_period 10  
10
```

```
dc_shell-t > set search_path ". /u/synopsys/libraries/syn"  
. /u/synopsys/libraries/syn
```

```
dc_shell-t > echo "clock period = " $clock_period  
clock_period = 10
```

```
dc_shell-t > echo clock_period  
clock_period
```

- **Variables are NOT strongly-typed in Tcl**



Nesting Commands

```
dc_shell-t> set_input_delay 5 -clock CLK [all_inputs]  
dc_shell-t> set index [lsearch [set a [lsort $l1]] $aValue]  
dc_shell-t> set source_files [glob src/*.v]
```

- **Nested commands are very useful in Tcl**
- **Commands are nested using “[“ and “]”:**
- **command1 [command2 [command3 ...]]**
- **Inner command is executed, then its result is passed to the outer command**



Abbreviations

```
dc_shell-t> source -echo myscript.tcl  
dc_shell-t> so -e myscript.tcl; #This works, too!
```

- **You can abbreviate commands and options**
 - Use the shortest unique abbreviation
- **You can alias commands**
 - Caveat: The alias can't be an existing command



Using Wildcards

- DC Tcl supports the wildcard character '*'
- Examples:

```
dc_shell-t> help create*  
dc_shell-t> create_clock -period 10 [get_port CLK*]
```

Display all DC Tcl commands
which begin with "create"

Matches ports CLK1, CLK2, CLK_FAST, etc.

Quoting - “ ” vs. { } in Tcl

- Use backslash (\) to escape individual characters
- Two ways of quoting text in Tcl:
 - Using “ and ” - weak quoting because variable, command, and backslash substitution will still occur
 - Using { and } - rigid quoting because no substitutions will occur
- Examples:

```
dc_shell-t> set a 5
```

```
5
```

```
dc_shell-t> set s "temp = data[$a]"
```

```
temp = data[5]
```

\$a is substituted with its value!

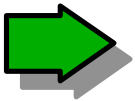
```
dc_shell-t> set s {temp = data[$a]}
```

```
temp = data[$a]
```

"temp = data[\$a]" is literal text

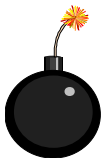
Using Lists in DC Tcl

- Lists are a key component of Tcl
- Lists can be built as strings separated by whitespace:



```
dc_shell-t> set l1 {e11 e12 e13};  
dc_shell-t> set l2 "e11 e12 e13"; #Command/var subst may occur!
```

- Don't use a comma to separate list items, a la DCSH:



```
dc_shell-t> set mylist "l1, $netlist, CLK, SEL"; # Wrong!
```



Expressions

- **Tcl is command oriented**
 - few operators are included!
- **For numerical operations**
 - Use the `expr` command
- **For lists**
 - Use list commands such as `list`, `concat`, `lappend`, etc.
- **Example expressions in DC Tcl:**

```
dc_shell-t> set period 10.0
10.0
dc_shell-t> set freq [expr ((1 / $period) * 1000)]
100.0
```

Control Flow

- Using the if command:

must be on same
line as the else!

```
if {$period > 10.0} {  
    echo "Clock period = $period (exceeds 10 ns)"  
} else {  
    echo "Clock period = $period"  
}
```

- Using the while command:

```
set idx 0  
set found "false"  
while {$found == "false"} {  
    if {$pin($idx) == "A"} {  
        set found "true"  
        echo "Pin A found"  
    }  
}
```

More Control Structures

- Examples of *for*, *foreach*, and *switch*:

```
# for loop example:
for {set cell 0} {$cell < $cell_tot} {incr cell} {
    echo "Cell $cell is $list($cell)"
}

# foreach loop example - iterates over elements of a list
set srcfiles [glob src/*.v]
foreach srcfile $srcfiles {
    analyze -f verilog $srcfile
}

# switch example (case statement):
switch -exact $x {
    a      {incr t1}
    b      {incr t2}
    default { echo "Nevermind" }; # must be last
}
```



Tcl References

- **See a Tcl manual for more details**
- **We recommend:**
 - **“Practical Programming in Tcl and Tk”
by Brent B. Welch, Prentice Hall**
 - **“Tcl and the Tk Toolkit”
by John Ousterhout, Addison Wesley**
 - **<http://www.scriptics.com>**
 - **Design Compiler Command-Line Interface Guide
Synopsys Online Documentation, v1999.05**



Agenda

- **Introduce DC Tcl**
- **Setup files changes**
- **DC Tcl Shell Basics**
- ➔ **DC Extensions to the Tcl language**
- **Writing Procedures**
- **Differences between DCSH and DC Tcl**
- **Converting DCSH scripts to DC Tcl**



Objects in DC Tcl

- **Designs consist of objects**

- designs
- cells
- ports
- pins
- clocks
- nets
- etc...

- **To select objects**

```
dc_shell-t> set var [get_objtype pattern]
```

- where *objtype* is one of:

- » cells, clocks, designs, libs, lib_cells, lib_pins, nets, pins, ports, etc...

- returns a collection handle

```
dc_shell-t> set dports [get_ports {data[*]}]  
{ "data[0]" "data[1]" "data[2]" }
```



DC Tcl Collections

- **DC commands**
 - Accept either a collection or a list for arguments
 - Always return a collection
- **Collections are more efficient than lists**
- **There are collection commands to:**
 - Select objects
 - Manipulate objects
 - Query objects
 - Filter objects



DC Tcl Collections (cont'd)

- **Collections can generally be:**
 - Assigned to a variable
 - Passed to another command (using nested commands)
 - Queried
 - Filtered

```
dc_shell-t> set clks [get_port CLK*]  
dc_shell-t> printvar clks  
clks                = "_sel20"  
dc_shell-t> create_clock -period 10 $clks; # Good!
```

Or...

```
dc_shell-t> create_clock -period 10 [get_port CLK*]; # Good!
```

But:

```
dc_shell-t> create_clock -period 10 "_sel20"; # BAD - won't work
```

DC Tcl Collections (cont'd)

- Typical usage of collections:

```
dc_shell-t> set data_ports [get_port data[*]]
dc_shell-t> query_objects $data_ports
{"data[0]", "data[1]", "data[2]"}
dc_shell-t> set_input_delay 3.0 -clock {CLK} $data_ports
dc_shell-t> query_objects [all_outputs]
{"out1", "out2", "out3"}
dc_shell-t> set_output_delay 5.5 -clock $clk [all_outputs]
```

- **NOTE:** `query_objects` doesn't return anything!
 - Use `get_object_name` to get the name of an individual object within a collection
- **This will NOT work as intended!**

```
dc_shell-t> set notalist [query_objects [all_outputs]]
{"data[0]", "data[1]", "data[2]"}
dc_shell-t> printvar notalist
alist                = ""
```

Iterating Over a Collection

- The Tcl foreach command cannot be used for collections since it operates on lists
- Use DC Tcl's foreach_in_collection instead:

```
foreach_in_collection variable collection(s) { body }
```

Loop iterator
variable

collection(s) you want
to iterate over

body of the loop
to be executed

```
# Script to loop thru all clocks and print their period
foreach_in_collection clk_itr [all_clocks] {
    set clk_name [get_attribute $clk_itr full_name]
    set clk_per [get_attribute $clk_itr period]
    echo "Clock period for $clk_name is $clk_per"
}
```

Manipulating Collections

- Add objects from a collection using `add_to_collection`
- Remove objects to a collection using `remove_from_collection`
- Analogous to using list addition(+) and subtraction (-) operators in DC

```
dc_shell-t> set mysel [add_to_collection [get_port "DATA*"] [get_port "CTRL*"]]  
dc_shell-t> set mysel [remove_from_collection [all_inputs] [get_port "CLK"]]
```

Same as DCSH's `all_inputs()` - "CLK"



Filtering Collections

- Similar to filter in DCSH
- Filter “on the fly” with -filter option of many DC Tcl commands
- More efficient to filter “on the fly” with -filter
- Example:



```
dc_shell-t> set fastclks [get_clock "CLK*" -filter "period < 10.0"]
```




Agenda

- **Introduce DC Tcl**
- **Setup file changes**
- **DC Tcl Shell Basics**
- **DC Extensions to the Tcl language**
- ➔ **Writing Procedures**
 - **Differences between DCSH and DC Tcl**
 - **Converting DCSH scripts to DC Tcl**



DC Tcl Procedures

- **Allows the user to write reusable, shareable routines**
- **Powerful features of procedures:**
 - Allow you to define your own commands!
 - Allow any number of arguments (can define default values)
 - Can have variable number of arguments
 - Pass arguments by value or reference
 - Can have local variables
 - Can use any commands or other procedures
 - Supports recursion

Example Tcl Procedure

```
#####
#  PROCEDURE:      collection_to_list
#  ABSTRACT:       Converts a collection of objects to a list of
#                  string object names.
#  RETURNS:        a list of object names (strings)
#                  empty list if it can not get_object_name
#                  in each member of the given collection.
#  SYNTAX:         collection_to_list a_handle
#####
proc collection_to_list { args } {

# This is an example of how to process arguments
  parse_proc_arguments -args $args result_array

# Iterate through the collection, building up the return list
  set var [list ]
  foreach_in_collection a $result_array(handle) {
    set name [get_object_name $a]
    lappend var $name
  }
  return $var;
}
```

Example Tcl Procedure (cont)

```
#
# Define the arguments, command information and usage
#
define_proc_attributes collection_to_list { \
    -info "Create a list from collection" \
    -define_args {
        {handle "A collection" handle string required} \
    }
}
```

- **For help on how to use variable arguments:**

```
dc_shell-t> man define_proc_attributes
```

```
dc_shell-t> man parse_proc_arguments
```



Agenda

- **Introduce DC Tcl**
- **Setup file changes**
- **DC Tcl Shell Basics**
- **DC Extensions to the Tcl language**
- **Writing Procedures**
- ➔ **Differences between DCSH and DC Tcl**
- **Converting DCSH scripts to DC Tcl**

Differences between DCSH and Tcl

DCSH Command	Tcl Command	Function
read	read_file read_db	Reads files into dc_shell same as PrimeTime
list <u>var</u>	printvar <u>var</u>	List the value for variable <u>var</u>
list -files	list_files	Lists the files loaded into dc_shell
list -commands	help list_commands	Lists the commands in the shell
list -licenses	list_licenses	Lists the licenses in use
list -variables	print_variable_group	List variable groups
=	set	Assignment operator
include	source	Includes a file
write	write write_file	Writes out a file



More Differences

- **DC-Tcl uses collections instead of lists**
- **Tcl is CASE SENSITIVE**
- **Tcl will exit a loop at the line where the exit command is given**
- **DCSH will exit a loop after executing the entire loop body first, no matter where the exit command is placed**



More Differences


- You must not alias an existing command in Tcl
- `dc_shell_status` is not supported in Tcl
 - Translator handles most cases
 - May require some hand translation

```
dc_shell-t> set dc_shell_status [command ...]
```

- '\$' character not allowed in variable name in Tcl



Agenda

- **Introduce DC Tcl**
 - **Setup file changes**
 - **DC Tcl Shell Basics**
 - **DC Extensions to the Tcl language**
 - **Writing Procedures!**
 - **Differences between DCSH and DC Tcl**
-  **Converting DCSH scripts to DC Tcl**



DCSH -> DC-Tcl Script Conversion

- Converter is available to ease transition from dc_shell scripts to Tcl scripts - called `dc-transcript`
- Run from UNIX command line:

```
$ dc-transcript dcsch_script tcl_script
```
- Can convert most scripts - cannot convert all scripts 100%
- Will flag constructs that it cannot translate



DCSH -> DC-Tcl Script Conversion

- By default, all include files are translated, then inserted into the `dc-transcript` output file
- Use the `-source_for_include` switch to translate all “include” statements into “source” statements
- The `-source_for_include` switch will not translate the given include scripts



Script Conversion Example

```
search_path = {/remote/release/1999.05/ \
  libraries/syn}
search_path = "." + search_path

alias cc create_clock

target_library = "mylib.db"
link_library = "*" + target_library
read -format db mydesign.db
current_design TOP
link

set_input_delay 5.5 all_inputs() - CLK
set_output_delay 3.25 all_outputs()
cc find(port, CLK)

compile -incremental

report_timing
exit
```

```
set search_path [list \
  /remote/release/1999.05/ \
  libraries/syn]
set search_path [concat {.} $search_path]

#alias cc create_clock

set target_library {mylib.db}
set link_library [concat {*}
  $target_library]
read_file -format db mydesign.db
current_design TOP
link

set_input_delay 5.5
[remove_from_collection \
  [all_inputs] CLK]
set_output_delay 3.25 [all_outputs]
create_clock [find port CLK]

compile -incremental

report_timing
exit
```



DC-Tcl Summary

- **Native implementation of industry standard Tcl**
- **Powerful programming language**
- **Tool provided to convert your old scripts**
- **Extensions provided to make Tcl even more powerful with Design Compiler and for consistency with PrimeTime**