OrCAD X TCL Sample Scripts

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

Overview

The OrCAD TCL Sample Snippets guide contains basic sample snippets, most of which can be directly used in the OrCAD® Capture Command window. To use the OrCAD TCL snippets and scripts effectively, you need to have a basic understanding of the TCL language. To quickly get started with TCL scripting in OrCAD Capture, read the documents in the following order:

1. OrCAD Capture Information Model
2. Getting Started with OrCAD Capture TCL
3. OrCAD TCL Sample Scripts

The OrCAD TCL Sample Scripts documents covers the following:

1. Basic OrCAD TCL Sample Snippets
2. Introduction to Basic OrCAD TCL Scripts
3. TCL Programming with OrCAD Capture Database
4. TCL Programming with OrCAD Capture GUI Interface

2

Basic OrCAD TCL Sample Snippets

This chapter provides OrCAD TCL scripts that you can use to do basic tasks in OrCAD Capture. You can paste most of the scripts as is to the Command Window to get the results. Some of the scripts have placeholders for custom code to get results.

Getting the current Capture session

```
set lSession $::DboSession_s_pDboSession
DboSession -this $lSession
```

Creating a new session

```
set lSession [DboTclHelper_sCreateSession]
```

Opening a design in the Capture session

```
set lStatus [DboState]
# set pDesignPath d:/spb172/tools/capture/samples/fulladd.dsn <- EXAMPLE
set lDesignPath [DboTclHelper_sMakeCString $pDesignPath]
set lDesign [$lSession GetDesignAndSchematics $lDesignPath $lStatus]
```

Iterating through all open designs in the session

```
set lDesignsIter [$lSession NewDesignsIter $lStatus]
#get the first design
set lDesign [$lDesignsIter NextDesign $lStatus]
set lNullObj NULL while { $lDesign!= $lNullObj} {
    #placeholder: do your processing on $lDesign
    #get the next design
    set lDesign [$lDesignsIter NextDesign $lStatus]
}
delete_DboSessionDesignsIter $lDesignsIter
```

Getting schematic of a design

```
# set pSchematicName SCHEMATIC1 ═ EXAMPLE
set lSchematicName [DboTclHelper_sMakeCString $pSchematicName]
set lSchematic [$lDesign GetSchematic $lSchematicName $lStatus]
```

Iterating through all schematics of a design

```
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 $lSchematic
    #get the next schematic view
    set lView [$lSchematicIter NextView  $lStatus]
}
delete_DboLibViewsIter $lSchematicIter
```

Getting page of a schematic

```
# set pPageName PAGE1 <- EXAMPLE
set lPageName [DboTclHelper_sMakeCString $pPageName]
set lPage [$lSchematic GetPage $lPageName $lStatus]
```

Iterating through all pages of a schematic

```
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 $lPage
    #get the next page
    set lPage [$lPagesIter NextPage $lStatus]
}
delete_DboSchematicPagesIter $lPagesIter
```

Iterating through all part instances of a page

```
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

    }
    #get the next part inst
    set lInst [$lPartInstsIter NextPartInst $lStatus]
}
delete_DboPagePartInstsIter $lPartInstsIter
```

Iterating through all hierarchical instances of a page

```
set lPartInstsIter [$lPage NewPartInstsIter $lStatus]

#get the first part inst
set lInst [$lPartInstsIter NextPartInst $lStatus]

while {$lInst!=$lNullObj} {
    #dynamic cast from DboPartInst to DboDrawnInst
    set lDrawnInst [DboPartInstToDboDrawnInst $lInst]

    if {$lDrawnInst != $lNullObj} {
      #placeholder: do your processing on $lDrawnInst
}

 #get the next part inst
 set lInst [$lPartInstsIter NextPartInst $lStatus]
}

delete_DboPagePartInstsIter $lPartInstsIter
```

Iterating through all wires in a page

```
set lWiresIter [$lPage NewWiresIter $lStatus]

#get the first wire set
lWire [$lWiresIter NextWire $lStatus]
set lNullObj NULL

while {$lWire != $lNullObj} {

 set lObjectType [$lWire GetObjectType]
 if {$lObjectType == $::DboBaseObject_WIRE_SCALAR} {
  #placeholder: do your processing on Wire scalar $lWire
 } elseif {$lObjectType == $::DboBaseObject_WIRE_BUS} {
  #placeholder: do your processing on Wire Bus $lWire
 }

 #get the next wire set
 lWire [$lWiresIter NextWire $lStatus]
}

delete_DboPageWiresIter $lWiresIter
```

Iterating through all globals in a page

```
set lGlobalsIter [$lPage NewGlobalsIter $lStatus]

#get the first global set
lGlobal [$lGlobalsIter NextGlobal $lStatus]

while { $lGlobal!=$lNullObj } {

 #placeholder: do your processing on $lGlobal

 #get the next global set
 lGlobal [$lGlobalsIter NextGlobal $lStatus]
}

delete_DboPageGlobalsIter $lGlobalsIter
```

Iterating through all title-blocks in a page

```
set lTitleBlocksIter [$lPage NewTitleBlocksIter $lStatus]

#get the first title block set

lTitle [$lTitleBlocksIter NextTitleBlock $lStatus]

while {$lTitle!=$lNullObj} {

 #placeholder: do your processing on $lTitle

 #get the next title block set
 lTitle [$lTitleBlocksIter NextTitleBlock $lStatus]

}

delete_DboPageTitleBlocksIter $lTitleBlocksIter
```

Iterating through all ports in a page

```
set lPortsIter [$lPage NewPortsIter $lStatus]

#get the first port of the page
set lPort [$lPortsIter NextPort $lStatus]

while {$lPort!=$lNullObj} {

 #placeholder: do your processing on $lPort

 #get the next port of the page
 set lPort [$lPortsIter NextPort $lStatus]
}

delete_DboPagePortsIter $lPortsIter
```

Iterating through all off-pages in a page

```
set lOffPagesIter [$lPage NewOffPageConnectorsIter $lStatus $::IterDefs_ALL]

#get the first off-page of the page
set lOffPage [$lOffPagesIter NextOffPageConnector $lStatus]

while {$lOffPage!=$lNullObj} {

 #placeholder: do your processing on $lOffPage

 #get the next off-page of the page
 set lOffPage [$lOffPagesIter NextOffPageConnector $lStatus]
}

delete_DboPageOffPageConnectorsIter $lOffPagesIter
```

Iterating through all graphics in a page

```
set lCommentsIter [$lPage NewCommentGraphicsIter $lStatus]

#get the first graphics of the page
set lGraphic [$lCommentsIter NextCommentGraphic $lStatus]
while {$lGraphic!=$lNullObj} {

 set lType [$lGraphic GetObjectType]
 if {$lType == $::DboBaseObject_GRAPHIC_BOX_INST} {
  set lBoxInst [DboGraphicInstanceToDboGraphicBoxInst $lGraphic]

  #placeholder: do your processing on $lBoxInst
 } elseif {$lType == $::DboBaseObject_GRAPHIC_LINE_INST} {
  set lLineInst [DboGraphicInstanceToDboGraphicLineInst $lGraphic]

  #placeholder: do your processing on $lLineInst
 } elseif {$lType == $::DboBaseObject_GRAPHIC_ELLIPSE_INST} {
  set lEllipseInst [DboGraphicInstanceToDboGraphicEllipseInst $lGraphic]

  #placeholder: do your processing on $lEllipseInst
 } elseif {$lType == $::DboBaseObject_GRAPHIC_ARC_INST} {

  set lArcInst [DboGraphicInstanceToDboGraphicArcInst $lGraphic]

  #placeholder: do your processing on $lArcInst
 } elseif {$lType == $::DboBaseObject_GRAPHIC_POLYLINE_INST} {

  set lPolylineInst [DboGraphicInstanceToDboGraphicPolylineInst $lGraphic]

  #placeholder: do your processing on $lPolylineInst
 } elseif {$lType == $::DboBaseObject_GRAPHIC_POLYGON_INST} {
  set $lPolygonInst [DboGraphicInstanceToDboGraphicPolygonInst $lGraphic]

  #placeholder: do your processing on $lPolygonInst
 } elseif {$lType == $::DboBaseObject_GRAPHIC_BITMAP_INST} {
  set lBitMapInst [DboGraphicInstanceToDboGraphicBitMapInst $lGraphic]

  #placeholder: do your processing on $lBitMapInst
 } elseif {$lType == $::DboBaseObject_GRAPHIC_COMMENTTEXT_INST} {
  set lTextInst [DboGraphicInstanceToDboGraphicCommentTextInst $lGraphic]

  #placeholder: do your processing on $lTextInst
 }
 #get the next graphics of the page
 set lGraphic [$lCommentsIter NextCommentGraphic $lStatus]
}

delete_DboPageCommentGraphicsIter $lCommentsIter
```

Iterating through all pins of a part instance or drawn instance

```
set lIter [$lInst NewPinsIter $lStatus]

set lNullObj NULL

#get the first pin of the part
set lPin [$lIter NextPin $lStatus]

while {$lPin !=$lNullObj } {

 #placeholder: do your processing on $lPin

 #get the next pin of the part
 set lPin [$lIter NextPin $lStatus]
}

delete_DboPartInstPinsIter $lIter
```

Iterating through all aliases of a wire

```
set lAliasIter [$lWire NewAliasesIter $lStatus]

#get the first alias of wire
set lAlias [$lAliasIter NextAlias $lStatus]

while { $lAlias!=$lNullObj} {

 #placeholder: do your processing on $lAlias

 #get the next alias of wire
 set lAlias [$lAliasIter NextAlias $lStatus]
}

delete_DboWireAliasesIter $lAliasIter
```

Iterating through all flat nets in a design

```
set lFlatNetsIter [$pDesign NewFlatNetsIter $lStatus]

#get the first flat net of design
set lFlatNet [$lFlatNetsIter NextFlatNet $lStatus]

while {$lFlatNet!=$lNullObj} {

 #placeholder: do your processing on $lFlatNet
 set lNetName [DboTclHelper_sMakeCString]
 $lFlatNet GetName $lNetName

 #get the next flat net of design
 set lFlatNet [$lFlatNetsIter NextFlatNet $lStatus]
}

delete_DboDesignFlatNetsIter $lFlatNetsIter
```

Iterating through all user properties of any object

```
set lPropsIter [$lObject NewUserPropsIter $lStatus]

set lNullObj NULL

#get the first user property on the object
set lUProp [$lPropsIter NextUserProp $lStatus]

while {$lUProp !=$lNullObj } {

 #placeholder: do your processing on $lUProp
 set lName [DboTclHelper_sMakeCString]

 set lValue [DboTclHelper_sMakeCString]

 $lUProp GetName $lName
 $lUProp GetStringValue $lValue

 #get the next user property on the object
 set lUProp [$lPropsIter NextUserProp $lStatus]
}

delete_DboUserPropsIter $lPropsIter
```

Iterating through all display properties of any object

```
set lPropsIter [$lObject 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
 set lName [DboTclHelper_sMakeCString]
 $lDProp GetName $lName

 #get the location
 set lLocation [$lDProp GetLocation $lStatus]

 #get the rotation
 set lRot [$lDProp GetRotation $lStatus]

 #get the font
 set lFont [DboTclHelper_sMakeLOGFONT]
 set lFont [$lDProp GetFont $lStatus]

 #get the color
 set lColor [$lDProp GetColor $lStatus]

 #get the next display property on the object
 set lDProp [$lPropsIter NextProp $lStatus]

}

delete_DboDisplayPropsIter $lPropsIter
```

Changing a display property of any object

```
proc ConvertUserToDoc { pPage pUser } {
 set lDocDouble [expr "[$pPage GetPhysicalGranularity] * $pUser + 0.5"]
 set lDoc [expr "round($lDocDouble)"]
 return $lDoc
}
proc AddDisplayProperty {} {
 # Get the selected objects
 set lSelObjs1 [GetSelectedObjects]
 set lObj1 [lindex $lSelObjs1 0]
 set lPropNameCStr [DboTclHelper_sMakeCString "ASSEMBLY"]
 set lPropValueCStr [DboTclHelper_sMakeCString "NC"]
 set lStatus [$lObj1 SetEffectivePropStringValue $lPropNameCStr $lPropValueCStr]
 set varNullObj NULL
 set pDispProp [$lObj1 GetDisplayProp $lPropNameCStr $lStatus]
 set lStatus [DboState]
 if { $pDispProp == $varNullObj } {
  set rotation 0
  set logfont [DboTclHelper_sMakeLOGFONT]
  set color $::DboValue_DEFAULT_OBJECT_COLOR
  #set displocation [DboTclHelper_sMakeCPoint [expr $xlocation] [expr $ylocation]]
  if {[catch {set lPickPosition [GetLastMouseClickPointOnPage]} lResult] } {
   set lX 0 set lY 0
   set displocation [DboTclHelper_sMakeCPoint $intX $intY]

  } else {

   set page [$lObj1 GetOwner]
   set lX [ConvertUserToDoc $page [lindex $lPickPosition 0]]
   set lY [ConvertUserToDoc $page [lindex $lPickPosition 1]]
   set displocation [DboTclHelper_sMakeCPoint $lX $lY]
   }
  set pNewDispProp [$lObj1 NewDisplayProp $lStatus $lPropNameCStr $displocation $rotation
$logfont $color]

  #DO_NOT_DISPLAY    = 0,
  #VALUE_ONLY    = 1,
  #NAME_AND_VALUE    = 2,
  #NAME_ONLY    = 3,
  #BOTH_IF_VALUED    = 4,
   $pNewDispProp SetDisplayType $::DboValue_NAME_AND_VALUE

 } else {
  $pDispProp SetDisplayType $::DboValue_NAME_ONLY
 }
}
```

Iterating through all effective properties of any object

```
set lPropsIter [$lObject NewEffectivePropsIter $lStatus]
set lNullObj NULL

#create the input/output parameters
set lPrpName [DboTclHelper_sMakeCString]
set lPrpValue [DboTclHelper_sMakeCString]
set lPrpType [DboTclHelper_sMakeDboValueType]
set lEditable [DboTclHelper_sMakeInt]

#get the first effective property
set lStatus [$lPropsIter NextEffectiveProp $lPrpName $lPrpValue $lPrpType $lEditable]

while {[$lStatus OK] == 1} {

 #placeholder: do your processing for $lPrpName $lPrpValue $lPrpType $lEditable

 #get the next effective property
 set lStatus [$lPropsIter NextEffectiveProp $lPrpName $lPrpValue $lPrpType $lEditable]

}

delete_DboEffectivePropsIter $lPropsIter
```

Getting attributes of a part or drawn instance

```
#get the name
set lName [DboTclHelper_sMakeCString]
$lInst GetName $lName

#get the location point
set lLocation [$lInst GetLocation $lStatus]

#get the location x
set lStartx [DboTclHelper_sGetCPointX $lLocation]

#get the location y
set lStarty [DboTclHelper_sGetCPointY $lLocation]

#get the source library name
set lLibName [DboTclHelper_sMakeCString]
$lInst GetSourceLibName $lLibName

#get the device designator
set lDeviceDesignator [DboTclHelper_sMakeCString]
set lDeviceDesignator [DboTclHelper_sMakeCString]
$lInst GetReferenceDesignator $lDeviceDesignator

#get the rotation
set lRot [$lInst GetRotation $lStatus]

#get the contents lib name
set lContentsLibName [DboTclHelper_sMakeCString]
$lInst GetContentsLibName $lContentsLibName

#get the contents view name
set lContentsViewName [DboTclHelper_sMakeCString]
$lInst GetContentsViewName $lContentsViewName

#get the contents view type
set lType [$lInst GetContentsViewType $lStatus]

#get the primitive type
set lPrimitiveType [$lInst GetIsPrimitiveProp $lStatus]

#get the part value
set lValue [DboTclHelper_sMakeCString]
$lInst GetPartValue $lValue

#get the reference
set lReferenceName [DboTclHelper_sMakeCString]
$lInst GetReference $lReferenceName

#get the bounding box on the page
set lBBox [$lInst GetOffsetBoundingBox $lStatus]

#get the top-left of the bbox
set lTopLeft [DboTclHelper_sGetCRectTopLeft $lBBox]

#get the bottom-right of the bbox
set lBottomRight [DboTclHelper_sGetCRectBottomRight $lBBox]

#get the x1
set lStartx [DboTclHelper_sGetCPointX $lTopLeft]

#get the y1
set lStarty [DboTclHelper_sGetCPointY $lTopLeft]

#get the x2
set lEndx [DboTclHelper_sGetCPointX $lBottomRight]

#get the y2
set lEndy [DboTclHelper_sGetCPointY $lBottomRight]
```

Getting wire attributes

```
#get the name
set lName [DboTclHelper_sMakeCString]
$lWire GetName $lName

#get the net name
set lNetName [DboTclHelper_sMakeCString]
$lWire GetNetName $lNetName

#get the start point
set  lStart [$lWire GetStartPoint $lStatus]
set lStartx [DboTclHelper_sGetCPointX $lStart]
set lStarty [DboTclHelper_sGetCPointY $lStart]

#get the end point
set lEnd [$lWire GetEndPoint $lStatus]
set lEndx [DboTclHelper_sGetCPointX $lEnd]
set lEndy [DboTclHelper_sGetCPointY $lEnd]

#get the color
set lColor [$lWire GetColor $lStatus]

#get the net
set lNet [$lWire GetNet $lStatus]
```

Getting root instance occurrence of a design

```
set lRootOcc [$lDesign GetRootOccurrence $lStatus]
```

Iterating through the instance occurrence hierarchy

```
proc traverse_hierarchy { lInstOcc } {

 set lStatus [DboState]
 set lNullObj NULL

 set lInstOccIter [$lInstOcc NewChildrenIter $lStatus $::IterDefs_INSTS]

 #get the first child occurrence
 set lChildOcc [$lInstOccIter NextOccurrence $lStatus]

 while { $lChildOcc!= $lNullObj} {

  #get the DboInstOccurrence pointer from DboOccurrence pointer
  set lChildInstOcc [DboOccurrenceToDboInstOccurrence $lChildOcc]

  # placeholder: do your processing on $lChildInstOcc

  # do a recursion of the procedure call for $lChildInstOcc
  traverse_hierarchy $lChildInstOcc

  #get the next child occurrence
  set lChildOcc [$lInstOccIter NextOccurrence $lStatus]

}

delete_DboOccurrenceChildrenIter $lInstOccIter
```

Iterating port occurrences within an instance occurrence

```
set lPortOccIter [$pInstOcc NewChildrenIter $lStatus $::IterDefs_PORTS]

#get the first child port occurrence
set lPortOcc [$lPortOccIter NextOccurrence $lStatus]

while { $lPortOcc!= $lNullObj} {

 # placeholder: do your processing on $lPortOcc

 #get the next child port occurrence
 set lPortOcc [$lPortOccIter NextOccurrence $lStatus]

}

delete_DboOccurrenceChildrenIter $lPortOccIter
```

Iterating off-page occurrences within an instance occurrence

```
set lOffpageOccIter [$pInstOcc NewChildrenIter $lStatus  $::IterDefs_OFFPAGES]

#get the first child offpage occurrence
set lOffpageOcc [$lOffpageOccIter NextOccurrence $lStatus]

while { $lOffpageOcc!= $lNullObj} {
 # placeholder: do your processing on $lOffpageOcc

 #get the next child offpage occurrence
 set lOffpageOcc [$lOffpageOccIter NextOccurrence $lStatus]
}

delete_DboOccurrenceChildrenIter $lOffpageOccIter
```

Iterating net occurrences within an instance occurrence

```
set lNetOccIter [$pInstOcc NewChildrenIter $lStatus  $::IterDefs_NETS]

#get the first child Net occurrence
set lNetOcc [$lNetOccIter NextOccurrence $lStatus]

while { $lNetOcc!= $lNullObj} {
 # placeholder: do your processing on $lNetOcc

 #get the next child Net occurrence
 set lNetOcc [$lNetOccIter NextOccurrence $lStatus]
}

delete_DboOccurrenceChildrenIter $lNetOccIter
```

Iterating title-block occurrences within an instance occurrence

```
set lTitleBlockOccIter [$pInstOcc NewChildrenIter $lStatus  $::IterDefs_TITLEBLOCKS]

#get the first child TitleBlock occurrence
set lTitleBlockOcc [$lTitleBlockOccIter NextOccurrence $lStatus]

while { $lTitleBlockOcc!= $lNullObj} {
 # placeholder: do your processing on $lTitleBlockOcc

 #get the next child TitleBlock occurrence
 set lTitleBlockOcc [$lTitleBlockOccIter NextOccurrence $lStatus]

}

delete_DboOccurrenceChildrenIter $lTitleBlockOccIter
```

3

Introduction to Basic OrCAD TCL Scripts

This chapter describes the following basic examples to get you started with the OrCAD TCL:

- Verifying the TCL Version and the TCL Installation
- Verifying your machine basic information using TCL
- OrCAD TCL Read/Write to different formats
    - Reading and Writing the CSV file with TCL
    - Reading and Writing the SKILL or Lisp format file with TCL
    - Writing a JSON String using TCL
    - Reading a JSON string using TCL

Verifying TCL Version and TCL Installation

The following TCL sample script displays:

- the installed TCL version
- the path of the TCL executable file

Sample-1.tcl

```
puts "Tcl version being accessed is [info tclversion]"
puts "The path of Tcl is [info nameofexecutable]"
```

Verifying System Information using TCL

The following TCL script lists:

- environment variables that are set in the system
- basic system information, such as, processor, Windows user name, and system type

- processes that are working at the time of script execution.

Sample-2.tcl

```
puts "This script demonstrates the list of items to check in Tcl installation"
puts "Use this script from command line or in Capture or PSpice command shells"
puts "Script provides all the details on Tcl environment  and system details"
puts "\n\n"
global env
parray env
puts "-----------------------------------------------------------"
puts "CMD - puts ::tcl::tm::path list"
catch {puts [::tcl::tm::path list]}
puts -----------------------------------------------------------
puts "CMD - info library"
puts [info library]
puts -----------------------------------------------------------
puts {"CMD - $::auto_path"}
catch {puts $::auto_path}
puts -----------------------------------------------------------
puts {"CMD - $::tcl_pkgPath"}
catch {puts $::tcl_pkgPath}
puts -----------------------------------------------------------
puts {"CMD - $LCTLIBPATH"}
catch {puts $::env(TCLLIBPATH)}
puts -----------------------------------------------------------
puts "CMD - info load"
puts [info load]
puts -----------------------------------------------------------
puts "CMD - info patchlevel"
puts [info patchlevel]
puts -----------------------------------------------------------
puts "CMD - packages names"
foreach n [package names] {
        foreach m [package versions $n] {
        set p [package ifneeded $n $m]
        puts "$n $m $p"
        }
}
puts ---------------------------------------------------------
puts "CMD - List Packages Present with version"
foreach n [package names] {
    if { ! [catch {package present $n} v] } {
        puts "$n [package present $n]"
    }
}
puts -----------------------------------------------------------
catch {puts [exec systeminfo]}
puts -----------------------------------------------------------
catch {puts [exec tasklist "/v"]}
puts ---------------------END--------------------------------------
```

Reading and Writing CSV Files using TCL

You can use Comma Separated Values (CSV) file to export and import data in OrCAD Capture. The following TCL source,

1. Reads the first row content of a sample CSV file (`sample1.csv`)
2. Writes the content read from `sample1.csv` file to another sample CSV file (`sample2.csv`)

Sample-3.tcl

```
package require csv
package require struct::matrix
struct::matrix::matrix m
set fd [open {sample1.csv}]
csv::read2matrix $fd m , auto
set chan [open {sample2.csv} w]
::csv::writematrix m $chan
set max_rows [m rows]
set max_col [m columns]
puts "Rows-$max_rows"
puts "Col-$max_col"
for {set i 0} {$i < $max_rows} {incr i} {
    puts "Value [m get cell $i 1]"
}
m destroy
close $chan
close $fd
```

Reading and Writing SKILL or Lisp Files usingTCL

The following TCL source locates the installation hierarchy and reads the PSpice Advanced Analysis Property File (`.prp`) from a PSpice Advanced Analysis Sample.

Sample-4.tcl

```
set cdnInstallPath [exec cds_root cds_root]
set inputFile "$cdnInstallPath\\tools\\pspice\\capture_samples\\advanls\\bpf\\bpf-
PSpiceFiles\\bpf\\bpf_sch.prp"
load orCommonTcl64.dll orCommonTcl
load "$cdnInstallPath/tools/bin/orParserTcl64.dll" orParser
package require OrCommonTcl
package require orParser 1.0
package provide orLispParser 1.0
set ::printmydata 0
namespace eval ::orLispParser {
    namespace export processElement
    namespace export processElementEnd

    namespace export processElementLeaf
}
proc ::orLispParser::processElement { pElemName pElemVal } {
    puts "element $pElemName $pElemVal"
}
proc ::orLispParser::processElementEnd { } {
#    puts "ElementEnd"
}
proc ::orLispParser::processElementLeaf { pElemLeafName pElemLeafVal } {
    puts "elementleaf $pElemLeafName $pElemLeafVal"
}
proc ::orLispParser::parseLisp { filename } {
    ::orParseLisp $filename
}
puts $inputFile
::orLispParser::parseLisp $inputFile
```

You can also run the above TCL source from a standalone TCL shell. To run the TCL source in a standalone TCL Shell, enter the following commands:

1. `load <orCommonTcl64.dll path> orCommonTcl`
2. `load <orParserTcl64.dll path> orParser`
3. `source <script path>`
   For example, `source orLispParser.tcl`
4. `::orLispParser::parseLisp <filename>`
   For example, `::orLispParser::parseLisp d:/temp/bipolar.prp`

Writing JSON String using TCL

Using the following TCL source, you can load OrCAD packages from a Cadence installed hierarchy and create a JSON string using the `orPrmJSON::encode` command.

Sample-5.tcl

```
load orCommonTcl64.dll orCommonTcl
package require OrLibJSON
set JSON_NULL \0
set JSON_STRING \1
set JSON_NUMBER \2
set JSON_BOOL \3
set JSON_ARRAY \4
set JSON_NODE \5
set gNullObj {}
proc getTypeString {pType} {
    set lRet {}
    switch $pType $::JSON_NULL {
        set lRet "NULL"
    } $::JSON_STRING {
        set lRet "string"
    } $::JSON_NUMBER {
        set lRet "number"
    } $::JSON_BOOL {
        set lRet "bool"
    } $::JSON_ARRAY {
        set lRet "array"
    } $::JSON_NODE {
        set lRet "object"
    }
    return $lRet
}
proc processArray {pNode pProcessor} {
    set lSize [orjson_size $pNode]
    for {set i 0} {$i < $lSize} {incr i} {
        set lNode [orjson_at $pNode $i]
        processNode $lNode $pProcessor
    }
}
proc getValue {pNode} {
    set lValue {}
    set lType [orjson_type $pNode]
    switch $lType $::JSON_STRING {
        set lValue [orjson_as_string $pNode]
    } $::JSON_NUMBER {
        set lValue [orjson_as_int $pNode]
    } $::JSON_BOOL {
        set lValue [orjson_as_bool $pNode]
    }
    return $lValue
}
proc processNode {pNode pProcessor {pLevel 0}} {
    incr pLevel
    set lName [orjson_name $pNode]
    set lType [orjson_type $pNode]
    if { $lType == $::JSON_NODE} {
        $pProcessor $lName "" [getTypeString $lType] $pLevel
    }
    if { $lType == $::JSON_ARRAY || $lType == $::JSON_NODE } {
        set lSize [orjson_size $pNode]
        for {set i 0} {$i < $lSize} {incr i} {
            if { $lType == $::JSON_ARRAY || $lType == $::JSON_NODE } {
                processNode [orjson_at $pNode $i] $pProcessor $pLevel
            }
        }
    } else {
        $pProcessor $lName [getValue $pNode] [getTypeString $lType] $pLevel
    }
}
proc parseJSON { pInput processor } {
    set jsonNode [orjson_parse $pInput]
    if { $jsonNode == $::gNullObj || $jsonNode == $::JSON_NULL} {
        puts "Error on parsing JSON $jsonNode"
        return;
    }
    processNode $jsonNode $processor
}
```

⚠ To create a JSON string without using the orPrmJSON package:

```
set jsonStr {{"Libraries":["lib1","lib2","lib3","lib4","lib5"]}}
```

Reading JSON String using TCL

The following TCL source:

- reads a directory file list
- creates a JSON string using the `libjson` API
- parses the JSON string using `libjson` API

Sample-6.tcl

```
load orCommonTcl64.dll orCommonTcl
package require OrLibJSON
set JSON_NULL \0
set JSON_STRING \1
set JSON_NUMBER \2
set JSON_BOOL \3
set JSON_ARRAY \4
set JSON_NODE \5
set gNullObj {}
```

```
proc getTypeString {pType} {
    set lRet {}
    switch $pType $::JSON_NULL {
        set lRet "NULL"
    } $::JSON_STRING {
        set lRet "string"
    } $::JSON_NUMBER {
        set lRet "number"
    } $::JSON_BOOL {
        set lRet "bool"
    } $::JSON_ARRAY {
        set lRet "array"
    } $::JSON_NODE {
        set lRet "object"
    }
    return $lRet
}
proc processArray {pNode pProcessor} {
    set lSize [orjson_size $pNode]
    for {set i 0} {$i < $lSize} {incr i} {
        set lNode [orjson_at $pNode $i]
        processNode $lNode $pProcessor
    }
}
proc getValue {pNode} {
    set lValue {}
    set lType [orjson_type $pNode]
    switch $lType $::JSON_STRING {
        set lValue [orjson_as_string $pNode]
    } $::JSON_NUMBER {
        set lValue [orjson_as_int $pNode]
    } $::JSON_BOOL {
        set lValue [orjson_as_bool $pNode]
    }
    return $lValue
}
proc processNode {pNode pProcessor {pLevel 0}} {
    incr pLevel
    set lName [orjson_name $pNode]
    set lType [orjson_type $pNode]

    if { $lType == $::JSON_NODE} {
        $pProcessor $lName "" [getTypeString $lType] $pLevel
    }
    if { $lType == $::JSON_ARRAY || $lType == $::JSON_NODE } {
        set lSize [orjson_size $pNode]
        for {set i 0} {$i < $lSize} {incr i} {
            if { $lType == $::JSON_ARRAY || $lType == $::JSON_NODE } {
                processNode [orjson_at $pNode $i] $pProcessor $pLevel
            }
        }
    } else {
        $pProcessor $lName [getValue $pNode] [getTypeString $lType] $pLevel
    }
}
proc parseJSON { pInput processor } {
    set jsonNode [orjson_parse $pInput]
    if { $jsonNode == $::gNullObj || $jsonNode == $::JSON_NULL} {
        puts "Error on parsing JSON $jsonNode"
        return;
    }
    processNode $jsonNode $processor
}
proc getLibrariesCallback {pName pValue pType pLevel} {
    puts "[string repeat "-" $pLevel] Name: $pName Value: $pValue  Type:$pType"
}
proc getLibraries { pDirList} {
    set root [orjson_new $::JSON_NODE]
    set jsonObj [orjson_new $::JSON_NODE]
    orjson_set_name $jsonObj "Libraries"
    orjson_push_back $root $jsonObj
    set jsonArray [orjson_new $::JSON_ARRAY]
    orjson_push_back $jsonObj $jsonArray
    foreach {mbr} $pDirList {
        set jsonObj [orjson_new $::JSON_NODE]
        orjson_set_a $jsonObj $mbr
        orjson_push_back $jsonArray $jsonObj
    }
    set jsontext [orjson_write $root]
    puts "JSON String output for directories listing: \n $jsontext\n\n"
    parseJSON $jsontext getLibrariesCallback
}
getLibraries [glob -directory [file normalize [exec cds_root cds_root]/tools/capture/library]
*.olb]
```

4

TCL Programming with OrCAD Capture Database

This chapter lists various TCL sample source files that access OrCAD Capture Database to do the following tasks:

- Set up TCL packages to access OrCAD Capture Database
- Access library database
- Access design file pages
- Access design hierarchy and display properties, such as, reference designators
- Access cache

Creating a Capture Session and Opening a Design

A specific API is required to read, write, or modify a Capture database because it is a compound binary file.

The Sample TCL source file:

- loads a package that allows access to OrCAD Capture database
- uses of a helper function to interface with the Microsoft Foundation Class (MFC) CString class
- creates a Session
- opens a database

Sample-7.tcl

```
set cdnInstallPath [exec cds_root cds_root]
load [file normalize [file join $cdnInstallPath/tools/capture orDb_Dll_TCL]] DboTclWriteBasic

set lDesignName [file normalize [file join $cdnInstallPath/tools/capture/samples fulladd.dsn]]

set lStatus [DboState]

puts "Creating OrCAD Database Session"
set lSession [DboTclHelper_sCreateSession]

puts "Using Helper to create database name CString"
set lDesignNameCStr [DboTclHelper_sMakeCString $lDesignName]

puts "Open the design database"
set lDesign [$lSession GetDesignAndSchematics $lDesignNameCStr $lStatus]

if { "NULL" != $lDesign} {
 puts "$lDesignName is open"
 $lSession RemoveLib $lDesign
} else {
 puts "$lDesignName could not open"
}

DboTclHelper_sDeleteSession $lSession
$lStatus -delete
```

Opening Library and Listing Packages

OrCAD Capture Database has container hierarchies, such as libraries, accessible using iterators. This sample TCL source file shows how iterators are used to iterate on container hierarchies.

The sample TCL source file:

- opens a library
- uses Iterators to list objects in the library

Sample-8.tcl

```
set cdnInstallPath [exec cds_root cds_root]
load [file normalize [file join $cdnInstallPath/tools/capture orDb_Dll_TCL]] DboTclWriteBasic

set lLibraryName [file normalize [file join $cdnInstallPath/tools/capture/library
MiscLinear.olb]]

set lStatus [DboState]
set errorMsgCStr [DboTclHelper_sMakeCString ""]
set lNameCStr [DboTclHelper_sMakeCString]

puts "Creating OrCAD Database Session"
set lSession [DboTclHelper_sCreateSession]
puts "Using Helper to create database name CString"
set lLibraryNameCStr [DboTclHelper_sMakeCString $lLibraryName]

puts "Open the Library database"
set lLibrary [$lSession GetLib $lLibraryNameCStr $lStatus]

if { [$lStatus Failed] } {
 $lStatus Message $errorMsgCStr
 puts ">> Can't open $lLibraryName \n[DboTclHelper_sGetConstCharPtr $errorMsgCStr]\n\n"
 exit
} else {
 puts "$lLibraryName is open"
}

puts "Creating Library Packages Iterator"
set lIter [$lLibrary NewPackagesIter $lStatus]
set lObj [$lIter Next $lStatus]
while {$lObj!="NULL"} {
 $lObj GetName $lNameCStr
 set lSize [DboPackage_sGetSize $lObj $lStatus]
 puts "Package [DboTclHelper_sGetConstCharPtr $lNameCStr] of Size $lSize"
 set lObj [$lIter Next $lStatus]
}
puts "Deleting Packages Iterator"
delete_DboLibPackagesIter $lIter

puts "Creating Library Parts Iterator"
set lLibPartIter [$lLibrary NewPartsIter $lStatus]
set lLibPart [$lLibPartIter NextPart $lStatus]
while {$lLibPart!="NULL"} {
 $lLibPart GetName $lNameCStr
 puts "Part [DboTclHelper_sGetConstCharPtr $lNameCStr]"
 set lLibPart [$lLibPartIter NextPart $lStatus]
}
puts "Deleting Parts Iterator"
delete_DboLibPartsIter $lLibPartIter

puts "Creating Library Symbols Iterator"
set lIter [$lLibrary NewSymbolsIter $lStatus]
set lObj [$lIter Next $lStatus]
while {$lObj!="NULL"} {
 $lObj GetName $lNameCStr
 puts "Symbol [DboTclHelper_sGetConstCharPtr $lNameCStr]"
 set lObj [$lIter Next $lStatus]
}
puts "Deleting Symbols Iterator"
delete_DboLibSymbolsIter $lIter

puts "Creating Library Cells Iterator"
set lIter [$lLibrary NewCellsIter $lStatus]
set lObj [$lIter Next $lStatus]
while {$lObj!="NULL"} {
 $lObj GetName $lNameCStr
 puts "Cells [DboTclHelper_sGetConstCharPtr $lNameCStr]"
 set lObj [$lIter Next $lStatus]
}
puts "Deleting Cells Iterator"
delete_DboLibCellsIter $lIter

puts "Closing $lLibraryName"
$lSession RemoveLib $lLibrary
puts "Closing Session"
DboTclHelper_sDeleteSession $lSession
```

Following Class (IsA) Hierarchies

Sometimes a package contains more than one devices. The `NewDevicesIter` class can be used to iterate over devices in a package. This sample TCL source file:

- uses iterators to list devices in a package
- demonstrates how to work when iterator returns a base class object instead of the object

Sample-10.tcl

```
set cdnInstallPath [exec cds_root cds_root]
load [file normalize [file join $cdnInstallPath/tools/capture orDb_Dll_TCL]] DboTclWriteBasic
set lLibraryName [file normalize [file join $cdnInstallPath/tools/capture/library
MiscLinear.olb]]
set lStatus [DboState]
set errorMsgCStr [DboTclHelper_sMakeCString ""]
set lNameCStr [DboTclHelper_sMakeCString]
puts "Creating OrCAD Database Session"
set lSession [DboTclHelper_sCreateSession]
puts "Using Helper to create database name CString"
set lLibraryNameCStr [DboTclHelper_sMakeCString $lLibraryName]
puts "Open the Library database"
set lLibrary [$lSession GetLib $lLibraryNameCStr $lStatus]
if { [$lStatus Failed] } {
 $lStatus Message $errorMsgCStr
 puts ">> Can't open $lLibraryName \n[DboTclHelper_sGetConstCharPtr $errorMsgCStr]\n\n"
 exit
} else {
 puts "$lLibraryName is open"
}
puts "Creating Library Packages Iterator"
set lIter [$lLibrary NewPackagesIter $lStatus]
set lObj [$lIter Next $lStatus]
```

```
while {$lObj!="NULL"} {
 $lObj GetName $lNameCStr
 set lSize [DboPackage_sGetSize $lObj $lStatus]
 puts "Package [DboTclHelper_sGetConstCharPtr $lNameCStr] of Size $lSize"
 set lObj [DboBaseObjectToDboLibObject $lObj]
 set lObj [DboLibObjectToDboPackage $lObj]
 set lIter2 [$lObj NewDevicesIter $lStatus]
 set lObj2 [$lIter2 NextDevice $lStatus]
 while {$lObj2!="NULL"} {
  $lObj2 GetName $lNameCStr
  puts "\tDevice [DboTclHelper_sGetConstCharPtr $lNameCStr]"
  set lObj2 [$lIter2 NextDevice $lStatus]
 }
 delete_DboPackageDevicesIter $lIter2
 set lObj [$lIter Next $lStatus]
}
puts "Deleting Packages Iterator"
delete_DboLibPackagesIter $lIter
puts "Creating Library Parts Iterator"
set lLibPartIter [$lLibrary NewPartsIter $lStatus]
set lLibPart [$lLibPartIter NextPart $lStatus]
while {$lLibPart!="NULL"} {
 $lLibPart GetName $lNameCStr
 puts "Part [DboTclHelper_sGetConstCharPtr $lNameCStr]"
 set lLibPart [$lLibPartIter NextPart $lStatus]
}
puts "Deleting Parts Iterator"
delete_DboLibPartsIter $lLibPartIter
puts "Creating Library Symbols Iterator"
set lIter [$lLibrary NewSymbolsIter $lStatus]
set lObj [$lIter Next $lStatus]
while {$lObj!="NULL"} {
 $lObj GetName $lNameCStr
 puts "Symbol [DboTclHelper_sGetConstCharPtr $lNameCStr]"
 set lObj [$lIter Next $lStatus]
}
puts "Deleting Symbols Iterator"
delete_DboLibSymbolsIter $lIter
puts "Creating Library Cells Iterator"
set lIter [$lLibrary NewCellsIter $lStatus]
set lObj [$lIter Next $lStatus]
while {$lObj!="NULL"} {
 $lObj GetName $lNameCStr
 puts "Cells [DboTclHelper_sGetConstCharPtr $lNameCStr]"
 set lObj [$lIter Next $lStatus]
}
puts "Deleting Cells Iterator"
delete_DboLibCellsIter $lIter
puts "Closing $lLibraryName"
$lSession RemoveLib $lLibrary
puts "Closing Session"
DboTclHelper_sDeleteSession $lSession
```

Traversing Design Object Instances

This sample TCL source file:

- reads the design from the `.dsn` file
- parses the schematic and page objects
- prints the properties on `DranInst` objects, that is, hierarchical blocks

Sample-11.tcl

```
set cdnInstallPath [exec cds_root cds_root]
load [file normalize [file join $cdnInstallPath/tools/capture orDb_Dll_TCL]] DboTclWriteBasic

namespace eval ::capPrintHierarchy {
 namespace export PrintHBlocks
}

proc ::capPrintHierarchy::PrintHBlocks { pFile pDesignName } {
 set lStatus [DboState]
 set lSession [DboTclHelper_sCreateSession]
 capVisitDesign $pFile $lSession $pDesignName $lStatus
 return
}

proc ::capPrintHierarchy::capVisitDesign {pFile pSession pDesignName pStatus} {
 set lDesignName [DboTclHelper_sMakeCString $pDesignName]
 #object call $pSession GetDesignAndSchematics $lDesignName $lStatus
 set lDesign [$pSession GetDesignAndSchematics $lDesignName $pStatus]
 set varNullObj NULL
 if { $lDesign == $varNullObj } {
  puts "Could Not Open design $pDesignName"
  return
  }
 puts $pFile "DSN::$pDesignName"
 capVisitSchematics $pFile $lDesign $pStatus
 $pSession RemoveLib $lDesign
 return
}
proc ::capPrintHierarchy::capVisitSchematics {pFile pDesign pStatus} {
 set lDesignNameCStr [DboTclHelper_sMakeCString]
 $pDesign GetName $lDesignNameCStr
 set lAbsoluteDesignPath [DboTclHelper_sGetConstCharPtr $lDesignNameCStr]

#create iterator of Schematic Type View
  set lSchematicIter [$pDesign NewViewsIter $pStatus $::IterDefs_SCHEMATICS]
  set lSchematicView [$lSchematicIter NextView  $pStatus]
  set lNullObj NULL
  set lSchematicName [DboTclHelper_sMakeCString]
  while { $lSchematicView!= $lNullObj} {
  set lSchematic [DboViewToDboSchematic $lSchematicView]
```

```
    set lSchematic [DboViewToDboSchematic $lSchematicView]
    $lSchematic GetName $lSchematicName
    puts $pFile "\tDSN::Sch::$lAbsoluteDesignPath->[DboTclHelper_sGetConstCharPtr $lSchematicName]"
    capVisitSchematicPages $pFile $pDesign $lSchematic $lSchematicName $pStatus
    set lSchematicView [$lSchematicIter NextView  $pStatus]
    }
    delete_DboLibViewsIter $lSchematicIter
    return
}
proc ::capPrintHierarchy::capVisitSchematicPages {pFile pDesign pSchematic pSchematicName
pStatus} {
 set lPagesIter [$pSchematic NewPagesIter $pStatus]
 set lPage [$lPagesIter NextPage $pStatus]
 set lNullObj NULL
 set lPageName [DboTclHelper_sMakeCString]
 while {$lPage!=$lNullObj} {
  $lPage GetName $lPageName
  puts $pFile "\t\tPage::[DboTclHelper_sGetConstCharPtr $lPageName]"
  capVisitHBlocksOnPages  $pFile $pDesign $pSchematic $pSchematicName $lPage $lPageName $pStatus
  set lPage [$lPagesIter NextPage $pStatus]
 }
 delete_DboSchematicPagesIter $lPagesIter
 return
}
proc ::capPrintHierarchy::capVisitHBlocksOnPages {pFile pDesign pSchematic pSchematicName pPage
pPageName pStatus} {
 set lPartInstsIter [$pPage NewPartInstsIter $pStatus]
 set lInst [$lPartInstsIter NextPartInst $pStatus]
 set lName [DboTclHelper_sMakeCString {Name}]
 set lNameValue [DboTclHelper_sMakeCString]
 set lRef [DboTclHelper_sMakeCString {Part Reference}]
 set lRefValue [DboTclHelper_sMakeCString]
 set lSrcLib [DboTclHelper_sMakeCString {Source Library}]
 set lSrcLibValue [DboTclHelper_sMakeCString]
 set lSrcPkg [DboTclHelper_sMakeCString {Source Package}]
 set lSrcPkgValue [DboTclHelper_sMakeCString]
 set lImplementationName [DboTclHelper_sMakeCString {Implementation}]
  set lImplementationNameValue [DboTclHelper_sMakeCString]
 set lImplementationPath [DboTclHelper_sMakeCString {Implementation Path}]
 set lImplementationPathValue [DboTclHelper_sMakeCString]
 set lImplementationType [DboTclHelper_sMakeCString {Implementation Type}]
  set lImplementationTypeValue [DboTclHelper_sMakeCString]
    set lNullObj NULL
 set lDesignName [DboTclHelper_sMakeCString]
 $pDesign GetName $lDesignName
 while {$lInst!=$lNullObj} {

    #dynamic cast from DboPartInst to DboDrawnInst
    set lDrawnInst [DboPartInstToDboDrawnInst $lInst]

    if {$lDrawnInst != $lNullObj} {

  $lInst GetEffectivePropStringValue $lName $lNameValue
     $lInst GetEffectivePropStringValue $lImplementationName $lImplementationNameValue
     $lInst GetEffectivePropStringValue $lImplementationPath $lImplementationPathValue
     $lInst GetEffectivePropStringValue $lImplementationType $lImplementationTypeValue

     puts $pFile "\t\t\tHBlock::[DboTclHelper_sGetConstCharPtr $lNameValue]"
      puts $pFile "\t\t\t\tPath->[DboTclHelper_sGetConstCharPtr $lDesignName]::
[DboTclHelper_sGetConstCharPtr $lImplementationPathValue]::[DboTclHelper_sGetConstCharPtr
$lImplementationNameValue]"
      puts $pFile "\t\t\t\tType::[DboTclHelper_sGetConstCharPtr $lImplementationTypeValue]"
      puts $pFile "\t\t\t\tName::[DboTclHelper_sGetConstCharPtr $lImplementationNameValue]"
    }  else   {
   $lInst GetEffectivePropStringValue $lRef $lRefValue
   puts $pFile "\t\t\tPart::[DboTclHelper_sGetConstCharPtr $lRefValue]"
   $lInst GetEffectivePropStringValue $lSrcLib $lSrcLibValue
   $lInst GetEffectivePropStringValue $lSrcPkg $lSrcPkgValue
   puts $pFile "\t\t\t\tPackage::[DboTclHelper_sGetConstCharPtr $lSrcPkgValue]"
   puts $pFile "\t\t\t\tLib::[DboTclHelper_sGetConstCharPtr $lSrcLibValue]"
   }

    #get the next part inst
    set lInst [$lPartInstsIter NextPartInst $pStatus]
 }

   delete_DboPagePartInstsIter $lPartInstsIter
   return
}
::capPrintHierarchy::PrintHBlocks stdout "$cdnInstallPath\\tools\\capture\\samples\\FULLAdd.dsn"
```

Traversing Design Object Occurrences

Capture automatically calculates occurrence connectivity from the instance objects. For each object instance, there is at least one occurrence. If any instance has more than one occurrence, the design is said to be in occurrence mode (or complex hierarchy).

The following sample TCL source file:

- uses iterators recursively to list design objects
- traverses design hierarchy using the `NewChildrenIter` iterator

Sample-12.tcl

```
set cdnInstallPath [exec cds_root cds_root]
load [file normalize [file join $cdnInstallPath/tools/capture orDb_Dll_TCL]] DboTclWriteBasic

proc capEnumerateInstOccurrence { pDesign pInstOcc pLevel} {
 set lStatus [DboState]
 set lPathName [DboTclHelper_sMakeCString]
 set lName [DboTclHelper_sMakeCString]
 set pLevel [expr $pLevel + 5]

 set lOffPageOccIter [$pInstOcc NewChildrenIter $lStatus  $::IterDefs_OFFPAGES]
 $lOffPageOccIter Sort $lStatus
 set lOffPageOcc [$lOffPageOccIter NextOccurrence $lStatus]
 while { $lOffPageOcc!= "NULL"} {
  set lId [$lOffPageOcc GetId $lStatus]
  $lOffPageOcc GetPathName $lPathName
  set lPathNameChar [DboTclHelper_sGetConstCharPtr $lPathName]
  set lOccId [$lOffPageOcc GetId $lStatus]
  puts "[string repeat "-" $pLevel] OffPageOcc:: $lOccId $lPathNameChar "
  set lOffPageOcc [$lOffPageOccIter NextOccurrence $lStatus]
 }
 delete_DboOccurrenceChildrenIter $lOffPageOccIter

 set lPortOccIter [$pInstOcc NewChildrenIter $lStatus  $::IterDefs_PORTS]
 $lPortOccIter Sort $lStatus
 set lPortOcc [$lPortOccIter NextOccurrence $lStatus]
 while { $lPortOcc!= "NULL"} {
  set lId [$lPortOcc GetId $lStatus]
  $lPortOcc GetPathName $lPathName
  set lPathNameChar [DboTclHelper_sGetConstCharPtr $lPathName]
  set lOccId [$lPortOcc  GetId $lStatus]
  puts "[string repeat "-" $pLevel] PortOcc:: $lOccId $lPathNameChar "
  set lPortOcc [$lPortOccIter NextOccurrence $lStatus]
 }
 delete_DboOccurrenceChildrenIter $lPortOccIter

 set lNetOccIter [$pInstOcc NewChildrenIter $lStatus  $::IterDefs_NETS]
 $lNetOccIter Sort $lStatus
 set lNetOcc [$lNetOccIter NextOccurrence $lStatus]
 while { $lNetOcc!= "NULL"} {
  set lId [$lNetOcc GetId $lStatus]
  $lNetOcc GetPathName $lPathName
  set lPathNameChar [DboTclHelper_sGetConstCharPtr $lPathName]
  set lOccId [$lNetOcc GetId $lStatus]
  puts "[string repeat "-" $pLevel] NetOcc:: $lOccId $lPathNameChar "
  set lNetOcc [$lNetOccIter NextOccurrence $lStatus]
 }
 delete_DboOccurrenceChildrenIter $lNetOccIter

 set lTitleBlockOccIter [$pInstOcc NewChildrenIter $lStatus  $::IterDefs_TITLEBLOCKS]
 $lTitleBlockOccIter Sort $lStatus
 set lTitleBlockOcc [$lTitleBlockOccIter NextOccurrence $lStatus]
 while { $lTitleBlockOcc!= "NULL"} {
  set lId [$lTitleBlockOcc GetId $lStatus]
  $lTitleBlockOcc GetPathName $lPathName
  set lPathNameChar [DboTclHelper_sGetConstCharPtr $lPathName]
  set lOccId [$lTitleBlockOcc GetId $lStatus]
  puts "[string repeat "-" $pLevel] TitleblockOcc:: $lOccId $lPathNameChar "
  set lTitleBlockOcc [$lTitleBlockOccIter NextOccurrence $lStatus]
 }
 delete_DboOccurrenceChildrenIter $lTitleBlockOccIter
```

```
 set lInstOccIter [$pInstOcc NewChildrenIter $lStatus  $::IterDefs_INSTS]
 $lInstOccIter Sort $lStatus
 set lChildOcc [$lInstOccIter NextOccurrence $lStatus]
 while { $lChildOcc!= "NULL"} {
  set lId [$lChildOcc GetId $lStatus]
  set lInstOcc [DboOccurrenceToDboInstOccurrence $lChildOcc]
  $lInstOcc GetReferenceDesignator $lName
  set lInstOccRefDes [DboTclHelper_sGetConstCharPtr $lName]
  set lOccId [$lInstOcc GetId $lStatus]
  puts "[string repeat "-" $pLevel] InstOcc:: $lOccId $lInstOccRefDes "
  capEnumerateInstOccurrence $pDesign $lInstOcc $pLevel
  set lChildOcc [$lInstOccIter NextOccurrence $lStatus]
 }
 delete_DboOccurrenceChildrenIter $lInstOccIter
}
proc capEnumerateOccurrences { pDesign lStatus} {
 set lRootOcc [$pDesign GetRootOccurrence $lStatus]
 set lName [DboTclHelper_sMakeCString]
 set lStatus [$pDesign GetRootName $lName]
 set lOccId [$lRootOcc GetId $lStatus]
 puts "Root: [DboTclHelper_sGetConstCharPtr $lName] Occurrence Id: $lOccId"
 capEnumerateInstOccurrence $pDesign $lRootOcc 0
 $lStatus -delete
}
proc capShowOccTree { pDesignName } {
 set lStatus [DboState]
 set lSession [DboTclHelper_sCreateSession]
 set lDesignNameCStr [DboTclHelper_sMakeCString $pDesignName]
 set lDesign [$lSession GetDesignAndSchematics $lDesignNameCStr $lStatus]
 if { "NULL" != $lDesign} {
  puts "$pDesignName is open"
  capEnumerateOccurrences $lDesign $lStatus
  $lSession RemoveLib $lDesign
 } else {
  puts "$pDesignName could not open"
 }
 DboTclHelper_sDeleteSession $lSession
 $lStatus -delete
}
set lDesignName [file normalize [file join $cdnInstallPath/tools/capture/samples fulladd.dsn]]

capShowOccTree $lDesignName
```

Listing Cache with Date or Time

OrCAD Capture design database caches the library objects. Using TCL, you can list the cache object data.

The following sample TCL source file traverses the cache and displays time stamp.

Sample-13.tcl

```
set cdnInstallPath [exec cds_root cds_root]
load [file normalize [file join $cdnInstallPath/tools/capture orDb_Dll_TCL]] DboTclWriteBasic
proc printObjectType { pObject } {
  set lCStr [DboTclHelper_sMakeCString]
  $pObject GetTypeString $lCStr
  set lObjTypeStr [DboTclHelper_sGetConstCharPtr $lCStr]
  set lObjType [$pObject GetObjectType]
  switch  $lObjType {
   0 { puts "NULL_OBJECT" }
   1 { puts "BASE_OBJECT" }
   2 { puts "GRAPHIC_OBJECT" }
   3 { puts "NAMED_OBJECT" }
   4 { puts "DESIGN" }
   5 { puts "LIBRARY" }
   6 { puts "PART_CELL" }
   7 { puts "VIEW" }
   8 { puts "EXTERNAL_VIEW" }
   9 { puts "SCHEMATIC" }
   10 { puts "PAGE" }
   11 { puts "PART_INSTANCE" }
   12 { puts "DRAWN_INSTANCE" }
   13 { puts "PLACED_INSTANCE" }
   14 { puts "TEMPLATE_INSTANCE" }
   15 { puts "PORT_INSTANCE" }
   16 { puts "PORT_INSTANCE_SCALAR" }
   17 { puts "PORT_INSTANCE_BUS" }
   18 { puts "PORT_INSTANCE_BUNDLE" }
   19 { puts "WIRE" }
   20 { puts "WIRE_SCALAR" }
   21 { puts "WIRE_BUS" }
   22 { puts "WIRE_BUNDLE" }
   23 { puts "PORT" }
   24 { puts "LIBRARY_PART" }
   25 { puts "SYMBOL_PIN" }
   26 { puts "SYMBOL_PIN_SCALAR" }
   27 { puts "SYMBOL_PIN_BUS" }
   28 { puts "SYMBOL_PIN_BUNDLE" }
   29 { puts "ENTRY" }
   30 { puts "COMMENT" }
   31 { puts "PACKAGE" }
   32 { puts "DEVICE" }
   33 { puts "GLOBAL_SYMBOL" }
   34 { puts "PORT_SYMBOL" }
   35 { puts "OFF_PAGE_SYMBOL" }
   36 { puts "EXPORT_BLOCK" }
   37 { puts "DBGLOBAL" }
   38 { puts "OFF_PAGE_CONNECTOR" }
   39 { puts "DISPLAY_PROP" }
   40 { puts "BOX" }
   41 { puts "LINE" }
   42 { puts "ARC" }
   43 { puts "ELLIPSE" }
   44 { puts "POLYGON" }
   45 { puts "POLYLINE" }
   46 { puts "COMMENT_TEXT" }
   47 { puts "BITMAP_VECT" }
   48 { puts "SYMBOL_VECT" }
   49 { puts "ALIAS" }
   50 { puts "NETSYMBOL_INSTANCE" }
   51 { puts "NET" }
   52 { puts "NET_SCALAR" }
   53 { puts "NET_BUS" }
   54 { puts "GRAPHIC_INSTANCE" }
   55 { puts "GRAPHIC_BOX_INST" }
   56 { puts "GRAPHIC_LINE_INST" }
   57 { puts "GRAPHIC_ARC_INST" }
   58 { puts "GRAPHIC_ELLIPSE_INST" }
   59 { puts "GRAPHIC_POLYGON_INST" }
   60 { puts "GRAPHIC_POLYLINE_INST" }
   61 { puts "GRAPHIC_COMMENTTEXT_INST" }
   62 { puts "GRAPHIC_BITMAP_INST" }
   63 { puts "GRAPHIC_SYMBOL_VECTOR_INST" }
   64 { puts "TITLEBLOCK_SYMBOL" }
   65 { puts "TITLEBLOCK_INSTANCE" }
   66 { puts "INST_OCCURRENCE" }
   67 { puts "NET_OCCURRENCE" }
   68 { puts "PORT_OCCURRENCE" }
   69 { puts "PORT_BUS_MEMBER_OCCURRENCE" }
   70 { puts "SCHEMATIC_INSTANCE" }
   71 { puts "SCHEMATIC_NET" }
   72 { puts "SCHEMATIC_NET_SCALAR" }
   73 { puts "SCHEMATIC_NET_BUS" }
   74 { puts "FLAT_NET" }
   75 { puts "ERC_SYMBOL" }
   76 { puts "BOOK_MARK_SYMBOL" }
   77 { puts "ERC_OBJECT" }
   78 { puts "BOOK_MARK" }
   79 { puts "SCHEMATIC_PORT" }
   80 { puts "SCHEMATIC_GLOBAL" }
   81 { puts "SCHEMATIC_OFFPAGE_CONNECTOR" }
   82 { puts "TITLEBLOCK_OCCURRENCE" }
   83 { puts "FILL" }
   84 { puts "PORT_INSTANCE_BUS_MEMBER" }
   85 { puts "WIRE_JUNCTION" }
   86 { puts "API_OBJ_PART" }
   87 { puts "BEZIER" }
   88 { puts "GRAPHIC_BEZIER_INST" }
   89 { puts "GRAPHIC_OLEEMBED_INST" }
   90 { puts "OLEEMBED_VECT" }
   91 { puts "OFF_PAGE_CONNECTOR_OCCURRENCE" }
   97 { puts "CUSTOMITEM_INSTANCE" }
   98 { puts "PIN_VECTOR" }
   104 { puts "NET_BUNDLE" }
   default { puts "$lObjTypeStr" }
  }
}
proc showCacheEntries { pDesign } {
    set lCacheNameCStr [DboTclHelper_sMakeCString]
    set lCacheLibNameCStr [DboTclHelper_sMakeCString]
    set lStatus [DboState]
 set lCacheObjectsIter [$pDesign NewCachesIter $lStatus $::IterDefs_ALL]
    set lCachedObject [$lCacheObjectsIter NextCachedObject  $lStatus]
    while { $lCachedObject!= "NULL" } {
  set lCachedLibObject [DboBaseObjectToDboLibObject $lCachedObject]
  $lCachedObject GetName $lCacheNameCStr
  $pDesign GetSourceLibName $lCacheNameCStr $lCachedLibObject $lCacheLibNameCStr
  puts -nonewline "Cache Entry - Library [DboTclHelper_sGetConstCharPtr $lCacheLibNameCStr] : "
  puts -nonewline "[DboTclHelper_sGetConstCharPtr $lCacheNameCStr] : "
  set lObjectType [$lCachedObject GetObjectType]
  set lIsOutOfDate [$pDesign CacheIsOutOfDate $lObjectType $lCacheLibNameCStr $lCacheNameCStr
$lStatus]
  puts -nonewline "OutOfDate-$lIsOutOfDate : "
  set lCacheTime [$pDesign GetCachedTime $lObjectType $lCacheLibNameCStr $lCacheNameCStr
$lStatus]
  set lCacheTimeStr [DboTclHelper_sGetConstCharPtr [$pDesign TimeToString $lCacheTime]]
  puts -nonewline "$lCacheTimeStr : "

  printObjectType $lCachedLibObject
  set lCachedObject [$lCacheObjectsIter NextCachedObject  $lStatus]
    }
    delete_DboDesignCachesIter $lCacheObjectsIter
    $lStatus -delete
}
proc capShowCacheData { pDesignName } {
 set lStatus [DboState]
 set lSession [DboTclHelper_sCreateSession]
 set lDesignNameCStr [DboTclHelper_sMakeCString $pDesignName]
 set lDesign [$lSession GetDesignAndSchematics $lDesignNameCStr $lStatus]
 if { "NULL" != $lDesign} {
  puts "$pDesignName is open"
  showCacheEntries $lDesign
  $lSession RemoveLib $lDesign
 } else {
  puts "$pDesignName could not open"
 }
 DboTclHelper_sDeleteSession $lSession
 $lStatus -delete
}
set lDesignName [file normalize [file join $cdnInstallPath/tools/capture/samples fulladd.dsn]]
capShowCacheData $lDesignName
```

Generating Flat Nets List

OrCAD Capture generates flat nets that are mapped to the physical nets of the PCB tools.

The following sample TCL source file lists the flat nets in the Full Adder design.

Sample-14.tcl

```
set cdnInstallPath [exec cds_root cds_root]
load [file normalize [file join $cdnInstallPath/tools/capture orDb_Dll_TCL]] DboTclWriteBasic
proc listFlatNet { pDesignName } {
 set lStatus [DboState]
 set lSession [DboTclHelper_sCreateSession]
 set lDesignNameCStr [DboTclHelper_sMakeCString $pDesignName]
 set lDesign [$lSession GetDesignAndSchematics $lDesignNameCStr $lStatus]
 if { "NULL" == $lDesign} {
  puts "$pDesignName could not open"
  exit
 }
 set lRoot [$lDesign GetRootOccurrence $lStatus]
 set lNameCStr [DboTclHelper_sMakeCString]
 set lFlatNetsIter [$lDesign NewFlatNetsIter $lStatus]
 set lFlatNet [$lFlatNetsIter NextFlatNet $lStatus]
 while {$lFlatNet != "NULL"} {
  $lFlatNet GetName $lNameCStr
  puts [DboTclHelper_sGetConstCharPtr $lNameCStr]
  set lFlatNet [$lFlatNetsIter NextFlatNet $lStatus]
 }
 $lFlatNetsIter -delete
 $lStatus -delete
 $lSession RemoveLib $lDesign
 DboTclHelper_sDeleteSession $lSession
}
set lDesignName [file normalize [file join $cdnInstallPath/tools/capture/samples fulladd.dsn]]
listFlatNet $lDesignName
```

⚠ When working on command line access – Occurrences (and Flat Net) data is only available after the call to GetRootOccurrence

Iterating Object properties

The *listEffectiveProperties {pObject}* function is used to iterate over properties of any objects.

The following sample TCL source file lists properties of each of the flat nets in the *Full Adder* design

Sample-15.tcl

```
set cdnInstallPath [exec cds_root cds_root]
load [file normalize [file join $cdnInstallPath/tools/capture orDb_Dll_TCL]] DboTclWriteBasic
proc listEffectiveProperties {pObject} {
 set lStatus [DboState]
 set cptr DboTclHelper_sGetConstCharPtr
 set lPropsIter [$pObject NewEffectivePropsIter $lStatus]
 #create the input/output parameters
 set lPrpName [DboTclHelper_sMakeCString]
 set lPrpValue [DboTclHelper_sMakeCString]
 set lPrpType [DboTclHelper_sMakeDboValueType]
 set lEditable [DboTclHelper_sMakeInt]
 set lStatus [$lPropsIter NextEffectiveProp $lPrpName $lPrpValue $lPrpType $lEditable]
 while {[$lStatus OK] == 1} {
  puts "\tName=[$cptr $lPrpName]  Value= [$cptr $lPrpValue] "
  set lStatus [$lPropsIter NextEffectiveProp $lPrpName $lPrpValue $lPrpType $lEditable]
 }
 delete_DboEffectivePropsIter $lPropsIter
 $lStatus -delete
}
proc listFlatNet { pDesignName } {
 set lStatus [DboState]
 set lSession [DboTclHelper_sCreateSession]
 set lDesignNameCStr [DboTclHelper_sMakeCString $pDesignName]
 set lDesign [$lSession GetDesignAndSchematics $lDesignNameCStr $lStatus]
 if { "NULL" == $lDesign} {
  puts "$pDesignName could not open"
  exit
 }
 set lRoot [$lDesign GetRootOccurrence $lStatus]
 set lNameCStr [DboTclHelper_sMakeCString]
 set lFlatNetsIter [$lDesign NewFlatNetsIter $lStatus]
 set lFlatNet [$lFlatNetsIter NextFlatNet $lStatus]
 while {$lFlatNet != "NULL"} {
  $lFlatNet GetName $lNameCStr
  puts [DboTclHelper_sGetConstCharPtr $lNameCStr]
  listEffectiveProperties $lFlatNet
  set lFlatNet [$lFlatNetsIter NextFlatNet $lStatus]
 }
 $lFlatNetsIter -delete
 $lStatus -delete
 $lSession RemoveLib $lDesign
 DboTclHelper_sDeleteSession $lSession
}
set lDesignName [file normalize [file join $cdnInstallPath/tools/capture/samples fulladd.dsn]]
listFlatNet $lDesignName
```

5

TCL Programmimg with OrCAD Capture GUI Interface

This chapter describes the following:

- Adding menus and installing packages
- Running design modification with generated code
- Generating TCL code for an example design
- Working interactively with selected objects
- Modifying properties on a design hierarchy

Adding Menu to OrCAD Capture

The following sample TCL source file:

- adds a menu to the main, schematic, and project manager windows of Capture
- sets up the TCL scripts loaded on start up

Do the following steps to run the sample TCL source file:

1. Add the following TCL source file at `<installation>\tools\capture\tclscripts\capAutoLoad`.

Sample-16.tcl

```
 namespace eval ::test {
    proc registerMenuActions { args } {
        catch {

            RegisterAction "testMenu_PM_1"   "::test::shouldProcess" "Alt+F"
"::test::testMePM" "PM"
            RegisterAction "testMenu_PM_2"   "::test::shouldProcess" "Alt+F1"
"::test::testMePM" "PM"
            RegisterAction "testMenu_PM_3"   "::test::shouldProcess" "Alt+F2"
"::test::testMePM" "PM"
            RegisterAction "testMenu_PM_4"   "::test::shouldProcess" "Shift+Z"
"::test::testMePM" "PM"

            RegisterAction "testMenu_SCM_1"   "::test::shouldProcess" "Alt+F"
"::test::testMeSCM" "SCHEMATIC"
            RegisterAction "testMenu_SCM_2"   "::test::shouldProcess" "Alt+F1"
"::test::testMeSCM" "SHEMATIC"
            RegisterAction "testMenu_SCM_3"   "::test::shouldProcess" "Alt+F2"
"::test::testMeSCM" "SCHEMATIC"
            RegisterAction "testMenu_SCM_4"   "::test::shouldProcess" "Shift+Z"
"::test::testMeSCM" "SCHEMATIC"
            InsertXMLMenu  [list [list "MyFile1"] "" "" [list "popup" "MyFileTest1"  "" ""
"" "" ""] ""]
            InsertXMLMenu  [list [list "MyFile1" "MyFileLevelpopup1"] "" "" [list "popup"
"MyFileLevel11"  "" "" "" "" ""] ""]
            InsertXMLMenu  [list [list "MyFile1" "MyFileLevelpopup1" "MyFileLevelAction2"]
"" "" [list "action" "&menu1"  "0" "ActionForMenu1" "UpdateForMenu1" "Ctrl+Z" "" "" "This
is my menu test2"] ""]
            InsertXMLMenu  [list [list "MyFile1" "MyFileLevelpopup1" "MyFileLevelAction3"]
"" "" [list "action" "&menu2"  "0" "ActionForMenu1" "UpdateForMenu1" "Shift+B" "" "" "This
is my menu test2"] ""]
            InsertXMLMenu  [list [list "MyFile1" "MyFileLevelpopup1" "MyFileLevelAction4"]
"" "" [list "action" "&menu3"  "0" "ActionForMenu1" "UpdateForMenu1" "Shift+X" "" "" "This
is my menu test2"] ""]
            InsertXMLMenu  [list [list "MyFile1" "MyFileLevelpopup1" "MySeparator1"] "1"
"MyFileLevelAction1" [list "separator"] ""]
            InsertXMLMenu  [list [list "MyFile1" "MyFileLevelpopup1" "MyFileLevelAction8"]
"" "" [list "action" "&menu6"  "0" "ActionForMenu1" "UpdateForMenu1" "Shift+Q" "" "" "This
is my menu test2"] ""]
            InsertXMLMenu  [list [list "MyFile1" "MyFileLevelpopup1" "MyFileLevelAction9"]
"" "" [list "action" "&menu7"  "0" "ActionForMenu1" "UpdateForMenu1" "Ctrl+F" "" "" "This
is my menu test2"] ""]
            InsertXMLMenu  [list [list "MyFile1" "MyFileLevelpopup1" "MyFileLevelAction11"]
"" "" [list "action" "&menu9"  "0" "ActionForMenu1" "UpdateForMenu1" "Alt+F" "" "" "This is
my menu test2"] ""]

            RegisterAction "ActionForMenu1"   "::test::shouldProcess" ""
"::test::testActionProc1" ""
            RegisterAction "UpdateForMenu1" "::test::shouldProcess" ""
"::test::testUpdateProc1" ""
            RegisterAction "ActionForMenu2"   "::test::shouldProcess" ""
"::test::testActionProc2" ""
            RegisterAction "UpdateForMenu2" "::test::shouldProcess" ""
"::test::testUpdateProc2" ""
        }
    }
    proc shouldProcess { args } {
        return 1
    }
    proc shouldProcess1 { args } {
        return 0
    }

    proc testMePM { args } {
        puts "CALLED FROM PM WINDOW"
    }

    proc testMeSCM { args } {
        puts "CALLED FROM SCHEMATICS"
    }

    proc testActionProc1 { args } {
       puts "CALLED FROM MAIN MENU"
    }

    proc testUpdateProc1 { args } {
       return true
    }

}
::test::registerMenuActions
```

2. Launch Capture
3. Choose *View - Command* Window to view the TCL scripts loaded at startup in the command window.
4. Open the design, in this example *Full Adder*, in Capture.
   The *Full Adder* design is located in the following folder:
   `<installation>\tools\capture\samples`

Setting Basic Application Structure with Menu

Loading package code in memory only when it is required ensures stability and lowers memory usage. The following sample TCL script can be used as a template to set up application packages.

The sample TCL source script provides a template on how to integrate application code with the menu. The script:

- adds a menu to Capture
- sets up the application package

Do the following steps to set up basic TCL application structure with menu:

1. Add the following `Sample-17.tcl` file to
   `<installation>\tools\capture\tclscripts\capAutoLoad`

Sample-17.tcl

```
namespace eval ::CDNSCustomAppInstall {
    set packageLoaded 0
    InsertXMLMenu  [list [list "Place" "PSpiceComponent" "CDNS"] "" "" [list "popup" "CDNS
Browser"  "" "" "" "" ""] ""]
    InsertXMLMenu  [list [list "Place" "PSpiceComponent"  "AlignSep2"] "0" "CDNS" [list
"separator"] ""]
    InsertXMLMenu  [list [list "Place" "PSpiceComponent"  "CDNS"  "Search"] "" "" [list
"action" "Search"  "0" "OnCDNSSearch" "OnUpdateCDNSSearch" "" "" "" "Search"] ""]
    RegisterAction "OnCDNSSearch"   "::CDNSCustomAppInstall::enable" ""
"::CDNSCustomAppInstall::OnSearch" ""
    RegisterAction "OnUpdateCDNSSearch" "::CDNSCustomAppInstall::enable" ""
"::CDNSCustomAppInstall::enableUpdate" ""
    proc enable { args } {
        return 1
    }
    proc enableUpdate { args } {
        if { [capIsPSpiceProject] && [IsSchematicViewActive] } {
            return true
        }
        return false
    }
    proc OnSearch {} {
        if {$::CDNSCustomAppInstall::packageLoaded == 0} {
            package require CDNSCustomApp
            set $::CDNSCustomAppInstall::packageLoaded 1
            puts "Package Loaded"
        }
        CDNSCustomApp::OnSearch
    }
}
```

2. Add the following TCL source code files, `CDNSCustomApp.tcl` and `pkgIndex.tcl`, to the `Sample-17` folder

CDNSCustomApp.tcl

```
package provide CDNSCustomApp 1.0
namespace eval ::CDNSCustomApp {
    proc OnSearch {} {
        puts "Method OnSearch Called"
    }
}
```

pkgindex.tcl

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

3. Copy the *`Sample-17`* folder to `<installation>\tools\capture\tclscripts`.
4. Restart Capture.
5. Open the *Full Adder* design file.
6. Verify in session log that the package is loaded only on menu click.
7. Add the following code in the `Sample-17.tcl` source file:

```
    proc enableUpdate { args } {
        if { [capIsPSpiceProject] && [IsSchematicViewActive] } {
            return true
        }
        return false
    }
```

8. Restart Capture.
9. Open the *Full Adder* design file.
10. Verify that the menu is disabled as the Full Adder design is not a PSpice project.
11. Open any project from `<installation>\tools\pspice\capture_samples` folder.
12. Verify that the menu is enabled.

⚠ It is important to load the application package only when a user clicks on the menu. Use the package provided and the `pkgIndex.tcl` file to load on "package require".

Accessing Session, Design, and Page Objects in Capture

The following sample TCL source file:

- gets the session objects from Capture and lists open designs
- finds full path of the current active project
- gets the active design

- queries the page in focus
- uses the `DboState` function to write to the session log
- uses the `capMessageBox` function to display a message

Do the following steps to access session, design, and page objects in Capture:

1. Start Capture.
2. Open the *Full Adder* design.
3. Open another design from `<installation>\tools\pspice\capture_samples`.
4. Select the Project Manager window of the Full Adder design.
5. Open the Full Adder page of the Full Adder schematic.
6. Make the Full Adder page active.
7. Using the `source` command, source the `Sample-18.tcl` file in the command window.
   ```
   source {<Absolute path of the Sample 18.tcl file>}
   ```

Sample-18.tcl

```
interp alias {} ? {} puts $errorInfo
set lSession $::DboSession_s_pDboSession
DboSession -this $lSession
set lStatus [DboState]
set lDesignsIter [$lSession NewDesignsIter $lStatus]
set lDesign [$lDesignsIter NextDesign $lStatus]
set lCStr [DboTclHelper_sMakeCString]
while { $lDesign!= "NULL"} {
    DboDesign_GetRootName $lDesign $lCStr
    DboState_WriteToSessionLog $lCStr

    set lDesign [$lDesignsIter NextDesign $lStatus]
}
delete_DboSessionDesignsIter $lDesignsIter
catch {set lProjectPath [GetActiveOpjName]}
DboTclHelper_sSetCString  $lCStr "Current Project file $lProjectPath"
DboState_WriteToSessionLog  $lCStr
catch {set lCurrentDesign [GetActivePMDesign]}
DboTclHelper_sSetCString  $lCStr "Current Design $lCurrentDesign"
DboState_WriteToSessionLog  $lCStr
catch {set lCurrentPage [GetActivePage]}
DboTclHelper_sSetCString  $lCStr "Current Page $lCurrentPage"
DboState_WriteToSessionLog  $lCStr
catch {set isSchViewActive [IsSchematicViewActive]}
DboTclHelper_sSetCString  $lCStr "Is Schematic View active: $isSchViewActive"
DboState_WriteToSessionLog  $lCStr
capDisplayMessageBox "Execution Completed. \n See output of this in Session Log" "Info"
```

Using Record and Replay Commands

Capture provides capabilities to record and replay commands. When journaling is enabled all commands are saved in a file.

The following example:

- enables command echo and journaling
- saves commands for future replay
- replays commands from Capture command line

Do the following steps to record and replay commands:

1. Open the `Sample-19\Sample-19.dsn` design.
2. Open the design file from the project window.

3. Enable Journaling
    a. Choose *Preferences - More Preferences*
    b. Select Command Shell
    c. Enable all options under journaling
4. Restart Capture.
5. Open the `Sample-19\Sample-19.dsn` design.
6. Create two hierarchical blocks as shown and connect the pins
7. Save and Exit Capture
8. Locate Capture Log
    a. Type `%TEMP%` in windows start
    b. Locate CAPTURELOG directory
    c. Copy the latest directory to `Sample-19\Sample-19\stage1`
9. Restore the Capture design to original state - copy `Sample-19\backup\Sample-19.dsn` to `Sample-19`.
10. Copy the contents of the `OrCAptureLogFile.captl` file to `Sample-19.tcl`. Add the following commands at the end of the file:

```
SelectPMltem "./Sample-19.dsn"
Menu "File::Close"
Menu "File::Exit"
```

11. Open Windows Command Shell in the database directory.
12. Execute the following command: `capture -product="OrCAD Capture" Sample-19.tcl`
13. Capture will come up and execute the TCL file that will recreate the blocks and wire. At the end Capture.exe will automatically close the design and exit.

Sample-19.tcl

```
Open d:/Sample_Scripts/Sample-19/Sample-19.opj
SelectPMItem "./Sample-19.dsn"
OPage "SCHEMATIC1" "PAGE1"
SelectPMItem "SCHEMATIC1/PAGE1"
OPage "SCHEMATIC1" "PAGE1"
PlaceBlock 1.89 1.63 3.61 2.99 "" "" "H1" "DEFAULT"
SetProperty {Implementation Type} {<none>}
PlacePin 3.60 2.30 "A" "Passive" FALSE
EndPlace
UnSelectAll
PlaceBlock 4.99 1.63 6.51 2.99 "" "" "H2" "DEFAULT"
SetProperty {Implementation Type} {<none>}
PlacePin 5.00 2.30 "B" "Passive" FALSE
EndPlace
UnSelectAll
MenuCommand "13992"
PlaceWire 3.60 2.30 5.00 2.30
EndPlace
UnSelectAll
Menu "File::Save"
```

This provides an easy way to record and execute OrCAD Capture commands.

It is recommended to try out each of the commands manually by pasting them on the command window.

Accessing OrCAD Database Objects from GUI

Capture provides an API to access database objects from GUI using `GetSelectedObjects`. This API provides a list of objects that are currently selected on canvas.

The following example:

- accesses the list of selected objects
- prints out the list of Selected objects and their types

Do the following steps to access OrCAD Database Objects from GUI:

1. Launch Capture.
2. Open the Full Adder design.
3. Open the Half Adder schematic page.
4. Select any set of the objects on Half Adder page.

5. In the TCL command window, source the `Sample-20.tcl` file using the `source` command.
   `source {<Absolute path of the Sample 20.tcl file>}`

Sample-20.tcl

```
set lCStr [DboTclHelper_sMakeCString]
set lSelectedObjectsList [GetSelectedObjects]
puts "Total number of Selected Objects: [llength $lSelectedObjectsList]"
foreach lObject $lSelectedObjectsList {
    $lObject GetTypeString $lCStr
    set lObjTypeStr [DboTclHelper_sGetConstCharPtr $lCStr]
    $lObject GetName $lCStr
    set lObjName [DboTclHelper_sGetConstCharPtr $lCStr]
    puts "Object:$lObject  Type:$lObjTypeStr  $lObjName"
}
```

This lists the selected objects and their total number.

Reading and Updating Part Reference from Occurrence of Selected instances

The `GetSelectedObjects` command lists the instances selected in a page.

The following example will:

- filter Parts and Hierarchical Blocks from selected objects
- read Reference property from Inst
- jump to the hierarchical occurrence of the active schematic
- get occurrence of the instance under schematic occurrence container
- read Reference property from Occurrences

Do the following steps to read and update part reference from selected instances occurrences:

1. Launch Capture.
2. Open the Full Adder design.
3. Open the Half Adder schematic page.
4. Select any number of objects on the Half Adder page.
5. Source the `Sample-21.tcl` file in the command window.

Sample-21.tcl

```
package provide myFirstAppPackage 1.0
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
        }
    }
```

Making a Property Visible

A new Display Property object is needed to make a property visible.

The following example that shows the use of the Font and Position API, performs the following tasks :

- makes a property visible in the database
- uses GUI interface to properly position the property with default Capture algorithm
- refreshes the page automatically to display changes

Do the following steps to make a property visible:

1. Launch Capture.
2. Open the *Full Adder* design
3. Open page *HALFADD*

4. Select any set of objects on HALFADD
5. In Tcl command window source `Sample-22.tcl`

Sample-22.tcl

```
set lStatus [DboState]
set lCStr [DboTclHelper_sMakeCString]
set lPropName [DboTclHelper_sMakeCString "Source Package"]
set pLocation [DboTclHelper_sMakeCPoint 0 0]
set pFont [DboTclHelper_sMakeLOGFONT "Arial" 8 0 0 0 400 0 0 0 0 7 0 1 16]
set lPage [GetActivePage]
set lSelectedObjectsList [GetSelectedObjects]
puts "Total number of Selected Objects: [llength $lSelectedObjectsList]"
foreach lObj $lSelectedObjectsList {
    $lObj GetTypeString $lCStr
    set lObjTypeStr [DboTclHelper_sGetConstCharPtr $lCStr]
    if { $lObjTypeStr == "Placed Instance" } {
        set lProp [$lObj NewDisplayProp $lStatus $lPropName $pLocation 0 $pFont 48]
        DboPartInst_PositionDisplayProp $lPage $lProp
    }
}
UnSelectAll
catch {Menu View::Zoom::Redraw}
```

Making a Property Invisible

Display Property object needs to be deleted to make a property invisible.

The following example will:

- delete a property visibility from Database
- refresh the page automatically to display changes

Do the following steps to make a property invisible:

1. Launch Capture.
2. Open the Full Adder design.
3. Open page HALFADD
4. Select any set of objects on HALFADD
5. In the command window, source the `Sample-22.tcl` file
6. Source Package property becomes visible

7. Select any set of objects on HALFADD
8. In the command window source the `Sample-23.tcl` file

Sample-24.tcl

```
set lStatus [DboState]
set lCStr [DboTclHelper_sMakeCString]
set lPropName [DboTclHelper_sMakeCString "Source Package"]
set pLocation [DboTclHelper_sMakeCPoint 0 0]
set pFont [DboTclHelper_sMakeLOGFONT "Arial" 8 0 0 0 400 0 0 0 0 7 0 1 16]
set lPage [GetActivePage]
set lSelectedObjectsList [GetSelectedObjects]
puts "Total number of Selected Objects: [llength $lSelectedObjectsList]"
foreach lObj $lSelectedObjectsList {
    $lObj GetTypeString $lCStr
    set lObjTypeStr [DboTclHelper_sGetConstCharPtr $lCStr]
    if { $lObjTypeStr == "Placed Instance" } {
        set lDispProp [$lObj GetDisplayProp $lPropName $lStatus]
        $lObj DeleteDisplayProp $lDispProp
    }
}
UnSelectAll
catch {Menu View::Zoom::Redraw}
```

Modifying Properties

The following example will:

- change the value of an existing property
- call Page Connectivity update as some property changes may change connectivity

Do the following steps do modify existing properties in an OrCAD Capture project:

1. Launch Capture.
2. Open the Full Adder design.
3. Open the HALFADD page.
4. Select any set of objects on HALFADD.
5. In the command window, source the `Sample-25.tcl` file.

⚠ Using Effective property – the reference property is reset. As we are working with Instance object – the property of instance object is changed

Sample-25.tcl

```
set lStatus [DboState]
set lCStr [DboTclHelper_sMakeCString]
set lPropValue [DboTclHelper_sMakeCString]
set lPropName [DboTclHelper_sMakeCString "Reference"]
set lSelectedObjectsList [GetSelectedObjects]
puts "Total number of Selected Objects: [llength $lSelectedObjectsList]"
foreach lObj $lSelectedObjectsList {
    $lObj GetTypeString $lCStr
    set lObjTypeStr [DboTclHelper_sGetConstCharPtr $lCStr]
    if { $lObjTypeStr == "Placed Instance" } {
        $lObj GetEffectivePropStringValue $lPropName $PropValue
        set lCurrentValue [DboTclHelper_sGetConstCharPtr $lPropValue]
        DboTclHelper_sSetCString $lPropValue "?"
        $lObj SetEffectivePropStringValue $lPropName $lPropValue
    }
}
UnSelectAll
DboTclHelper_sEvalPage [GetActivePage]
catch {Menu View::Zoom::Redraw}
```

Deleting Properties

The following example will:

- change the value of an existing property
- call Page Connectivity update as some property changes may change connectivity

Do the following steps to delete existing property in a Capture project:

1. Launch Capture.
2. Open the Full Adder design.
3. Open the `HALFADD` page with the `halfadd_B` occurrence.
4. Select any set of objects on `HALFADD`
5. In the Capture command window source the `Sample26.tcl` file.
   Once the TCL file is sourced, the occurrence property is deleted.

Sample-26.tcl

```
set lStatus [DboState]
set lCStr [DboTclHelper_sMakeCString]
set lSelectedObjectsList [GetSelectedObjects]
puts "Total number of Selected Objects: [llength $lSelectedObjectsList]"
foreach lObj $lSelectedObjectsList {
    $lObj GetTypeString $lCStr
    set lObjTypeStr [DboTclHelper_sGetConstCharPtr $lCStr]
    set InstOcc [GetInstanceOccurrence]
    if { "NULL" != $InstOcc} {
        if { $lObjTypeStr == "Placed Instance" } {
            $lObj GetName $lCStr
            set lObjName [DboTclHelper_sGetConstCharPtr $lCStr]
            set lPartOcc [$lObj GetObjectOccurrence $InstOcc]
            set lInstOcc [DboOccurrenceToDboInstOccurrence $lPartOcc]
            $lInstOcc DeleteEffectiveProp $lPropName
        }
    }
}
UnSelectAll
DboTclHelper_sEvalPage [GetActivePage]
catch {Menu View::Zoom::Redraw}
```

Placing a Part in a Capture Design

There are many ways to place parts in Capture canvas. A very complicated way is to add it only using database commands. Ideally, this method may be avoided and instead GUI commands may be used with Capture in Batch mode if an action has to be done in automation.

In GUI commands, there are again two methods –

- Interactive Method – an example is to place PSpice Ground on menu click
- Non-Interactive Method – to provide the location where part needs to be placed through code

The following example will:

- place a part in the interactive mode
- place a part in the non-interactive Mode

Do the following steps to place a part in a Capture design:

1. Launch Capture
2. Open the Full Adder design
3. Open page HALFADD with occurrence `halfadd_B`

4. Enter the following commands in command window:
   ```
   PlacePart 1.50 1.20 "[exec cds_root
   cds_root]\\tools\\capture\\library\\Amplifier.olb" MAX469 "" FALSE
   PlacePart 7.00 1.00 "[exec cds_root
   cds_root]\\tools\\capture\\library\\Amplifier.olb" AD8072 "B" FALSE
   ```
5. Enter the following command to place component interactively with mouse:
   ```
   ::PlacePartEx "[exec cds_root
   cds_root]\\tools\\capture\\library\\pspice\\breakout.olb" "Dbreak" "" 0
   ```

Importing and Exporting of XML and TCL Code for a Capture Library

Capture supports export and import of XML for libraries. While importing XML it is also possible to generate the TCL code that generates the same library.

The following example will:

- export XML from Capture library (`.olb`)
- import XML to create new Capture library (`.olb`)
- view TCL on the methods to create Capture library (`.olb`)

Do the following steps to export and import XML, and view TCL:

1. Open Windows Command line.
2. Set environment variable with the following command:
   ```
   set CAPTURE_WRITE_TCL_ONXMLIMPORT=1
   ```
   ⚠ You can also set the environment variable in the Windows environment user interface.
3. Create a new folder `Sample-27`
4. Launch Capture.
5. Execute the following commands in the command window.
   ```
   cd {<The absolute path of the Sample-27 folder>}
   set pLibPath [file normalize "[exec cds_root
   cds_root]/tools/capture/library/amplifier.olb"]
   ```

```
XMATIC_OLB2XML $pLibPath amplifier.xml
XMATIC_XML2OLB amplifier.xml temp.olb
exec notepad l.tcl
```

You will notice that the Sample-27 folder has the following three new files:

- `Amplifier.xml` - XML exported from the amplifier.olb file
- `Temp.olb` - a library created from the XML
- `l.tcl` - TCL commands used to create Temp.olb

⚠ The TCL file created is large because of the large library used here. To understand the TCL API, it is recommended to create a library with just one part library for which you would like to see the TCL file and then execute the statements again.

Importing and Exporting XML and TCL Code for a Design

Capture supports export and import of XML for design files. While importing XML it is also possible to generate Tcl code that generates the same design.

The following example will:

- export XML from Capture design (.dsn)
- import XML to create new Capture design (.dsn)
- view TCL code on the methods to create Capture designs (.dsn)

In this example, the goal is to learn the TCL API to create a hierarchical block with a pin. For this reason, we have taken a design with just one hierarchical block and a one pin on it.

Do the following steps to import and export XML and TCL file for a design:

1. Open Windows Command prompt.
2. Set environment variable with the following command:
   ```
   set CAPTURE_WRITE_TCL_ONXMLIMPORT=1
   ```
3. Launch Capture.
4. Execute the following commands in Capture command window:
   ```
   cd {<absolute path of the Sample-28 design file>}
   ```

```
XMATIC_DSN2XML Sample-28.dsn fulladd.xml
```

```
XMATIC_XML2DSN fulladd.xml temp.dsn
```

```
exec notepad l.tcl
```

5. In the Sample-28 folder, the following three new files are created:
    a. `Sample-28.xml` - XML exported from Sample-28.dsn
    b. `Temp.dsn` - design created from the XML file
    c. `l.tcl` - TCL commands used to create Temp.dsn

On examining generated Tcl file, we can locate the code needed for creating a hierarchical block with one pin. Using this method, we can get the exact TCL code for creating various types of database scenarios.

Using Callbacks

OrCAD Capture provides callbacks that allow you to embed code for different actions. For example, a function can be added to be called after the design is saved. The function might then check-in the file into a repository automatically after every save.

The following example will:

- add user-defined pre-save function to be called before any library is saved
- add user-defined post-save function to be called after any library is saved
- add user-defined pre-save function to be called before any design is saved
- add user-defined post-save function to be called after any design is saved

Do the following steps to get the callbacks function names:

1. Launch Capture.
2. Source `Sample-29.tcl` into Capture's command window.

Sample-29.tcl

```
proc capSaveTriggerTrue {args} {
    return 1
}
proc capPreLibSave { pLibrary } {
       puts "capPreLibSave on $pLibrary called"
}
proc capPostLibSave { pLibrary } {
       puts "capPostLibSave on $pLibrary called"
}
proc capPreDesignSave { pDesign } {
       puts "capPreDesignSave on $pDesign called"
}
proc capPostDesignSave { pDesign } {
       puts "capPostDesignSave on $pDesign called"
}
RegisterAction "_cdnOrOnLibraryPreSave" "capSaveTriggerTrue" "" "capPreLibSave" ""
RegisterAction "_cdnOrOnLibraryPostSave" "capSaveTriggerTrue" "" "capPostLibSave" ""
RegisterAction "_cdnOrOnDesignPreSave" "capSaveTriggerTrue" "" "capPreDesignSave" ""
RegisterAction "_cdnOrOnDesignPostSave" "capSaveTriggerTrue" "" "capPostDesignSave" ""
```

3. Open any design or library.
4. On saving note the messages printed from user defined called functions.