Title: Tcl/Tk Commands

Product: OrCAD – Capture, CIS

Summary: Collection of useful TCL/Tk Commands

Author/Date: Beate Wilke / 23.01.2013

How to use this document:

All listed Tcl commands and codes are written in Courier new font to divide them from other text.

All attached Tcl code examples are stored with additional extension .txt to open them from PDF document as text file. To use Tcl scripts, this extension need to be removed.
# Table of Contents

1 Introduction .......................................................................................... 3
   1.1 The Tcl Command Window ......................................................... 3
   1.2 To display the Command window ............................................. 3
   1.3 Clear the Command window ...................................................... 3
   1.4 Store TCL Commands in local environment ............................... 3
2 Cadence TCL Utilities ........................................................................... 4
3 OrCAD Marketplace Apps ...................................................................... 5
4 TCL/Tk Commands for Capture ............................................................. 6
   4.1 Hide rectangle from Hierarchical Block ..................................... 6
   4.2 Property Visibility ........................................................................ 7
   4.3 Get the MAC Adress ................................................................. 7
   4.4 ClearTempPackage ..................................................................... 8
   4.5 Create Library Part Using TCL .................................................. 9
   4.6 How to Update Variant Information in the Titleblock in variant view? .................................. 11
   4.7 Customizing Menus and Toolbars ............................................ 13
   4.8 Shortcuts .................................................................................... 14
   4.9 Customize page .......................................................................... 15
5 TCL/Tk Commands for Capture CIS ......................................................... 16
   5.1 Make multible values in database field visible in CIS .................. 16
   5.2 EOL Management ..................................................................... 17
   5.3 Save CIS Explorer Viewed table to Text-File ............................. 19
   5.4 How to open Part Type tree in expanded mode ......................... 20
   5.5 CIS TCL data access sample ..................................................... 21
6 FlowCAD App ..................................................................................... 22
   6.1 FloWare Tortoise SVN Integration ........................................... 22
   6.2 Test point Checker .................................................................... 22
7 Appendix ............................................................................................. 23
   7.1 How to Update Variant Information in the Titleblock in variant view? .................................. 23
   7.2 Property Visibility ..................................................................... 26
   7.3 Customizing Menus and Toolbars ............................................ 28
   7.4 EOL Management ..................................................................... 32
1 Introduction

Tcl (Tool Command Language) is designed as a glue language for integrating applications and is a popular scripting language, embedded in various EDA (Electronic Design Automation) tools.

1.1 The Tcl Command Window

The Capture environment includes a Command window. You use this window to execute Tcl commands. Also, when you perform an operation (function) in Capture, the associated command is registered with the Tcl interpreter and the command is reported in the Command window.

1.2 To display the Command window

Go to View > Toolbar and enable Command Window. The Command Window menu item is a toggle menu. This implies that if you select the menu again, you will disable and hide the window.

To get more information about Tcl Scripts please read:
- Under $CDSROOT/tools/capture/tclscripts you find the OrCAD_Capture_TclTk_Extensions.pdf. This file includes all details about Capture Tcl commands.

1.3 Clear the Command window

In the Command window type in cls and press enter.

1.4 Store TCL Commands in local environment

Cadence Tcl scripts are stored in Cadence installation structure. You can place your own TCL file in your local environment. Please see FlowCAD_AN_Install_SiteAndHome.pdf for more details. You can download this AN from FlowCAD Homepage or ask FlowCAD Hotline.
2 Cadence TCL Utilities

Open any project and select **Accessories > Cadence Tcl/Tk Utilities > Utilities.** You get this window.

![Tcl/Tk Applications Dashboard](image)

This is a list of implemented Tcl utilities. All utilities are based on scripts from `$CDS_ROOT\tools\capture\tclscripts`. This folder contains a lot more Tcl examples than utilities menu shows.

For more information about Cadence Tcl utilities look at Tcl documentation **OrCAD_Capture_TclTk_Extensions.pdf**. You find this file under `$CDS_ROOT\tools\capture\tclscripts`. Some very useful Tcl utilities have a short introduction in this AN.
3 OrCAD Marketplace Apps

On Capture Start Page you find a Link to OrCAD Marketplace at the lower right corner.

Some freeware Apps from OrCAD Marketplace need additional Tcl files to get them running. The files need to be stored in user environment. User environment is defined by system variable HOME. If the desired folders are not available, please create them.

Here are the files:

File - DefineMarketplaceTCLAppRoot.tcl
Desired folder - %HOME%\cdssetup\OrCAD_Capture\tclscripts\capAutoLoad

File - LoadTCLApp.tcl
Desired folder - %Home%\Marketplace_TCLApps\TCLApploader\TCL.
4 TCL/Tk Commands for Capture

4.1 Hide rectangle from Hierarchical Block

Create any hierarchical block with pins.
Select hierarchical block and use right mouse button > Edit Part.
Edit the Hierarchical block – place e.g. a triangle of lines as shown below.

Close the Part mode and select Yes to update. The design should now look like this:

Open the Tcl Command Window, select the Hierarchical Blocks and execute the following command in the Tcl Command window:

```
DboBaseObject_SetBitmask [GetSelectedObjects] $::DboBaseObject_MASK_BOXOFF
```

This hides the block border. To undo this use:

```
DboBaseObject_UnsetBitmask [GetSelectedObjects] $::DboBaseObject_MASK_BOXOFF
```
4.2 Property Visibility

Run the attached script in Capture’s command window and it will set the visibility of property values to Do Not Display for all parts present in the design.

You need to source the Global.tcl code (set value of Property Value to Do not Display) and run it.

4. Source {Path to the Global.tcl}. If file exist at D:\TCL_Scripts, then command will be like:

> source {D:\TCL_Scripts\Global.tcl}

The visibility of any property can be changed to:

- Do Not Display = 0
- Value Only = 1
- Name and Value = 2
- Name Only = 3
- Both Value Exists = 4

The same file can be modified in a text editor to change the visibility of other displayed property present in the design. An invisible property can’t be changed to other display status using this script. The attached movie file shows the steps to do it.

Link to TCL code in text file - Global.tcl

4.3 Get the MAC Adress

From SPB 16.6 with hotfix 19 you can get MAC ID using following TCL command.

GetMACAddresses
4.4 ClearTempPackage

Solution to prevent Capture Crash during Cleanup Cache.

The link between the Design Cache and the part placed on the schematic is through the source library name.

Problem comes, when the cache data is correct, but the part and cache entry source library names are different. In this case, when the part needs data, cache does not return any match due to the different source library names.

The TCL command ClearTempPackage makes the source library name, present in cache, same as the part library name and also does cleaning of the Temporary cache object which should not be present.

However, it will work only when the source library name of a part points to the correct library path, since cache entry is updated with this value. It will not work, when this entry is not correct.

How to use it:
- Select DSN file in Project View
- Change to Command Window
- Type in ClearTempPackage

In Version 16.60 s040, and older version, you will get the error Message:

[ 1]RuntimeError No PM Window Containing DSN

You can ignore it. It will be fixed in one of the next hotfix.
4.5 Create Library Part Using TCL

You can use the following TCL command to generate part. Basically the command is the Tcl version for dialog, New Part Creation Spreadsheet.

![New Part Creation Spreadsheet](image)

Syntax:
capCreatePartsFromData "option1" "Path" "List"

Where:
- **option1**
  - true, if part should be created in a new library
  - false, if part should be created in an existing library
- **Path** - Path of library(.olb)
- **List**
  - OptionList, pin1list, pin2list,...
  - OptionList - "no. of section" "Prefix" "Alpha or numeric for section"
  - pinlist - "Number" "Name" "Type" "Pin Visibility" "Shape" "PinGroup" "Position" "Section"

To create your own file, use text editor to collect information and format it

**Example:**
capCreatePartsFromData "true"
"F:\User\lib\Test.olb"
[list
  [list "tcltest4" "2" "U" "1"]
  [list "12" "tcl" "3 State" "1" "Clock" "5" "Top" "A"]
  [list "13" "tcl" "Bidirectional" "1" "Line" "5" "Left" "B"]
]

```tcl
list
  [list "tcltest4" "2" "U" "1"]
  [list "12" "tcl" "3 State" "1" "Clock" "5" "Top" "A"]
  [list "13" "tcl" "Bidirectional" "1" "Line" "5" "Left" "B"]
```

```python
list
  [list "tcltest4" "2" "U" "1"]
  [list "12" "tcl" "3 State" "1" "Clock" "5" "Top" "A"]
  [list "13" "tcl" "Bidirectional" "1" "Line" "5" "Left" "B"]
```
capCreatePartsFromData - TCL Command
"true" - Use existing OLB, "false" - Use new OLB
"F:\User\lib\Test.olb" - Path to OLB
[list [list [list "tcltest4" "2" "U" "1"] ] - Start of list with part definitions

Now create a one row command, copy it to Command window and execute.

capCreatePartsFromData "true" "F:\User\lib\Test.olb" [list [list [list "tcltest3" "2" "U" "1"] [list "12" "tcl" "3 State" "1" "Clock" "5" "Top" "A"] [list "13" "tcl" "Bidirectional" "1" "Line" "5" "Left" "B"]]]

That's the result:

Link to TCL code in text file - CreatePart_editor.tcl.txt (editor format) and CreatePart_command.tcl.txt (command format)

NOTE:
If you have created a part, let’s say “tcltest3” in a newly created OLB, now when you run the script second time with option1 set to "False" for the same .OLB, use a different name for part, e.g. “tcltest4”. You can’t overwrite existing parts.
4.6 How to Update Variant Information in the Titleblock in variant view?

This script is an example based on a sample design from Cadence. You need to customize the titleblock symbol and the Tcl script. The script also includes the design specific property values of the Titleblock properties. So every design needs its own script.

You can refer this example Tcl script to get a feeling how it works. This script works on these properties.

- RevCode
- Title
- Doc

Follow these steps to learn how the script works.

1. Go to $CDSROOT\tools\capture\samples and open bench_allegro.dsn. Open the design and save it. This is to make sure we have corresponding .OPJ file.

2. Go to View > Command Window (If Command Window is not opened).

3. Now source the Variant.tcl code and run it.

4. Source {Path to the variant.tcl}. If file exist at D:\Variant_DSN, then command will be like below.

   > source {D:\Variant_DSN\variant.tcl}

5. After successfully running it will return 1. If you get a 0, then re-run the file.

6. Now switch to View > Variant View and select any variant.

![Select a Design Variant](image)
You will get several Call Back Windows.

![Pre Variant View CallBack](image)

Pre Variant Call Back and Post Variant Call Back are common for all variant design on which you run the script and remaining call backs windows are related to number of schematic pages.

Click **OK** on every window and then you will see the changes in titleblock.

**Link** to TCL code in text file - [Variant.tcl](#)
4.7 Customizing Menus and Toolbars

Add Menu permanent

Follow below steps to add a new menu to the Capture tool bar:

1. Close Capture (if already invoked).
2. Go to `%CDSROOT%\tools\capture\tclscripts\capAutoLoad` and paste the attached file `testMenu.tcl`.
3. Invoke Capture. You would see a new menu “MyFileTst1” in tool bar menus.

You can modify the script to add the menus as required.

Add the menu only for current session

We assume that the .tcl script is placed at `%CDSROOT%\tools\capture\tclscripts\capUtils`.

1. In Capture first open Command Window
2. Type below command in Command Line Prompt
3. source C:\\Cadence\\SPB_16.6\\tools\\capture\\tclscripts\\\capUtils\\testMenu.tcl
4. Now open a design and go to schematic page. You will see the newly added menu MyFileTst1.

**Note:**
When you reboot your machine, sub menu will not appear.

Add Menu in 16.5

The ability to add custom menu under the main Menu-bar is only available in Capture 16.6. In Capture 16.5, we can only add custom menu under 'Accessories' menu. To add a customized sub-menu under Accessories menu, use the attached TCL script capMenuUtil.tcl, when it runs, it adds a menu named Net Properties under the Accessories menu of Capture, as shown below.

The menu contains popup for Line styles, namely Dot Line and Default. The user can select single or multiple nets on the schematic page and then choose one of the two options for assigning the desired net Line style. Following are the steps to do add the menu:
1. Close Capture, if already running
2. Copy the file `capMenuUtil.tcl` to the location, `$CDSROOT\tools\capture\tclscripts\capAutoLoad`
3. Next time Capture is invoked, a menu option named **Net Properties** should be displayed under **Accessories** menu, as shown in the above snapshot.

Also, the script itself can be modified to add a submenu and associated pop-ups for different properties under the same menu.

**Link** to TCL code in text file - `testMenu.tcl` and `capMenuUtil.tcl`.

### 4.8 Shortcuts

All Capture shortcuts are predefined by the tool and can’t be changed. The only way to create your own shortcuts is to create new menu item like described above and add a shortcut definition to it. You can’t overwrite existing shortcuts. Use only not existing keys or key combinations to create your own shortcuts.

First you need to know what is the command you want to attach to the shortcut. Open command window and execute the command you like in Capture. Our example is to close the current page. The command information from command window is "57927".

Now you need to add the shortkey Shift+X and the command information to your newly created menu. Please see attached code example to create your own shortkey.

**Shortkey can be:**
- A single key
- A combination with Ctrl (Strg)
- And / or a combination with Shift
- And / or a combination with Alt

**Link** to TCL code in text file - `shortcut.tcl`
4.9 Customize page

In folder $CDSROOT\tools\capture\tclscripts\capCustomSamples Cadence provides several customer samples. CapCustomizePage.tcl and TESTCUSTOMIZEPAGE.OLB can be used to place more than one symbol (titleblock) during page creation. For example, to place an additional symbol with copyright information on the left edge. The script also includes a function which allows the user to choose page size during page creation.

To execute this custom sample go to Accessories > Cadence Tcl/Tk Utilities > Utilities > Customize Page > Launch.

Activate the function with Enable Page Customization and close window using OK. This action creates new entries in capture.ini. Now create a new page. This window opens.

Choose the correct page size and continue with OK.

Now you need to customize script and OLB to get your own symbols placed. Therefore move Tcl script to your local environment. How to manage script in your local environment you find in chapter 1.4.
5 TCL/Tk Commands for Capture CIS

5.1 Make multible values in database field visible in CIS

In CIS DB you can create multi value fields. Therefore you add several values separated by comma in CIS database. We will use field datasheet as example.

Field value in CIS database:
- Datasheet1.pdf,Datasheet2.pdf,QM-Report.docx

In CIS Explorer only a few fields are defined as multi value. One example is Schematic Part. If you want to define other multi value fields, you can do it using this TCL command:

```
SetCISMultiValuedField "FieldName"
```

Datasheet example:

```
SetCISMultiValuedField Datasheet
SetCISMultiValuedField prop1
SetCISMultiValuedField prop2
```

Now open CIS Explorer through Place Database part operation. The above commands would change Datasheet, prop1 and prop2 as multi valued. If the field of the selected part has multi value field in database, then you will see it as drop down list in CIS Explorer.

These commands add a section [CIS MultiValued Field] in capture.ini file and define these properties as multivalued by setting it to 1. Like, for above 3 commands, INI file will have:

```
[CIS MultiValued Field]
Datasheet=1
Prop1=1
Prop2=1
```

To clear all the multivalued field, use below TCL command:

```
ClearMultiValuedField
```

**Note:**
It is not recommended to make keyed properties like Part Number and Value as multi-valued.

You can combine the multi value action with EOL Management, explained in next chapter.
5.2 EOL Management

EOL stands for End Of Live

Color and hide parts

Using TCL you can color parts with specific information. We use property Part_Status as example. The attached script works with FlowCAD Starter Library.

<table>
<thead>
<tr>
<th>Table</th>
<th>Part Number</th>
<th>Part Type</th>
<th>Part_Status</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1-Capacitor</td>
<td>FC-CAP-0006</td>
<td>ELEC</td>
<td>obsolete</td>
</tr>
<tr>
<td>2</td>
<td>1-Capacitor</td>
<td>FC-CAP-0061</td>
<td>ELEC</td>
<td>standard</td>
</tr>
<tr>
<td>3</td>
<td>1-Capacitor</td>
<td>FC-CAP-1000</td>
<td>ELEC</td>
<td>special</td>
</tr>
</tbody>
</table>

You see the Part_Status field and its values. Obsolete part are marked in red, special parts are marked yellow and standard parts are marked in green.

It depends on the settings in TCL script, which property value assigned a color to the part. You need to modify the script to match it with your database.

AddCISCriteriaEx <query> <operator> <0/1> <text color> <row color>

Example:

AddCISCriteriaEx {("Part_Status"=""obsolete")} {1} [CISGetColor 0 0 0] [CISGetColor 255 0 0] {("Part_Status"=""obsolete")} non case sensitive, defines the filter (property value)

[CISGetColor 0 0 0] the first defines text color, the second defines field color

Where operator is "AND" or "OR"

You can also use property value to filter hidden parts.

- 0 =hide the part
- 1 =show the part
Other examples of the command:

AddCISCriteriaEx
{""Status""""contains""""EOL""""""OR""""""Status""""""contains""""obsolete""""""} {0}
[CISGetColor 255 125 192] [CISGetColor 255 255 255]

AddCISCriteriaEx {
{""Company Part Status""""=""In Stock""""""and""""""Stock count""""""<""""1500""""""or""""""(""""Pre-booked""""""=""""0"""")""""""} {1} [CISGetColor 0 0 0]
[CISGetColor 205 92 92]

AddCISCriteriaEx {
{""Company Part Status""""=""In Stock""""""and""""""Stock count""""""<""""1500""""""and""""""(""""Pre-booked""""""=""""0"""")""""""} {1} [CISGetColor 0 0 0]
[CISGetColor 205 92 92]

Block parts for placement

At the end of the TCL Script you find capPlacePartCheckTrue. This command blocks all parts with property Value = obsolete for placement and create Warning message for user.

EOL Management is implemented in FlowCAD example site. Ask FlowCAD Hotline for download.

Link to TCL code in text file - CisTclSetting.tcl.txt
5.3 Save CIS Explorer Viewed table to Text-File

With this command you can export the currently selected table or query result from CIS Explorer. For example all parts with Part_Status = obsolete.

Result:

<table>
<thead>
<tr>
<th>Table</th>
<th>Part_Number</th>
<th>Part_Type</th>
<th>Value</th>
<th>Tolerance</th>
<th>Rating</th>
<th>Part_Status</th>
<th>Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part_Status</td>
<td>+ obsolete</td>
<td>X7R</td>
<td>3.3n</td>
<td>10%</td>
<td>absolute</td>
<td>X7R</td>
<td></td>
</tr>
<tr>
<td>Part_Status</td>
<td>+ obsolete</td>
<td>X7R</td>
<td>2.2n</td>
<td>10%</td>
<td>absolute</td>
<td>X7R</td>
<td></td>
</tr>
<tr>
<td>Part_Status</td>
<td>+ obsolete</td>
<td>FILM</td>
<td>1n</td>
<td>10%</td>
<td>absolute</td>
<td>FILM</td>
<td></td>
</tr>
<tr>
<td>Part_Status</td>
<td>+ obsolete</td>
<td>FILM</td>
<td>1n</td>
<td>10%</td>
<td>absolute</td>
<td>FILM</td>
<td></td>
</tr>
<tr>
<td>Part_Status</td>
<td>+ obsolete</td>
<td>FILM</td>
<td>2.2n</td>
<td>10%</td>
<td>absolute</td>
<td>FILM</td>
<td></td>
</tr>
<tr>
<td>Part_Status</td>
<td>+ obsolete</td>
<td>X7R</td>
<td>3.3n</td>
<td>10%</td>
<td>absolute</td>
<td>X7R</td>
<td></td>
</tr>
<tr>
<td>Part_Status</td>
<td>+ obsolete</td>
<td>X7R</td>
<td>2.2n</td>
<td>10%</td>
<td>absolute</td>
<td>X7R</td>
<td></td>
</tr>
<tr>
<td>Part_Status</td>
<td>+ obsolete</td>
<td>FILM</td>
<td>1n</td>
<td>10%</td>
<td>absolute</td>
<td>FILM</td>
<td></td>
</tr>
<tr>
<td>Part_Status</td>
<td>+ obsolete</td>
<td>X7R</td>
<td>3.3n</td>
<td>10%</td>
<td>absolute</td>
<td>X7R</td>
<td></td>
</tr>
<tr>
<td>Part_Status</td>
<td>+ obsolete</td>
<td>X7R</td>
<td>2.2n</td>
<td>10%</td>
<td>absolute</td>
<td>X7R</td>
<td></td>
</tr>
<tr>
<td>Part_Status</td>
<td>+ obsolete</td>
<td>FILM</td>
<td>1n</td>
<td>10%</td>
<td>absolute</td>
<td>FILM</td>
<td></td>
</tr>
<tr>
<td>Part_Status</td>
<td>+ obsolete</td>
<td>FILM</td>
<td>1n</td>
<td>10%</td>
<td>absolute</td>
<td>FILM</td>
<td></td>
</tr>
<tr>
<td>Part_Status</td>
<td>+ obsolete</td>
<td>FILM</td>
<td>2.2n</td>
<td>10%</td>
<td>absolute</td>
<td>FILM</td>
<td></td>
</tr>
<tr>
<td>Part_Status</td>
<td>+ obsolete</td>
<td>FILM</td>
<td>2.2n</td>
<td>10%</td>
<td>absolute</td>
<td>FILM</td>
<td></td>
</tr>
<tr>
<td>Part_Status</td>
<td>+ obsolete</td>
<td>FILM</td>
<td>2.2n</td>
<td>10%</td>
<td>absolute</td>
<td>FILM</td>
<td></td>
</tr>
<tr>
<td>Part_Status</td>
<td>+ obsolete</td>
<td>FILM</td>
<td>2.2n</td>
<td>10%</td>
<td>absolute</td>
<td>FILM</td>
<td></td>
</tr>
</tbody>
</table>

TCL command: CISDumpExplorerView

Example with export file name: CISDumpExplorerView {D:\query.txt}
5.4 How to open Part Type tree in expanded mode

When you open CIS Explorer, the database tables are always shown in collapsed format.

To open CIS Explorer tables and Part Type in expand mode use

- SetOptionString CisExplorerPartTypeTreeExpand TRUE

You get this new entry in Preferences section of capture.ini:

- CisExplorerPartTreeExpand=TRUE

When you open CIS Explorer again, all tables and Part Type are shown in expand mode.

To revert this action use Tcl command:

- SetOptionString CisExplorerPartTypeTreeExpand FALSE
5.5 CIS TCL data access sample

This script is a custom example. In current version it works with standard Cadence installation but can also be modified. The current file location is 
%H\%CDSROOT\tools\capture\tclscripts\capCustomSamples\capCIS.tcl. It includes several examples to manage CIS data.

Included Variant Editor functions:
- Dump CIS Data
- Add / Remove Group,
- Add / Remove SubGroup ,
- Add / Remove Parts on Schematic to Group
- Add / Remove Parts with Occurrence to Group
- Add Variant BOM
- Add Group to BOM
- Add SubGroup to BOM
6 FlowCAD App

Ask FlowCAD Hotline for Download of these files.

6.1 FloWare Tortoise SVN Integration

This module enables import of a design into SVN without leaving PCB design environment. You can import the complete project structure into SVN, including all external files which are not created by OrCAD/allegro tools (e.g. datasheets or other documents).

6.2 Test point Checker

This tool checks, if you have placed a test point on each net in the schematic. When on a net a test point is missing, the net is listed in the session log window.

Link to Datasheet - FlowCAD_Check_TestpointOnNet_help.pdf
7 Appendix

7.1 How to Update Variant Information in the Titleblock in variant view?

```
proc GetCISDesign {} {
    set lDesign [GetActivePMDesign]
    set lCISDesign [CPartMgmt_GetCisDesign $lDesign]
    return $lCISDesign
}

proc VariantPropSetEnabler {args} {
    return true
}

proc DisplayAllVariant { } {
    set pCISDesign [GetCISDesign]
    set lBOMVariantContainer [pCISDesign GetBOMVariantContainer]
    set lBOMVariantCount [lBOMVariantContainer GetBomCount]
    set lBOMVariantName [DboTclHelper_sMakeCString]
    for {set i 0} {$i<$lBOMVariantCount} {incr i} {
        $lBOMVariantContainer GetBomName $i $lBOMVariantName
        set lBOMVariantNameStr [DboTclHelper_sGetConstCharPtr
        $lBOMVariantName]
        puts $lBOMVariantNameStr
    }
}

proc VariantPropSetCallBack { aBaseObject } {
    # capDisplayMessageBox "Setting the Variant" "Variant CallBack"
    set lPropNameTitle [DboTclHelper_sMakeCString {Title}]
    set lPropNameDoc [DboTclHelper_sMakeCString {Doc}]
    set lPropNameRevDoc [DboTclHelper_sMakeCString {RevCode}]
    set lActiveVariant [GetActiveVariant]
    capDisplayMessageBox $lActiveVariant "Variant CallBack"
    if {$lActiveVariant=="Variation #1"} {
        set lStr [concat $lActiveVariant _Title#1]
        set lPropValue [DboTclHelper_sMakeCString $lStr]
        $aBaseObject AddVariantProp $lPropNameTitle $lPropValue
        set lStr [concat $lActiveVariant _Rev1]
        set lPropValue [DboTclHelper_sMakeCString $lStr]
        $aBaseObject AddVariantProp $lPropNameRevDoc $lPropValue
    }
```
set lStr [concat $lActiveVariant _Doc1]
set lPropValue [DboTclHelper_makeCstring $lStr]
$aBaseObject AddVariantProp $lPropNameDoc $lPropValue
}

if {$lActiveVariant=="Variation #2"} {
    set lStr [concat $lActiveVariant _Title#2]
    set lPropValue [DboTclHelper_makeCstring $lStr]
    $aBaseObject AddVariantProp $lPropNameTitle $lPropValue

    set lStr [concat $lActiveVariant _Rev2]
    set lPropValue [DboTclHelper_makeCstring $lStr]
    $aBaseObject AddVariantProp $lPropNameRevDoc $lPropValue

    set lStr [concat $lActiveVariant _Doc2]
    set lPropValue [DboTclHelper_makeCstring $lStr]
    $aBaseObject AddVariantProp $lPropNameDoc $lPropValue
}
}

proc VariantPartPropSetCallBack { aBaseObject } {
    # capDisplayMessageBox "Setting the Variant" "Variant CallBack"
    set lPropNameTitle [DboTclHelper_makeCstring {TestVariant}]
    set lActiveVariant [GetActiveVariant]
    #capDisplayMessageBox $lActiveVariant "Variant CallBack"
    if {$lActiveVariant=="Variation #1"} {
        set lStr [concat $lActiveVariant _Test#1]
        set lPropValue [DboTclHelper_makeCstring $lStr]
        $aBaseObject AddVariantProp $lPropNameTitle $lPropValue
    }
    if {$lActiveVariant=="Variation #2"} {
        set lStr [concat $lActiveVariant _Test#2]
        set lPropValue [DboTclHelper_makeCstring $lStr]
        $aBaseObject AddVariantProp $lPropNameTitle $lPropValue
        $aBaseObject AddVariantProp $lPropNameDoc $lPropValue
    }
}

proc GetTitleBlockCount { aInstOcc } {
    set lStatus [DboState]
    set lTitleBlockOccIter [$aInstOcc NewChildrenIter $lStatus $::IterDefs_TITLEBLOCKS]
    set lTitleBlockOcc [$lTitleBlockOccIter NextOccurrence $lStatus]
    set i 0
    set lNullObj NULL
    while {$lTitleBlockOcc!=$lNullObj} {
        incr i
        #capDisplayMessageBox $lStatus "$i" "$lTitleBlockOcc"
    }
}
set lTitleBlockOcc [$lTitleBlockOccIter NextOccurrence $lStatus]
    return $i
}

proc PreVariantViewSetCallBack { aDesign } {
    set lActiveVariant [GetActiveVariant]
    set lStatus [DboState]
    set lDesignOccIter [$aDesign NewOccurrencesIter $lStatus]
    set pDboInstOcc [$lDesignOccIter NextOccurrence $lStatus]
    set varNullObj NULL
    set lTitleBlockCount 0
    while {$pDboInstOcc!=$varNullObj} {
        set lTitleBlockCount [expr $lTitleBlockCount+[GetTitleBlockCount $pDboInstOcc]]
        set pDboInstOcc [$lDesignOccIter NextOccurrence $lStatus]
    }
    delete_DboDesignOccurrencesIter $lDesignOccIter
    set lMessage [concat $lActiveVariant - Title Block Count:$lTitleBlockCount]
    capDisplayMessageBox $lMessage "Pre Variant View CallBack"
}

proc PostVariantViewSetCallBack { aDesign } {
    set lActiveVariant [GetActiveVariant]
    capDisplayMessageBox $lActiveVariant "Post Variant View CallBack"
}

RegisterAction "_cdnOrVariantViewSetTitleBlockProp"
    "VariantPropSetEnabler" "VariantPropSetCallBack"
RegisterAction "_cdnOrPreVariantViewMode"
    "VariantPropSetEnabler" "PreVariantViewSetCallBack"
RegisterAction "_cdnOrPostVariantViewMode"
    "VariantPropSetEnabler" "PostVariantViewSetCallBack"
RegisterAction "_cdnOrVariantViewSetPartProp"
    "VariantPropSetEnabler" "VariantPartPropSetCallBack"
7.2 Property Visibility

```tcl
set lStatus [DboState]
set lDesign [GetActivePMDesign]
set lSchematicIter [lDesign NewViewsIter $lStatus $::IterDefs_SCHEMATICS]
# get the first schematic view
set lView [lSchematicIter NextView $lStatus]
set lNullObj NULL
while { $lView != $lNullObj } { 
    # dynamic cast from DboView to DboSchematic
    set lSchematic [DboViewToDboSchematic $lView]
    # placeholder: do your processing on $lSchematic
    set lPagesIter [lSchematic NewPagesIter $lStatus]
    # get the first page
    set lPage [lPagesIter NextPage $lStatus]
    set lNullObj NULL
    while {$lPage!=$lNullObj} {
        # placeholder: do your processing on $lPage
        set lPartInstsIter [lPage NewPartInstsIter $lStatus]
        # get the first part inst
        set lInst [lPartInstsIter NextPartInst $lStatus]
        while {$lInst!=$lNullObj} {
            # dynamic cast from DboPartInst to DboPlacedInst
            set lPlacedInst [DboPartInstToDboPlacedInst $lInst]
            if {$lPlacedInst != $lNullObj} {
                # placeholder: do your processing on $lPlacedInst
                set lPropsIter [lPlacedInst NewDisplayPropsIter $lStatus]
                set lNullObj NULL
                # get the first display property on the object
                set lDProp [lPropsIter NextProp $lStatus]
                while {$lDProp !=$lNullObj } { 
                    # placeholder: do your processing on $lDProp
                    # get the name of Display Property
                    set lName [DboTclHelper_sMakeCString]
                    $lDProp GetName $lName
                    set lNameString [DboTclHelper_sGetConstCharPtr $lName]
                    if { $lNameString == "Value" } {
                        # setting the display property to DND
                        $lDProp SetDisplayType 0
                    }
                    #if {($lDProp == "VALUE") {
                    #SetDisplayType "VALUE" 0
                    #}
                    set lDProp [lPropsIter NextProp $lStatus]
                }
                delete_DboDisplayPropsIter $lPropsIter
                # set lStatus [lPlacedInst
                SetEffectivePropStringValue $lPropNameCStr $lPropValueCStr]
            }
```
#set varNullObj NULL
#set pDispProp [lPlacedInst GetDisplayProp
$lPropNameCStr $lStatus]
#set lStatus [DboState]
#set pNewDispProp [lPlacedInst NewDisplayProp
$lStatus $lPropNameCStr $displocation $rotation $logfont $color]
#DO_NOT_DISPLAY = 0,
#VALUE_ONLY = 1,
#NAME_AND_VALUE = 2,
#NAME_ONLY = 3,
#BOTH_IF_VALUE = 4,
#$pNewDispProp SetDisplayType
::DboValue_NAME_AND_VALUE
}
#get the next part inst
set lInst [lPartInstsIter NextPartInst $lStatus]
}
delete_DboPagePartInstsIter $lPartInstsIter
#get the next page
set lPage [lPagesIter NextPage $lStatus]
}
delete_DboSchematicPagesIter $lPagesIter
#get the next schematic view
set lView [lSchematicIter NextView $lStatus]
}
delete_DboLibViewsIter $lSchematicIter
7.3 Customizing Menus and Toolbars

TCL Code testMenu.tcl

package provide testMenu 1.0

namespace eval ::testMenu {

    proc registerMenuActions { args } {
        catch {

            InsertXMLMenu [list [list "MyFile1"] "" "" [list "popup" "MyFileTest1" "" "" "" "" ] ] ]
            InsertXMLMenu [list [list "MyFile1" "MyFileLevelpopup1"] "" "" [list "popup" "MyFileLevel11" "" "" "" "" ] ] ]
            InsertXMLMenu [list [list "MyFile1" "MyFileLevelpopup1" "MyFileLevelpopup1"] "" "" [list "popup" "MyFileLevel11" "" "" "" "" ] ] ]
            InsertXMLMenu [list [list "MyFile1" "MyFileLevelpopup1" "MyFileLevelpopup1" "MyFileLevelpopup1"] "" "" [list "separator"] ]]
            InsertXMLMenu [list [list "MyFile1" "MyFileLevelpopup1" "MyFileLevelpopup1" "MyFileLevelpopup1"] "" "" [list "separator"] ]]

            RegisterAction "ActioForMenu1" "::testMenu::shouldProcess" "::testMenu::testActionProc1" ""
            RegisterAction "UpdateForMenu1" "::testMenu::shouldProcess" "::testMenu::testUpdateProc1" ""

            RegisterAction "ActioForMenu2" "::testMenu::shouldProcess" "::testMenu::testActionProc2" ""
            RegisterAction "UpdateForMenu2" "::testMenu::shouldProcess" "::testMenu::testUpdateProc2" ""

        }
    }

    proc shouldProcess { args } {
        return 1
    }

    proc testActionProc1 { args } {
        puts "Got in testActionProc1!"
    }

    proc testUpdateProc1 { args } {
        return true;
    }
proc testActionProc2 { args } {
    puts "Got in testActionProc2!"
}

proc testUpdateProc2 { args } {
    return false;
}

::testMenu::registerMenuActions
TCL Code capMenuUtil.tcl

```tcl
#package require TCL 8.4
package provide capMenuUtil 1.0
namespace eval ::capMenuUtil {

proc ::capMenuUtil::addPageAccessoryMenu { } {
    # AddAccessoryMenu <User menu under Accessories> <SubMenu under user menu> <TCL callback #handler with 2 parameters pPage and pOcc>
    AddAccessoryMenu "Net Properties" "DOT Line" ::capMenuUtil::dotStyle
    AddAccessoryMenu "Net Properties" "Default" ::capMenuUtil::defaultStyle
}

proc ::capMenuUtil::dotStyle { pPage pOcc } {
    set selectedobjs [GetSelectedObjects]
    set dbostate [DboState]
    foreach wireobj $selectedobjs {
        if {
            [GetObjectType] $::DboBaseObject_WIRE_SCALAR
            || [GetObjectType] $::DboBaseObject_WIRE_BUS
        } {
            set netObj [GetNet $dbostate]
            $netObj SetEffectiveColorValue $::DboValue_DEFAULT_OBJECT_COLOR $::DboValue_DOT_LINE
            $::DEFAULT_LINE_WIDTH [GetInstanceOccurrence]
        }
    }
}

proc ::capMenuUtil::defaultStyle { pPage pOcc } {
    set selectedobjs [GetSelectedObjects]
    set dbostate [DboState]
    foreach wireobj $selectedobjs {
        if {
            [GetObjectType] $::DboBaseObject_WIRE_SCALAR
            || [GetObjectType] $::DboBaseObject_WIRE_BUS
        } {
            set netObj [GetNet $dbostate]
            $netObj SetEffectiveColorValue $::DboValue_DEFAULT_OBJECT_COLOR $::DboValue_DEFAULT_LINE_STYLE
            $::DEFAULT_LINE_WIDTH [GetInstanceOccurrence]
        }
    }
}
```

proc ::capMenuUtil::capTrue { } {
  return 1
}

RegisterAction "cdnCapTclAddPageCustomMenu" "::capMenuUtil::capTrue"
  "" "::capMenuUtil::addPageAccessoryMenu" ""

RegisterAction "cdnCapTclAddDesignCustomMenu" "::capMenuUtil::capTrue"
  "" "::capMenuUtil::addDesignAccessoryMenu" ""
7.4 EOL Management

```tcl
proc capTrue { args } {
    return 1
}

RegisterAction "_cdnOrPreCisExplorerCreate" "capTrue" "" "CisExplorerPreCreate" ""

proc CisExplorerPreCreate { args } {
    set result 1
    AddCISCriteriaEx {("Part_Status"="obsolete")} {1} [CISGetColor 0 0 0] [CISGetColor 255 0 0]
    AddCISCriteriaEx {("Part_Status"="special")} {1} [CISGetColor 0 0 0] [CISGetColor 255 255 0]
    AddCISCriteriaEx {("Part_Status"="standard")} {1} [CISGetColor 0 0 0] [CISGetColor 0 255 0]
    return $result
}

proc capPlacePartCheckTrue { args } {
    return 1
}

proc PlacePartCheck { args } {
    set result 1
    foreach {name val} $args {
        if {[string compare -nocase $val "obsolete"]==0} {
            capDisplayMessageBox "Place Part Warning For Condition $name=$val" "Place Part Warning"
            set result 0
        }
    }
    return $result
}

RegisterAction "_cdnOrPreCisPlacePart" "capPlacePartCheckTrue" "" "PlacePartCheck" ""
```