

Getting Started with OrCAD X TCL

Product Version 23.1

September 2023

© 2023 Cadence Design Systems, Inc.
Printed in the United States of America.

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

Open SystemC, Open SystemC Initiative, OSCI, SystemC, and SystemC Initiative are trademarks or registered trademarks of Open SystemC Initiative, Inc. in the United States and other countries and are used with permission.

Trademarks: Trademarks and service marks of Cadence Design Systems, Inc. (Cadence) contained in this document are attributed to Cadence with the appropriate symbol. For queries regarding Cadence's trademarks, contact the corporate legal department at the address shown above or call 800.862.4522.

All other trademarks are the property of their respective holders.

Restricted Permission: This publication is protected by copyright law and international treaties and contains trade secrets and proprietary information owned by Cadence. Unauthorized reproduction or distribution of this publication, or any portion of it, may result in civil and criminal penalties. Except as specified in this permission statement, this publication may not be copied, reproduced, modified, published, uploaded, posted, transmitted, or distributed in any way, without prior written permission from Cadence. Unless otherwise agreed to by Cadence in writing, this statement grants Cadence customers permission to print one (1) hard copy of this publication subject to the following conditions:

1. The publication may be used only in accordance with a written agreement between Cadence and its customer.
2. The publication may not be modified in any way.
3. Any authorized copy of the publication or portion thereof must include all original copyright, trademark, and other proprietary notices and this permission statement.
4. The information contained in this document cannot be used in the development of like products or software, whether for internal or external use, and shall not be used for the benefit of any other party, whether or not for consideration.

Disclaimer: Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information. Cadence is committed to using respectful language in our code and communications. We are also active in the removal and replacement of inappropriate language from existing content. This product documentation may however contain material that is no longer considered appropriate but still reflects long-standing industry terminology. Such content will be addressed at a time when the related software can be updated without end-user impact.

Restricted Rights: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor.

Contents


| | |
|--|----|
| 1 | 4 |
| Overview | 4 |
| Audience | 4 |
| Prerequisites | 4 |
| 2 | 6 |
| Introduction to the OrCAD TCL Environment | 6 |
| 3 | 7 |
| Setting up the OrCAD TCL Environment | 7 |
| 4 | 8 |
| Basics of the TCL Language | 8 |
| Few Rules for Substitution in TCL | 9 |
| TCL Variable Types | 12 |
| 5 | 13 |
| Guidelines for New OrCAD TCL Applications | 13 |
| 6 | 15 |
| Creating OrCAD TCL Application | 15 |
| My First TCL Source File | 15 |
| Creating the Logic | 15 |
| Sourcing the TCL Source File | 16 |
| Accessing the TCL Source File from OrCAD Capture GUI | 21 |
| Generating the TCL source file package | 23 |
| Auto-loading the TCL application package | 25 |
| Encrypting the TCL solution | 27 |

Overview

Cadence provides TCL/Tk scripting capability in OrCAD® Capture to enable designers to create custom applications and scripts for their environment. With scripting and customization, designers can automate the manual processes and complete projects faster.

This guide covers the following sections to get you started with the TCL Application development in OrCAD Capture. For detailed information on the TCL APIs, see the *OrCAD TCL API Reference Guide*.

1. [Introduction to the OrCAD TCL Environment](#)
2. [Setting up the OrCAD TCL Environment](#)
3. [Guidelines to Create New TCL Application](#)
4. [Basics of the TCL Language](#)
5. [Creating OrCAD TCL Application](#)

 To better understand the basics of OrCAD TCL scripting, some experience in other scripting languages, such as Perl or Python, will be helpful.


Audience


The audience of the document is the first-time users of the TCL Language in the OrCAD TCL Environment.

Prerequisites

Before you start, ensure that the following software are installed on your system:

- Capture CIS 17.4

 OrCAD Capture Information Model document will be a good start to better understand the OrCAD TCL Environment. Therefore, it is highly recommended that before you start working on this document, you should have read the OrCAD Capture Information Model document.

 The terms OrCAD Capture and Capture have been used interchangeably in the document.

Introduction to the OrCAD TCL Environment

TCL, an interpreter-based scripting language, is used in OrCAD Capture to create custom features that do not exist natively, and automate the tedious manual tasks. The core TCL/TK functionality includes procedures and commands for data manipulation, control constructs, mathematical expressions, file I/O routines, system calls, registry handling, GUI designing, and many more. The always growing additional packages of TCL/TK are just making almost everything possible that can be done using any procedural language.

The integrated TCL interpreter in OrCAD Capture allows any commands from these TCL/TK packages to run seamlessly. On top of that, OrCAD Capture provides a rich set of its own TCL commands that gives immense power to the users to interact with both the Capture's User Interface and its database through the scripting interface.

The Capture's Command Window can be accessed (if not already available) by selecting *View – Toolbar – Command Window*.

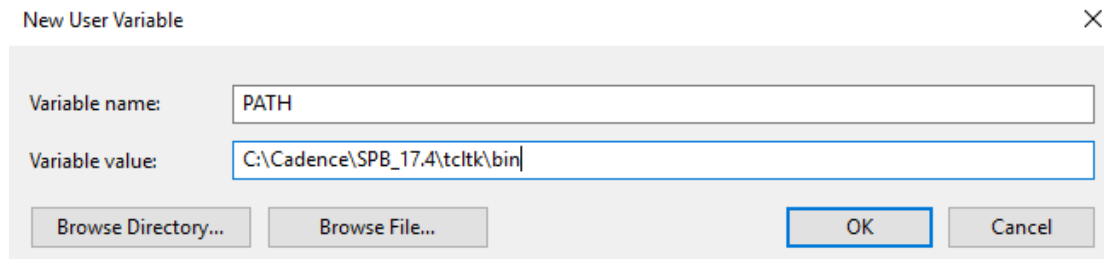
The Capture's TCL/TK framework works with the default TCL/TK installation that is a part of the standard Cadence Product installation. To use advanced TCL features or TK GUI extensions, you may need additional TCL packages to be installed in your machine. To set up the OrCAD TCL environment, see the [Setting up the OrCAD TCL Environment](#) chapter.

Setting up the OrCAD TCL Environment

As the TCL/Tk framework is installed with the Cadence Product standard installation, you do not require to install additional TCL packages on your machine. You can confirm TCL installation on your machine by verifying the following path: `<installation_hierarchy>tcltk\bin`.

To set up the TCL environment on your machine, do the following:

1. Add the TCL installation path in the `PATH` environment variable to access TCL from the Windows Command Prompt.



2. Open windows command prompt and enter `tclsh` to enable TCL shell in Windows Command Prompt.
Prompt changes to the `%` symbol.
3. Enter `info tclversion` in the command prompt to verify the TCL version installed in the machine.
The output is `8.6` for the 17.4-2019 release.
4. Enter `info nameofexecutable` to verify the executable path.

Once you have set up the OrCAD TCL environment in your machine, you need to understand and try some of the basics of the TCL language. See the [Basics of the TCL Language](#) chapter to learn TCL basics.

Basics of the TCL Language

In this chapter, you will learn some of the basic commands and concepts related to the TCL language to help you better understand the OrCAD TCL environment.

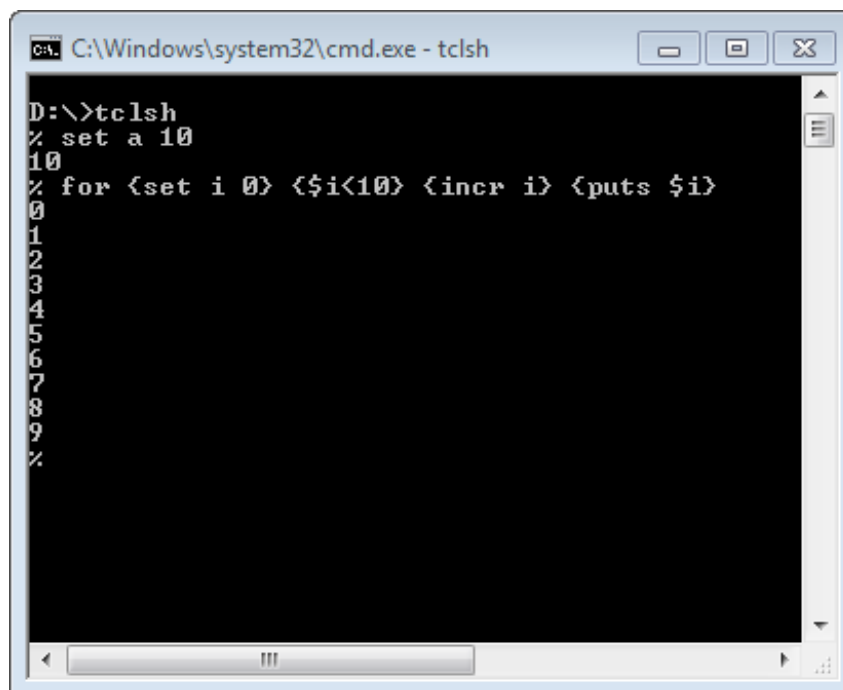
The TCL language does not have any special syntax design. Everything is based on some simple commands.

- `set` is a command that takes 2 parameters. For example,

```
set a 10
```

- A for loop in TCL is a command that takes four parameters. For example,

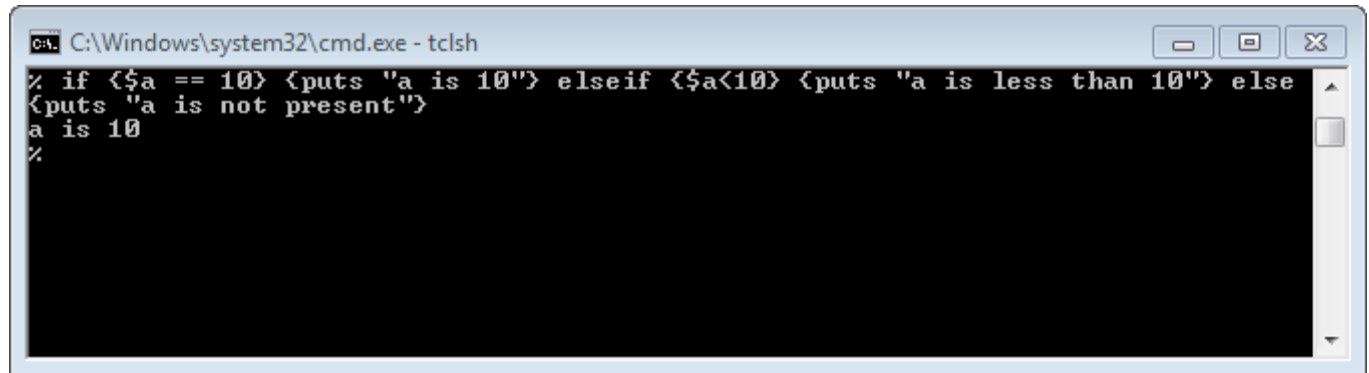
```
for {set i 0} {$i < 10} {incr i} {puts $i}
```



```
C:\Windows\system32\cmd.exe - tclsh
D:\>tclsh
% set a 10
10
% for {set i 0} {<$i<10} {incr i} {puts $i}
0
1
2
3
4
5
6
7
8
9
%
```

- If clause is a command. For example:

```
if{condition}{script} elseif {condition}{script} else {condition}{script}
```

```
C:\Windows\system32\cmd.exe - tclsh
% if { $a == 10 } { puts "a is 10" } elseif { $a < 10 } { puts "a is less than 10" } else
{ puts "a is not present" }
a is 10
%
```

- A comment is a command. For example:

```
#this is a comment that can be implemented as proc{#}args{}
```

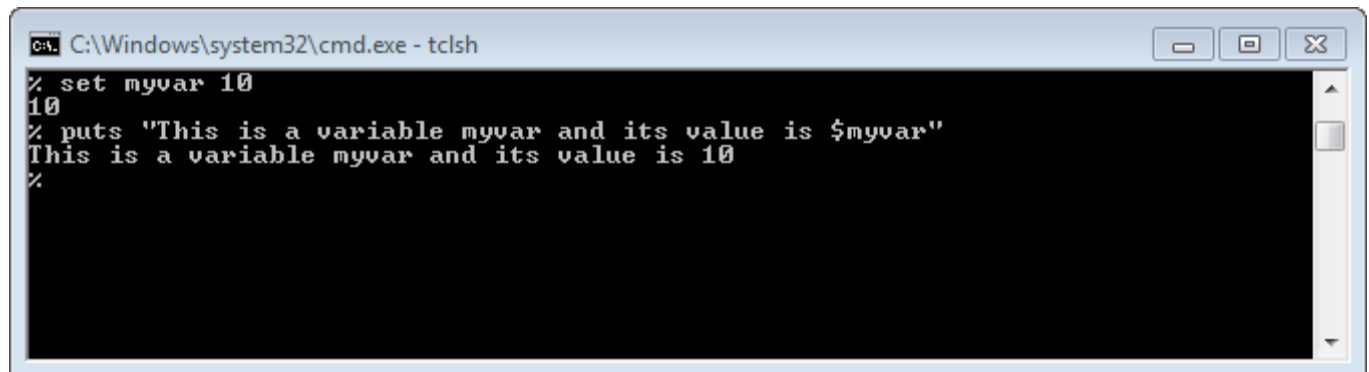
In the TCL language, all the data types that include integers and float, are considered strings by the TCL interpreter. For example, `set a 10.5` and `set a "10.5"` are same in TCL.

i Commands in TCL can be mapped to functions in the C language. TCL's Command name is the C's function name and parameters to a TCL command are parameters to a C function.

Few Rules for Substitution in TCL

- `$var` is always substituted by the value of variable `var`.

```
puts "This is a variable myvar and its value is $myvar"
```



```
C:\Windows\system32\cmd.exe - tclsh
% set myvar 10
10
% puts "This is a variable myvar and its value is $myvar"
This is a variable myvar and its value is 10
%
```

- Backslash (\) allows characters to be placed in a string that would otherwise have a special meaning. A \" is substituted by \".

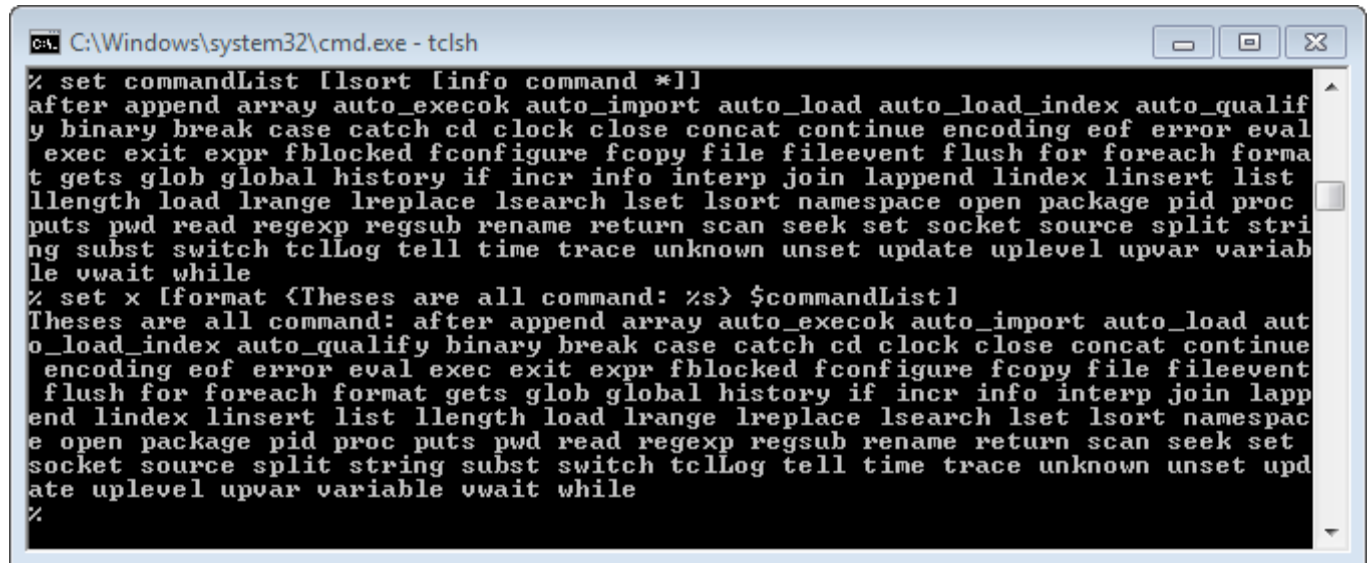
```
puts "This is \"Cadence\""
```

```
% puts "This is \"Cadence\""
This is "Cadence"
%
```

- [command parameter(s)] – command inside brackets ([]) is evaluated first and is substituted by its result.

```
set commandList [lsort [info commands*]]
```

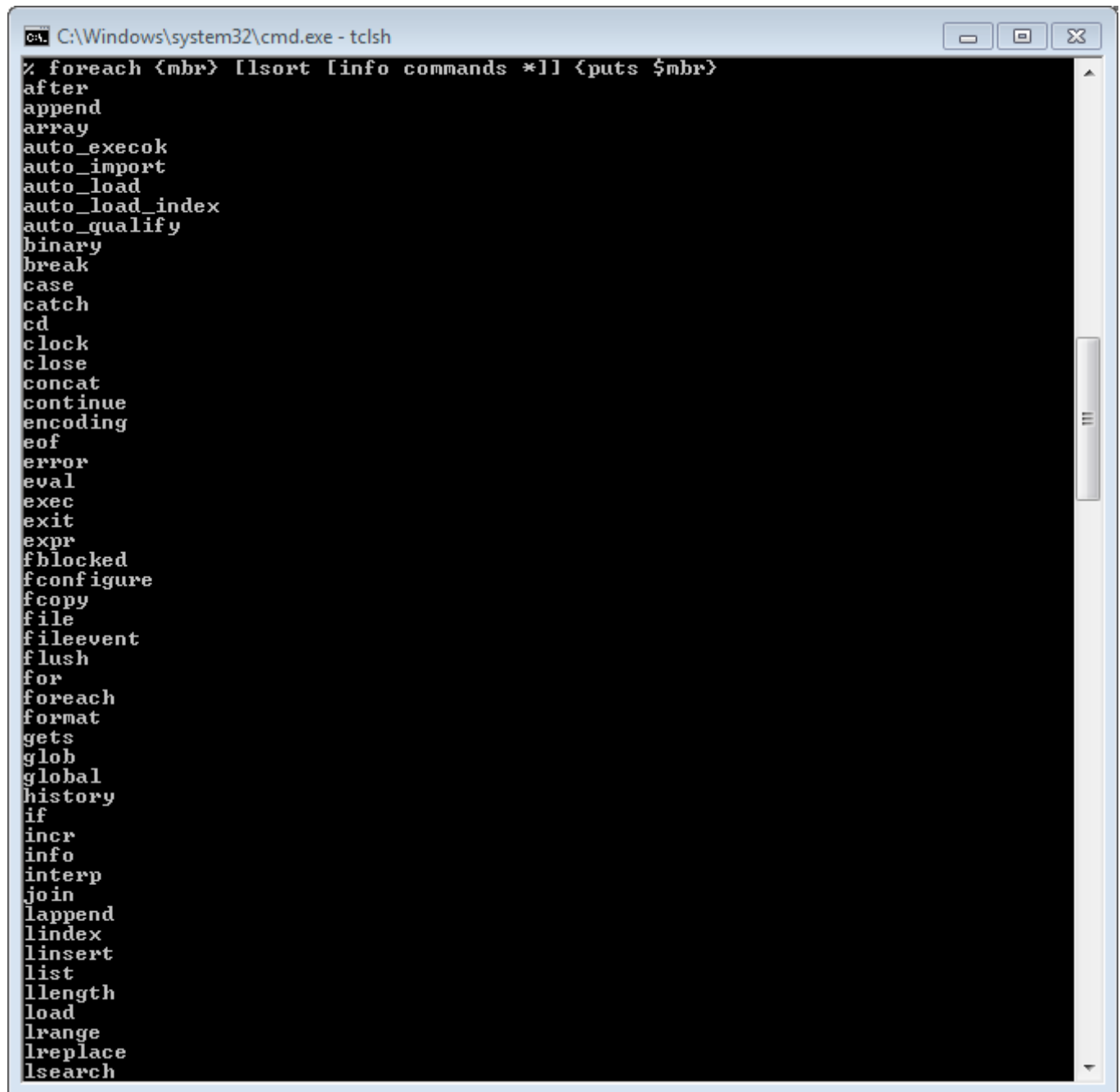
```
set x [format {There are all commands: %s} $commandList]
```



```
C:\Windows\system32\cmd.exe - tclsh
% set commandList [lsort [info command *]]
after append array auto_execok auto_import auto_load auto_load_index auto_qualify
binary break case catch cd clock close concat continue encoding eof error eval
exec exit expr fblocked fconfigure fcopy file fileevent flush for foreach format
gets glob global history if incr info interp join lappend lindex linsert list
llength load lrange lreplace lsearch lset lsort namespace open package pid proc
puts pwd read regexp regsub rename return scan seek set socket source split string
subst switch tclLog tell time trace unknown unset update uplevel upvar variable
vwait while
% set x [format {Theses are all command: %s} $commandList]
Theses are all command: after append array auto_execok auto_import auto_load auto_load_index
auto_qualify binary break case catch cd clock close concat continue encoding eof error eval
exec exit expr fblocked fconfigure fcopy file fileevent flush for foreach format gets glob
global history if incr info interp join lappend lindex linsert list llength load lrange
lreplace lsearch lset lsort namespace open package pid proc puts pwd read regexp regsub
rename return scan seek set socket source split string subst switch tclLog tell time trace
unknown unset update uplevel upvar variable vwait while
%
```

- {strings} - substitution is disabled inside parenthesis ({}), therefore {string} is substituted by string as is.

```
foreach {mbr} [lsort [info commands *]] {puts $mbr}
```



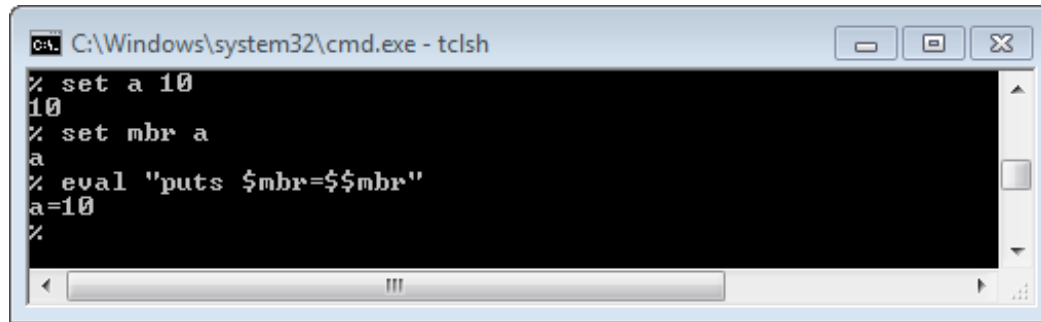
```
C:\Windows\system32\cmd.exe - tclsh
% foreach {mbr} [lsort [info commands *]] {puts $mbr}
after
append
array
auto_execok
auto_import
auto_load
auto_load_index
auto_qualify
binary
break
case
catch
cd
clock
close
concat
continue
encoding
eof
error
eval
exec
exit
expr
fblocked
fconfigure
fcopy
file
fileevent
flush
for
foreach
format
gets
glob
global
history
if
incr
info
interp
join
lappend
lindex
linsert
list
llength
load
lrange
lreplace
lsearch
```

- Using the `eval` command, second substitution gets done.

```
set a 10
```

```
set mbr a
```

```
eval "puts $mbr=$mbr"
```



```
C:\Windows\system32\cmd.exe - tclsh
% set a 10
10
% set mbr a
a
% eval "puts $mbr=$mbr"
a=10
%
```

TCL Variable Types

- **Simple Variable Type**

set x "This is a simple variable"

- **Associative Array:** used to store data

set matrix(3,5) 35.2

set matrix("name") "Cadence Design Systems"

parray matrix

- **Global Variables:** creates a global variable that can be accessed as `$::myvar`

global myvar

- **Namespace Variables:** limits a variable to a particular namespace

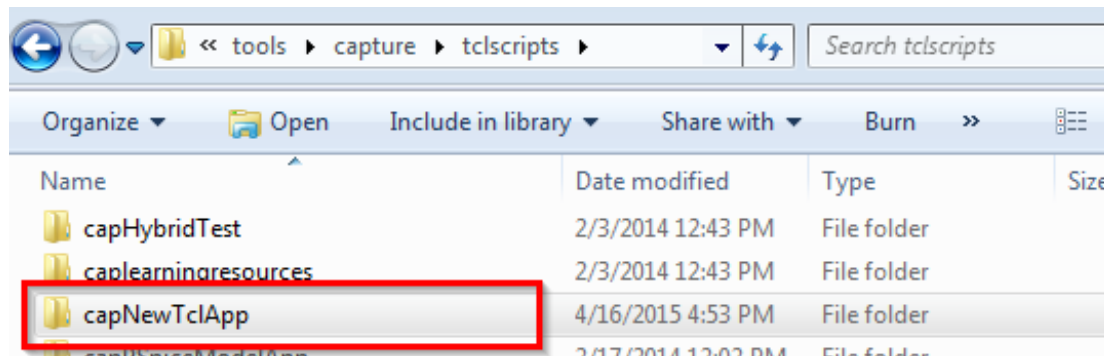
variable myvar

Guidelines for New OrCAD TCL Applications

For smooth and easy application development for OrCAD Capture using the TCL language, Cadence recommends the following guidelines to be followed:

1. Always add a new folder at `<Cadence_Installation_Root>\tools\capture\tclscripts` to create new TCL applications for OrCAD Capture, .

For example, add **capNewTclApp** folder for your first OrCAD TCL application at the mentioned location.



2. The new folder must contain all the custom TCL source files that are required for the TCL application, and a special TCL file, `pkgIndex.tcl`.

The content of the `pkgIndex.tcl` file maps the specific packages with the TCL source filenames. The content of the `pkgIndex.tcl` file is something similar to the following code:

```
package ifneeded capNewTclApp 1.0 [list source [file join $dir myFisrtApp.tcl]]
```

OrCAD Capture automatically sources the `pkgIndex.tcl` files present inside all the sub-folders of the **tclscripts** folder. This ensures that the custom TCL packages automatically become available in the current OrCAD Capture session.


3. If you want to automatically load the custom menus related to the new TCL solution during OrCAD Capture launch, place an autoloascript at

`<Cadence_Installation_Root>\tools\capture\tclscripts\capAutoLoad.`

The TCL callback procedures for the sub-menus calls appropriate TCL procedures from the custom packages placed inside the newly created sub-folder inside the **tclscripts** folder. This

is done by calling the `package require <package name>` command before calling the corresponding TCL procedures.

4. If possible, create sub-menus under the *Accessories* menu to provide an entry-point for the custom TCL script solution.

 For successful Capture launch, avoid adding a TCL script inside the **capAutoLoad** folder that does some heavy computation and impacts performance.

Creating OrCAD TCL Application

In the chapter [Basics of the TCL Language](#), we got accustomed to some basics related to the TCL commands. These basics will help us understand some of the advanced concepts that we will use to create our first OrCAD TCL application in this chapter. We will cover the following topics that are required to create an OrCAD TCL application:

1. My First TCL source file
 - a. Creating a logic that solves our problem
 - b. Running the script from Command Window by sourcing mechanism
 - c. Making the TCL source file accessible from the OrCAD Capture GUI
2. Generating the TCL source file package
3. Adding a new autoload script for the new TCL application
4. Encrypting the TCL package to protect intellectual property rights

My First TCL Source File

You can create your first TCL source file using Notepad, NotePad++, Sublime Text, or any other text editor that you are comfortable with. Remember to save the source file with the `.tcl` file extension.

In chapter 2, you have created the **capNewTclApp** folder at `<installation>\tools\capture\tclscripts`, now you will create your first TCL source file in this folder. Let's name the source file `myFirstApp.tcl`.

Creating the Logic

In the following source file, there are some TCL commands to rotate a single part's placed instance to 180 degrees. You can copy and paste the following code and save it in `myFirstApp.tcl`.

myFirstApp.tcl

```
namespace eval ::capRotate {  
    proc capRotatePart {} {  
        set lobj [GetSelectedObjects]  
        set ltype [$lobj GetObjectType]  
        if { $ltype == $::DboBaseObject_PLACED_INSTANCE && [llength $lobj] == 1 } {  
            Rotate  
            Rotate  
        }  
    }  
}
```

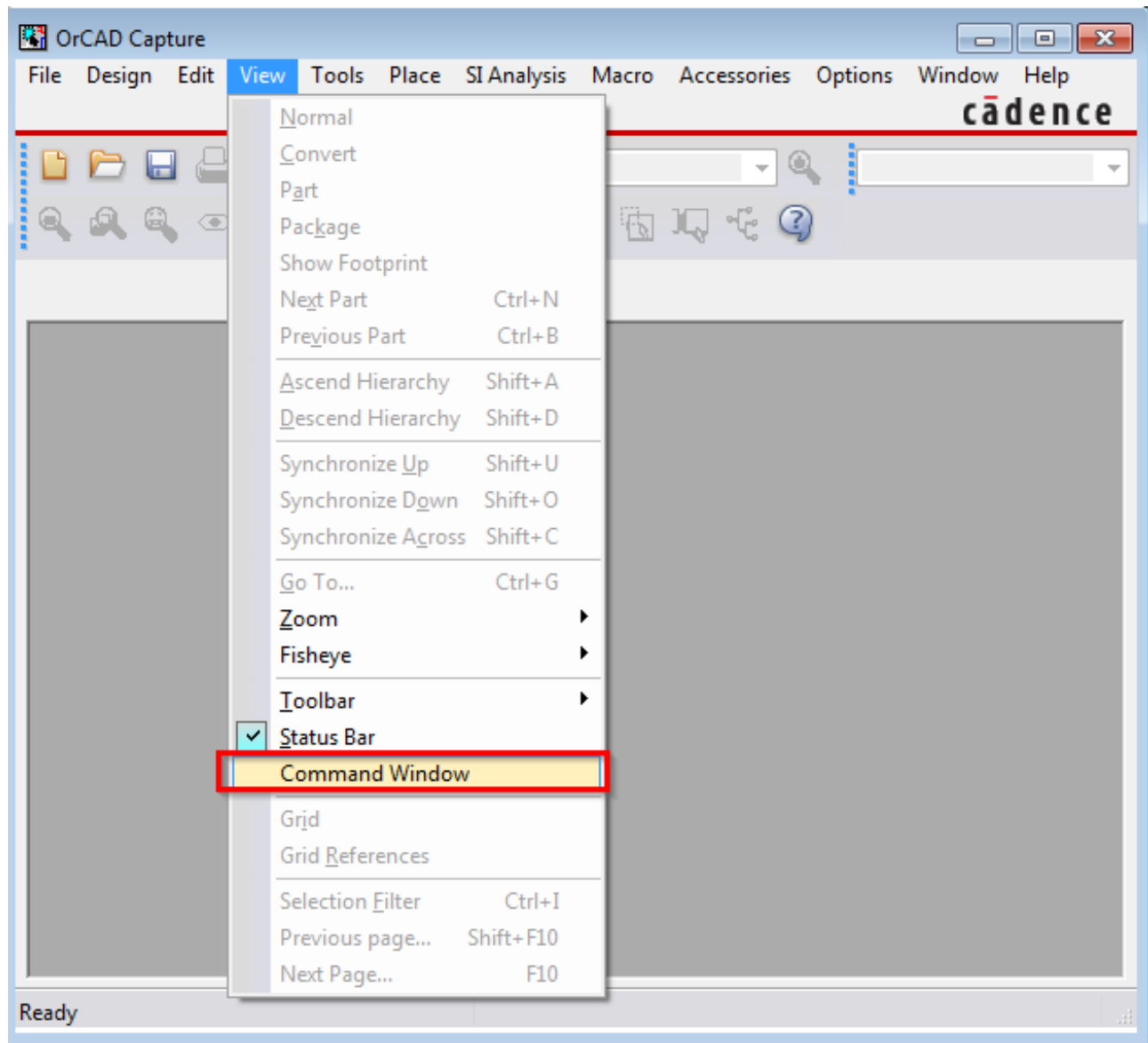
Sourcing the TCL Source File

Now let's source your First TCL source file in OrCAD Capture and execute it to verify the desired results.

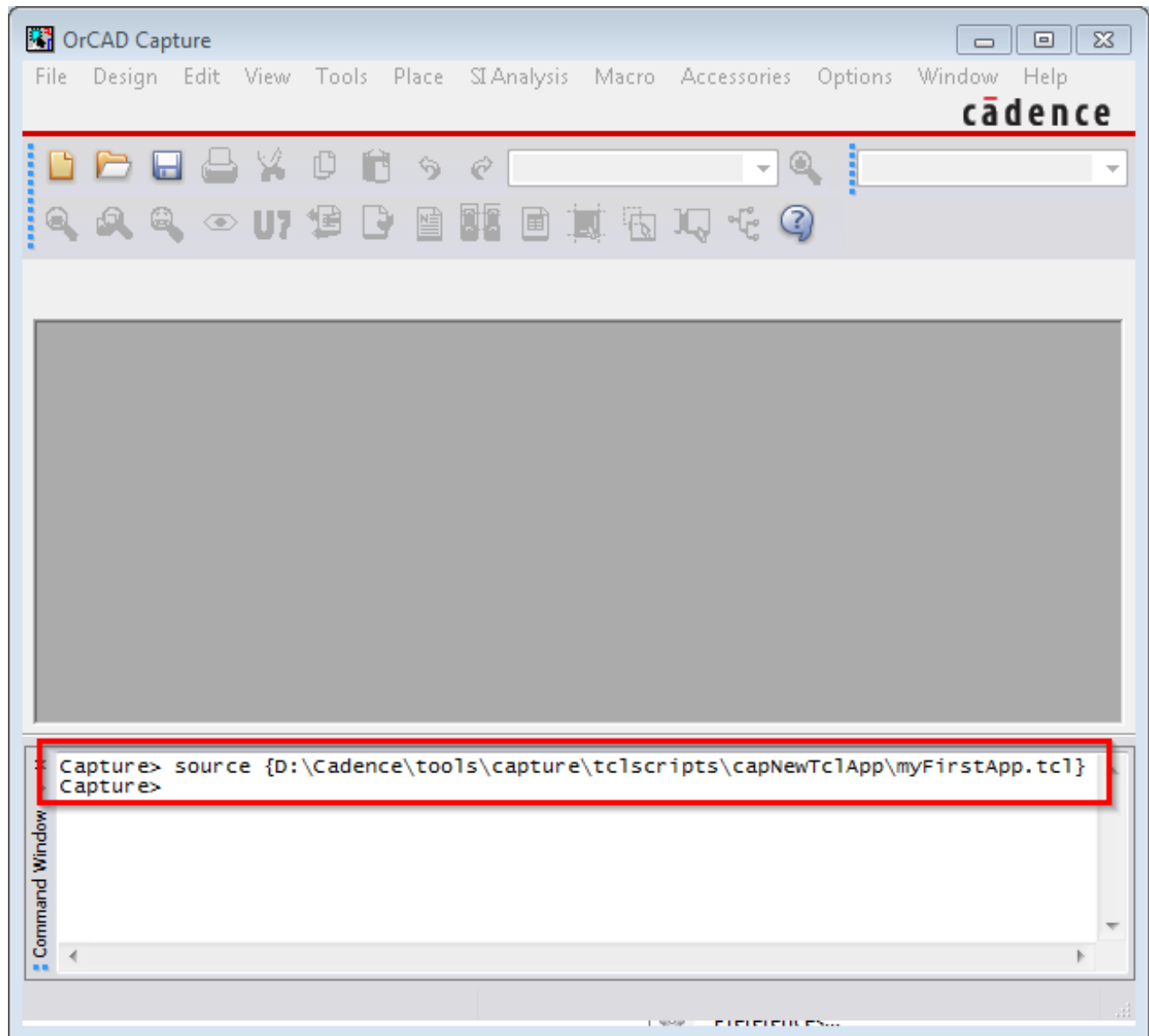
 You can open any OrCAD Capture design to verify the output, but in this section **FULLADD.DSN** is used. The **FULLADD.DSN** file is located at <Cadence_Installation>\tools\capture\samples.

To source a TCL source file in OrCAD Capture, do the following:

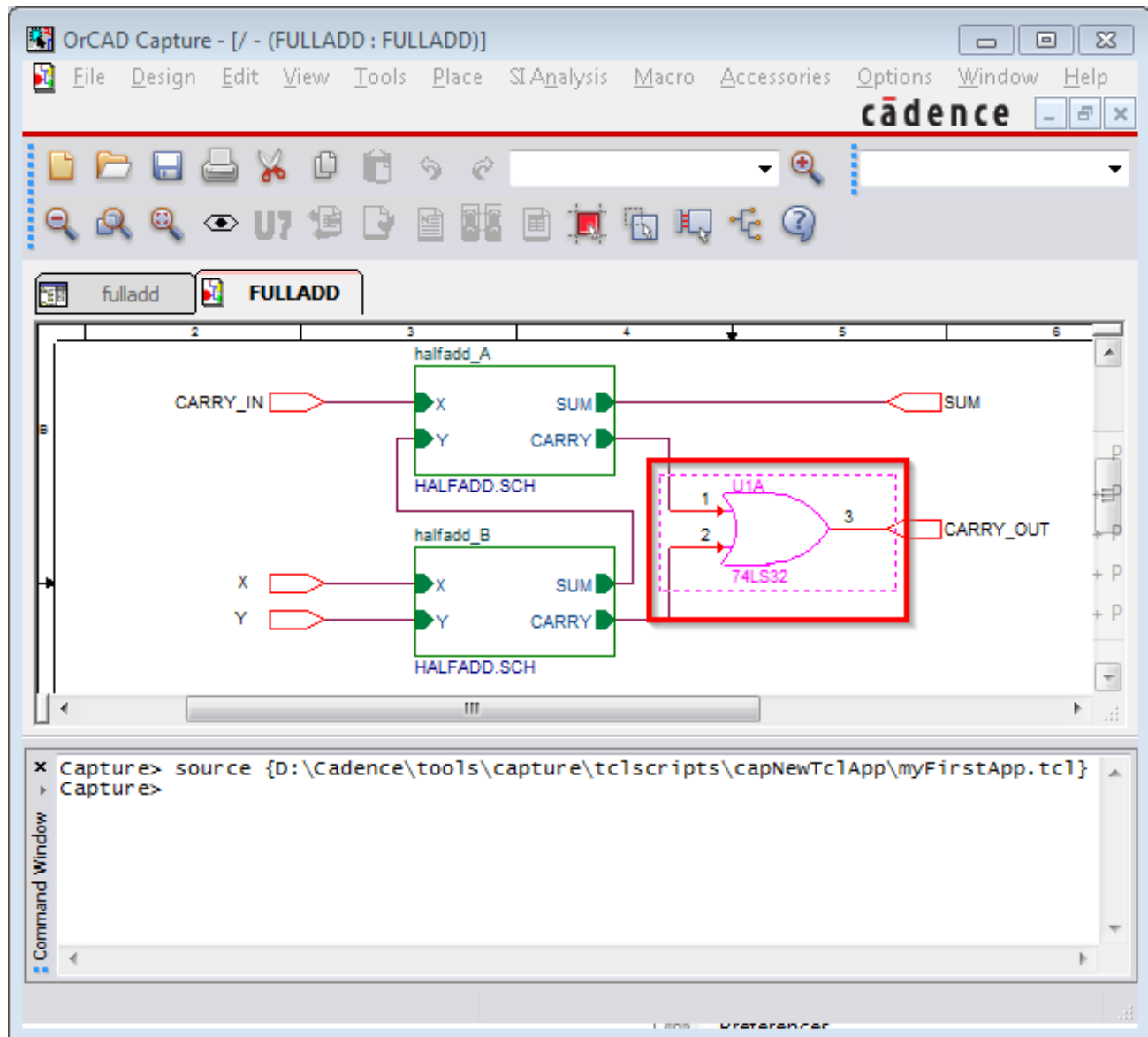
1. Select *View – Command Window* to open the Command Window in OrCAD Capture.



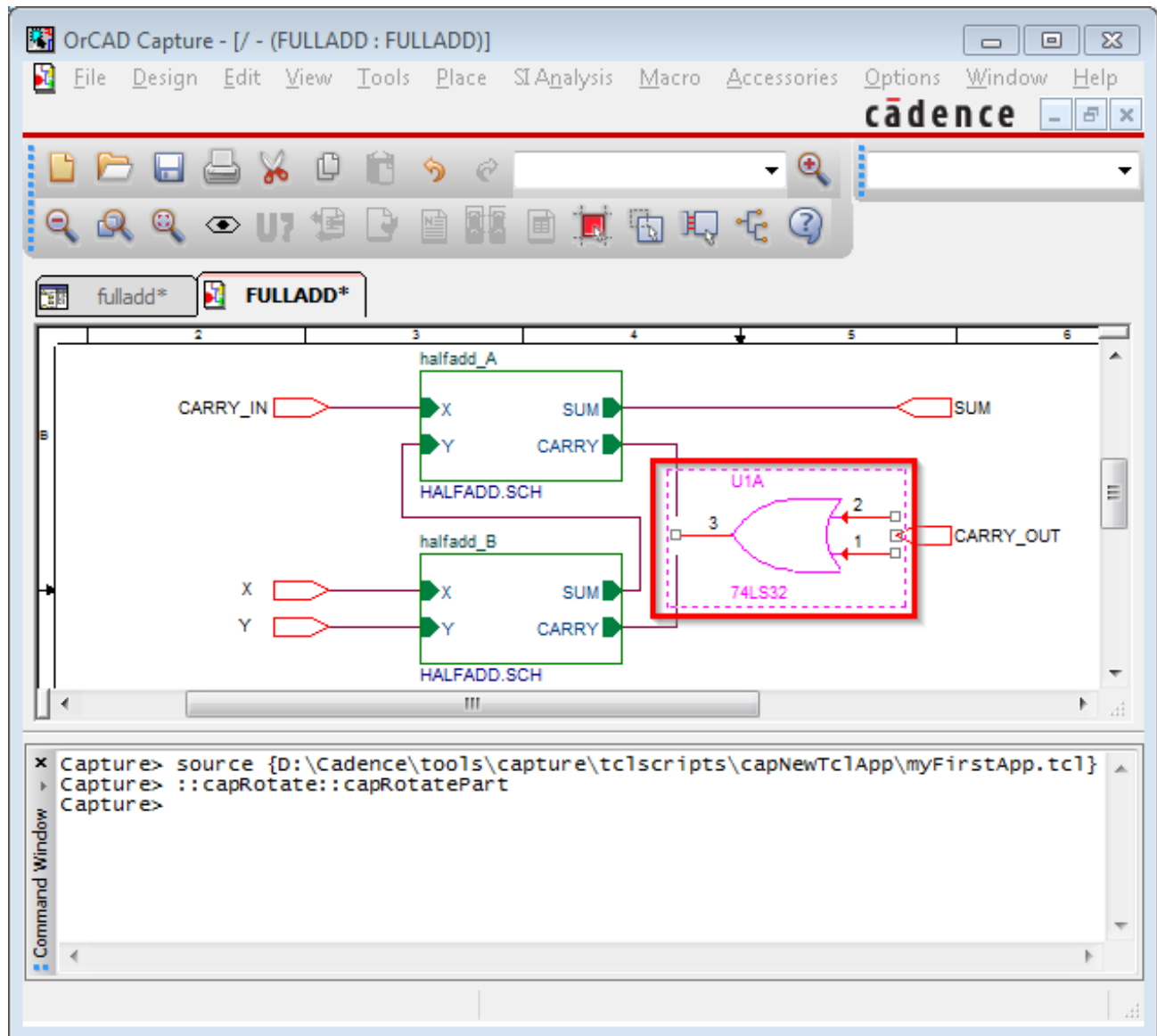
2. Type `source {TCL source file absolute path}` in the Command Window and press `Enter`.
For example, the **capNewTclApp** folder is located at
`<Cadence_Installation>\tools\capture\tclscripts` and we have saved **myFirstApp.tcl**
in **capNewTclApp**. Therefore, the source command can be typed as `source`
`{<Cadence_Installation>\tools\capture\tclscripts\capNewTclApp\myFirstApp.tcl}`.



3. Open the **FULLADDER.DSN** file in OrCAD Capture and select any part's placed instance, such as the **U1A OR Gate** in the **FULLADD** schematic page.

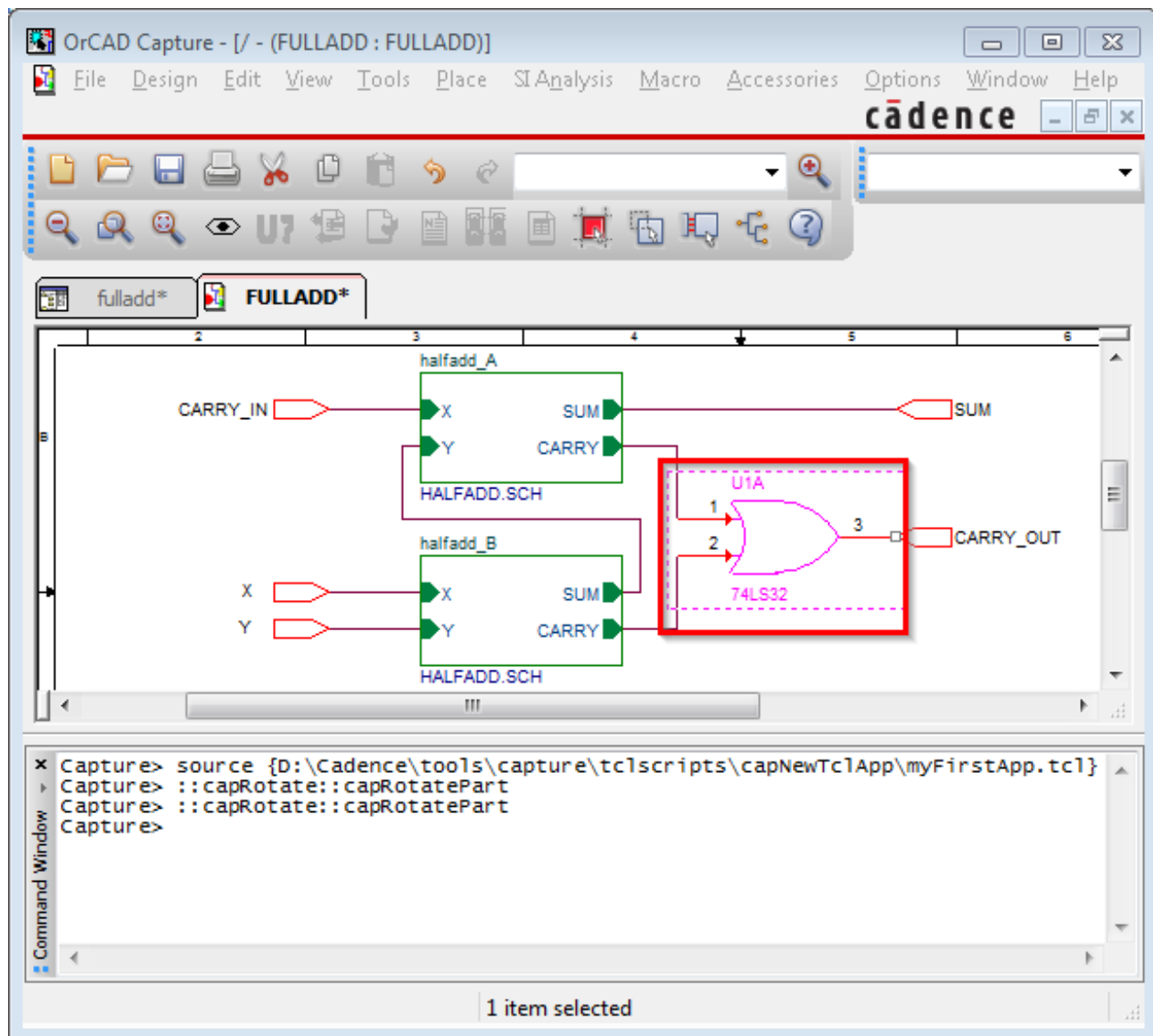


4. Type `::capRotate::capRotatePart` in the Command Window and press `Enter`.



5. Again type `::capRotate::capRotatePart` in the Command Window.

- ✓ You can also access the last executed command in the Command Window using the up (↑) and down (↓) arrow of the keyboard.



You have successfully sourced the TCL source file in the OrCAD Capture current session and executed the file to rotate the selected part's placed instance by 180 degree twice. In the next section, you will learn to add GUI option in OrCAD Capture for your TCL solution.

Accessing the TCL Source File from OrCAD Capture GUI

You can make your TCL solution more user-friendly by adding a menu option for your TCL solution in the OrCAD GUI.

As per requirements, menu option(s) for TCL solution(s) can be added in Schematic Editor or Project Manager. In this section, we will add the TCL solution in the Schematic Editor pop-up menu as it is easier to select any option from the pop-up menu when a part is selected.

You can copy and paste the following TCL code in the `myFirstApp.tcl` file and save it. Do not forget to delete the previous TCL code before adding the following one.

myFirstApp.tcl

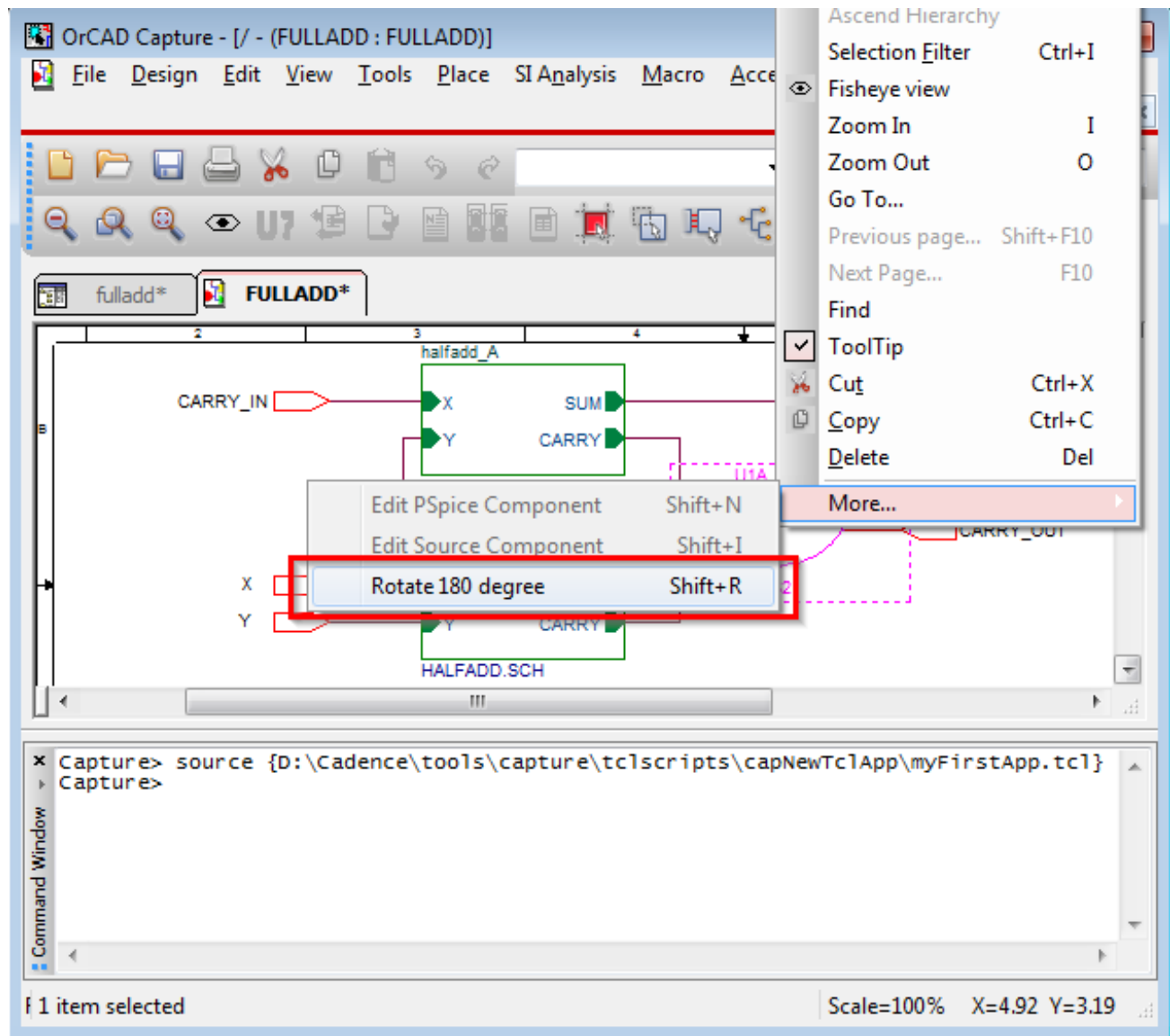
```
namespace eval ::capRotate {
    namespace export capRotatePart
    namespace export capRotatePartEnabler
    RegisterAction "Rotate 180 degree" "::capRotate::capRotatePartEnabler" "Shift+R"
    "::capRotate::capRotatePart" "Schematic"
}
proc ::capRotate::capRotatePartEnabler {} {
    set lEnableRotate 0
    set lSelObjs [GetSelectedObjects]
    set lObjType [DboBaseObject_GetObjectType $lSelObjs]
    if { ($lObjType == $::DboBaseObject_PLACED_INSTANCE) && ([llength $lSelObjs] == 1) } {

        set lEnableRotate 1
    }
    return $lEnableRotate
}
proc ::capRotate::capRotatePart {} {
    set lobj [GetSelectedObjects]
    set ltype [$lobj GetObjectType]
    if { $ltype == $::DboBaseObject_PLACED_INSTANCE && [llength $lobj] == 1 } {

        Rotate
        Rotate
    }
}
```

Once you have updated the `myFirstApp.tcl` file, open the **FULLADD.DSN** file in OrCAD Capture and do the following steps:

1. Source the TCL source file in the Command Window using the `source` command. For example, `source {<Cadence_Installation>\tools\capture\tclscripts\capNewTclApp\myFirstApp.tcl}`.
2. Select the U1A OR Gate in the FULLADD schematic page and right-click.
3. In the pop-up menu, select *More - Rotate 180 degree*.
You can notice that the selected part, that is, U1A OR Gate, gets rotated by 180 degrees.



4. Again right-click and select *More - Rotate 180 degree* in the pop-up menu.
You can again notice that the part rotated by 180 degrees.

After adding a menu option in OrCAD Capture GUI you can notice that you still need to source the TCL source file, that is `myFirstApp.tcl`, from the Command Window. Let's generate a package for the `myFirstApp.tcl` file to remove the sourcing step that needs to be followed when the TCL source file is accessed.

Generating the TCL source file package

Sometimes running the source commands on the Command Window gets tedious and

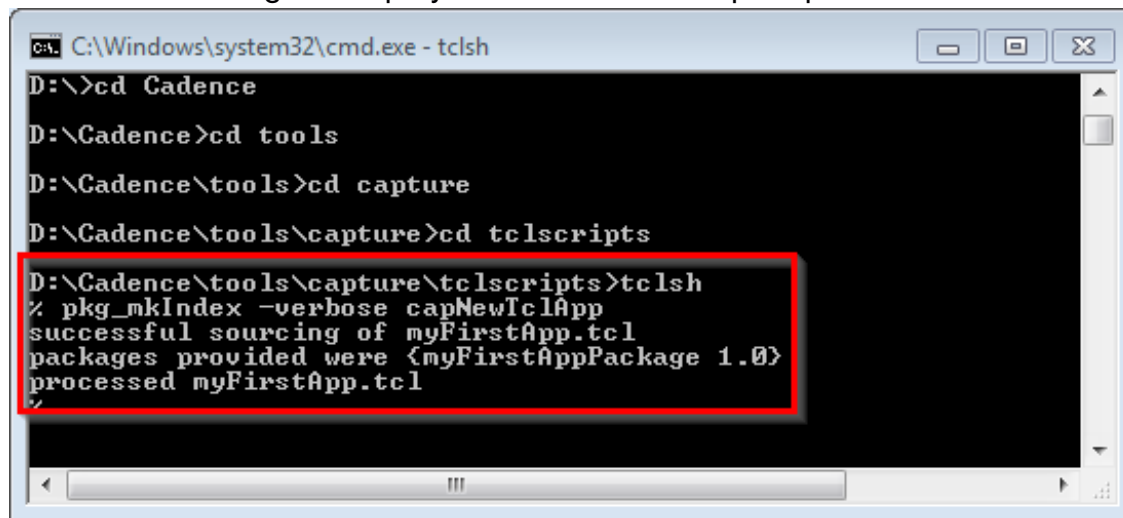
cumbersome if you want to share your automation solutions with other designers. You can make the process more smooth and easy by making a package of the TCL source file that you want to share with other designers. Once you have made the package, you can easily access your TCL solution using the `package require {Package Name}` command from the Command Window.

Before you can generate a package for your TCL source file, add the following code at the top of your existing code in the `myFirstApp.tcl` TCL file.

```
package provide myFirstAppPackage 1.0
```

To generate a package for your TCL source file, that is, `myFirstApp.tcl`, do the following steps:

1. Open the Windows Command Prompt and change the directory to the **capNewTclApp** folder.
2. Type the `tclsh` command.
The prompt changes to percent sign (%).
3. Type the `pkg_mkIndex -verbose capNewTclApp` command and press Enter.
A success message is displayed in the command prompt.



```
C:\Windows\system32\cmd.exe - tclsh
D:\>cd Cadence
D:\Cadence>cd tools
D:\Cadence\tools>cd capture
D:\Cadence\tools\capture>cd tclscripts
D:\Cadence\tools\capture\tclscripts>tclsh
% pkg_mkIndex -verbose capNewTclApp
successful sourcing of myFirstApp.tcl
packages provided were {myFirstAppPackage 1.0}
processed myFirstApp.tcl
```

You can also browse the **capNewTclApp** folder and see that the `pkgIndex.tcl` file is present in the folder.

4. Open the `pkgIndex.tcl` file and view the content inside it.

pkgIndex.tcl

```
# Tcl package index file, version 1.1
# This file is generated by the "pkg_mkIndex" command
# and sourced either when an application starts up or
# by a "package unknown" script. It invokes the
# "package ifneeded" command to set up package-related
# information so that packages will be loaded automatically
# in response to "package require" commands. When this
# script is sourced, the variable $dir must contain the
# full path name of this file's directory.

package ifneeded myFirstAppPackage 1.0 [list source [file join $dir myFirstApp.tcl]]
```

You can see that the `package ifneeded` command checks the package name and version, and sources `myFirstApp.tcl` from a directory path, that is, the absolute path of the TCL source file.

Once you have generated the package of the `myFirstApp.tcl` file, you can access the package from the OrCAD Capture Command Window using the `package require myFirstAppPackage` command. You can now access the package and use the TCL solution in OrCAD Capture GUI.

Auto-loading the TCL application package

You have created your first TCL application package and other designers are quite happy with your TCL solution. But you do not want to type the `package require myFirstAppPackage` command every time you use your TCL solution. Therefore, the best way to do that is to automatically load your TCL solution into OrCAD Capture during launch time.

To get the `myFirstAppPackage` package automatically loaded into OrCAD Capture, you need to add an initialization TCL file in the **capAutoLoad** folder, which is located at `<Cadence_Installation>\tools\capture\tclscripts`.


To add an initialization TCL file in the **capAutoLoad** folder, do the following steps:

1. Create a new TCL file in the **capAutoLoad** folder and name it `capNewTclAppInit.tcl`.
2. Add the following code in `capNewTclAppInit.tcl` and save it.

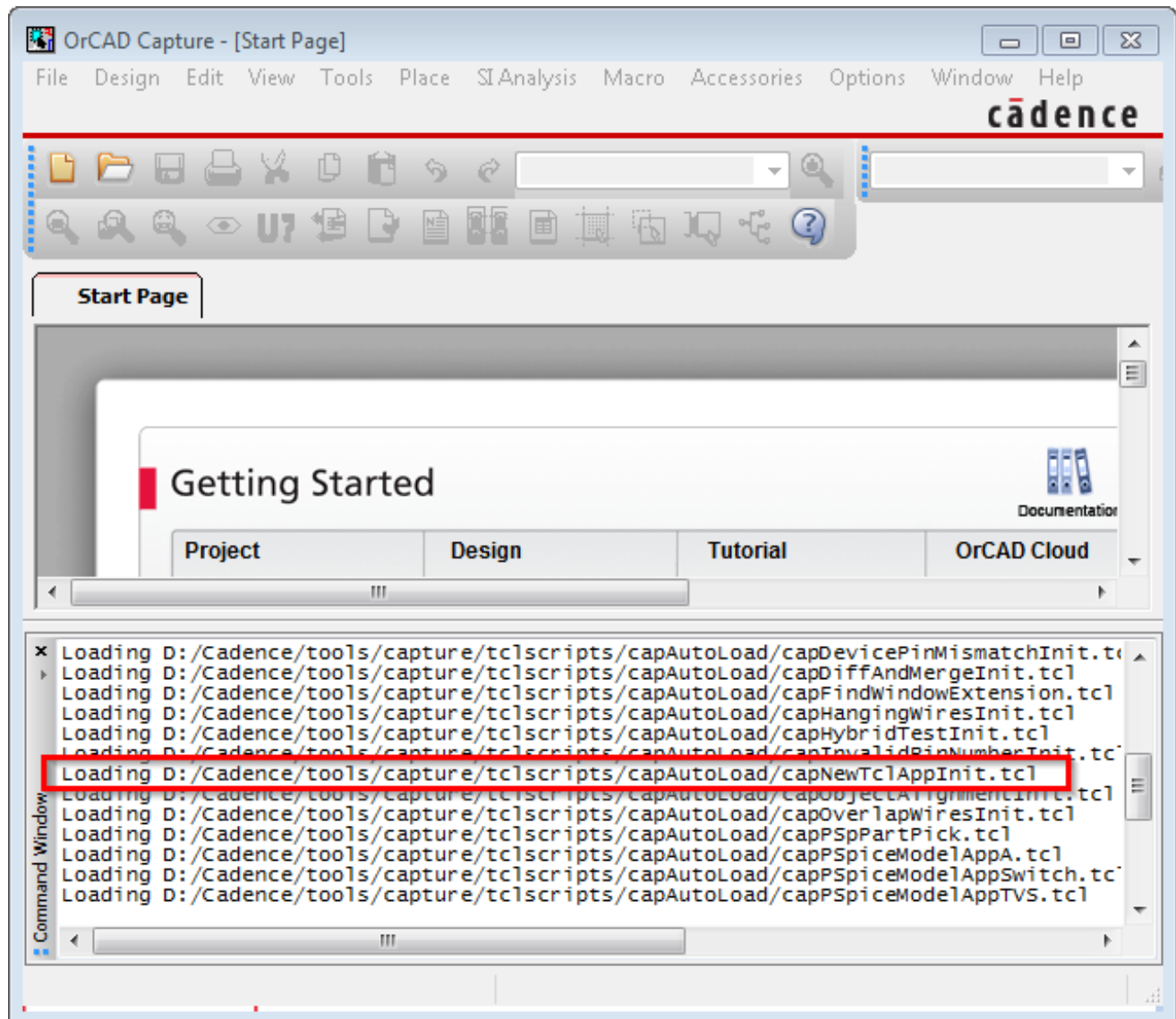
capNewTclAppInit.tcl

```
#####  
#  WARRANTY: NONE. THIS PROGRAM WAS WRITTEN AS "SHAREWARE" AND IS AVAILABLE AS IS  
#          AND MAY NOT WORK AS ADVERTISED IN ALL ENVIRONMENTS. THERE IS NO  
#          SUPPORT FOR THIS PROGRAM  
#          NOTE: YOU ARE STRONGLY ADVISED TO BACKUP YOUR DESIGN  
#          BEFORE RUNNING THIS PROGRAM  
#  TCL file: capNewTclAppInit.tcl  
#####  
  
proc capMyFirstApp_capLaunch { args } {  
    if { [catch {package require myFirstAppPackage}] } {  
        }  
    }  
capMyFirstApp_capLaunch
```

3. Open OrCAD Capture.

 If you have an OrCAD Capture session already opened in your machine. Close it and launch a new OrCAD Capture session.

You can notice that the `capNewTclAppInit.tcl` file, which is your TCL solution's initialization file, is loaded in OrCAD Capture during initialization.



You can now use your first OrCAD TCL application that is GUI-based and user-friendly, and automatically gets loaded in OrCAD Capture at initialization. Now, there is no need to type any command in the Command Window to access the package, your TCL solution is available in the Schematic Editor pop-up menu after OrCAD Capture launch.

Encrypting the TCL solution

You have created your first OrCAD TCL application and you are really proud of it. Everyone in the team is using your solution and completing the tedious manual task automatically in a short span of time. But there is always a chance that the solution gets shared with a person who does not belong to your team and you do not want your logic to be read by that person.

There could be a situation where you just want the encrypted binary solution to be available to everyone in your team. To do this, you just need to type some commands in the Command Window and your TCL solution gets converted into an encrypted binary file.

⚠ After you generate an encrypted binary file, save the original TCL file in a secure location for later use. If you discard the original file, there is no way to recover it from the encrypted binary.

⚠ To encrypt the TCL script, the `orcad::encrypt` command is not supported in the 17.2-2016 release and onwards. The TCL file encrypted in 16.6 release using the `orcad::encrypt` command are loadable in 17.2-2016 (and onwards) using the `orcad::load` command.

Do the following steps to install AvisaState TCL Development Kit and generate the .tbc file:

1. Install ActiveTcl (from ActiveState) that supports TCL 8.6 or lower on your machine.
2. Run the following command on the command prompt after successful installation of the DevKit: `tclcompiler86.exe <TCL File Path>`
The output file is encrypted and has .tbc extension.
3. Browse the **capNewTCLApp** folder and open **myFirstApp.tbc** in any text editor. You can notice that the content of the encrypted binary file is really difficult to read and understand.
4. Open **pkgIndex.tcl** and change **myFirstApp.tcl** to **myFirstApp.tbc** in the TCL source code and save it.

pkgIndex.tcl

```
# Tcl package index file, version 1.1
# This file is generated by the "pkg_mkIndex" command
# and sourced either when an application starts up or
# by a "package unknown" script. It invokes the
# "package ifneeded" command to set up package-related
# information so that packages will be loaded automatically
# in response to "package require" commands. When this
# script is sourced, the variable $dir must contain the
# full path name of this file's directory.

package ifneeded myFirstAppPackage 1.0 [list source [file join $dir myFirstApp.tbc]]
```

5. Once you have generated **myFirstApp.tbc** and made the changes in **pkgIndex.tcl**. Relaunch OrCAD Capture to use your TCL solution.

Once you complete the steps, your TCL solution is encrypted and safe. After encryption, you can share the TCL script with your team members and other designers without worrying about the logic being stolen.

Using this guide, you have created your First OrCAD TCL Application. If you want to learn more about OrCAD TCL application development, refer to the following documents:

- OrCAD Capture Information Model
- OrCAD Capture Sample TCL Scripts