cadence®

Design Data Translators SKILL Reference

Product Version IC23.1
November 2023

Contents

1

XStream Functions

The SKILL programming language lets you customize and extend your design environment. SKILL provides a safe, high-level programming environment that automatically handles many traditional system programming operations, such as memory management. SKILL programs can be immediately executed in the Cadence environment.

This information set describes custom layout SKILL functions for layout editor, parameterized cells, compactor, structure compiler, placement and routing translation, layout XL, custom digital placer, and Virtuoso constraint manager. It is aimed at developers and designers of integrated circuits and assumes that you are familiar with:

- The SKILL programming language.
- The Virtuoso design environment and application infrastructure mechanisms designed to support consistent operations between all Cadence tools.
- The applications used to design and develop integrated circuits in the Virtuoso design environment, notably Virtuoso Layout Suite and Virtuoso Schematic Editor.
- The OpenAccess version 2.2 technology file.

Component description format (CDF), which lets you create and describe your own components for use with Layout XL.

Licensing Requirements

This section describes an overview of how licensing is implemented in physical translators, XStream In, XStream Out, XOasis In, XOasis Out, LEF In, LEF Out, DEF In, and DEF Out translators. The licensing scheme is the same for both command line and GUI translators.

In case, a license is not available for a translator and you select *File – Import – LEF/DEF/Stream* option from CIW, a form will be displayed without a license. You can specify the values in the form and click the OK button. The translator in this case would perform a validation check for license and try to checkout the license. You will not be able to do the translation if the license is unavailable.

This license "`111`" is required to run all these physical design translators.

These translators work only with the Design Framework II license "`111`" and with no other product license. Therefore, even if Layout XL or a higher tier license is available, you will not be able to run these translators if license "`111`" is not available. Also, to be able to run some advanced SKILL APIs that used in `libInit.il` or Pcell SKILL code, you will require functionality based licenses (L/XL/GXL).

This chapter provides syntax, descriptions, and examples for the SKILL functions associated with the XStream translator.

There are two types of SKILL functions in XStream:

- Command-Line SKILL Functions
- GUI SKILL Functions
- Callback Functions

Command-Line SKILL Functions

poCellNameMap

```
poCellNameMap(
     t_lib
     t_cell
     t_view
     )
     => t_mapName / nil
```

Description

Passes the library name, cell name and the view name for each cell in the library to a user-defined procedure.

While translating a cellview, Stream Out first checks for the cell name map file. If a cell name map file is not specified, then Stream Out checks whether the user-defined SKILL file is provided and the `poCellNameMap` function is defined. If the function is defined, it is called and the library name, cell name, and view name are passed to this function. The string value returned by this function is considered as new name for the cellview. For more information, see the Cell Name Map File section.

When both *Cell Map File Name* and `poCellNameMap` are specified, value specified for *Cell Map File Name* is used.

Arguments

t_lib	OpenAccess library name
t_cell	OpenAccess cell name
t_view	OpenAccess view name

Value Returned

t_mapName	The translated cell name
`nil`	If the cell name is not mapped

Example

In this example, the characters `"_po"` are added to every cell name. The library and view are ignored.

```
procedure( poCellNameMap( lib cell view )
   prog( ( mapname )
        sprintf( mapName "%s_po" cell)
                return( mapName )
   );prog
);procedure
```

poLayerMap

```
poLayerMap(
     t_layerName
     t_purposeName
     )
     => l_layerDatatype / nil
```

Description

This is a user-defined function that passes the layer number and datatype for each layer in the input Stream file, when it is called. Stream Out interprets the output of the procedure in the same way it interprets the OpenAccess name and purpose. However, Stream Out checks for this function only if the layer map file (`-layerMap` option) is not specified. If the layer map file is specified, the `poLayerMap` function is ignored. For more information, see the Layer Map File section.

Arguments

t_layerName	OpenAccess layer name.
t_purposeName	OpenAccess purpose name.

Value Returned

l_layerDataType	A list containing two integers. The first integer is the translated layer number. The second integer is the translated data type.
`nil`	If no mapping is provided.

Example

In this example, the `"text drawing"` layer-purpose pair is mapped to GDSII layer 15 and data type 0. The `"diff drawing"` layer-purpose pair is mapped to layer 18 and data type 0.

```
procedure( poLayerMap( layerName layerPurpose )
          prog( ( lay )
              case( layerName
                  ( "text"        return( list( 15 0 ) ) )
                  ( "diff"        return( list( 18 0 ) ) )
                  ( "substrate"   return( list( 20 0 ) ) )
                  ( t             return( list( 55 0 ) ) );default
              );case
          );prog
);procedure
```

textFontMap

```
textFontMap(
     t_fontName
     )
     => l_streamFont / nil
```

Description

Passes each font type in a Design Framework II library to a user-defined procedure.

While translating the OpenAccess label, Stream Out checks whether the font is mapped through text font map file. If the text font map file is not provided or the font is not mapped through the text font map file, then Stream Out checks for the `textFontMap` SKILL function. If the function is defined, then Stream Out calls this function and OpenAccess font name as the argument.

Valid Stream font numbers are 0, 1, 2, and 3. The OpenAccess font names are `Unknown`, `EuroStyle`, `Gothic`, `Math`, `Roman`, `Script`, `Stick`, `Fixed`, `Swedish`, and `MilSpec`. For more information, see the Text Font Map File section.

Arguments

t_fontName	The OpenAccess font name.

Value Returned

l_streamFont	Stream font number.
`nil`	Font not mapped.

Example

In this example, the DFII fonts `Gothic`, `Roman`, and `Script` are mapped to Stream fonts `3`, `1`, and `2`. All others are mapped to the Stream font `0`.

```
procedure( textFontMap( font )
          prog( ()
               case( font
                    ( "Gothic"      return( 3 ) )
                    ( "Roman"       return( 1 ) )
                    ( "Script"      return( 2 ) )
                    (t              return( 0 ) )
               );case
```

```
          );prog
);procedure
```

piCellNameMap

```
piCellNameMap(
     t_cell
     )
     => l_cellview / nil
```

Description

Passes the cell name for each cell name in the input file to a user-defined procedure.

Stream In interprets the output of the procedure in the same way as it interprets the output of the same procedure in the OpenAccess library, cell, and view names. For more information, see the Cell Name Map File section.

Arguments

t_cell	The input cell name.

Value Returned

l_cellview	A list containing the OpenAccess library, cell, and view names.
nil	No cell and view names are found.

Example

In this example, the characters `"_pi"` are added to every cell name and `reflib1` and `layout` is returned as library name and view name.

```
procedure( piCellNameMap( cell )
         prog( ( mapname )
              sprintf( mapname "%s_pi" cell )
              return( list( “reflib1” mapname "layout" ) )
         );prog
);procedure


procedure( piCellNameMap( cell )
         prog( ( mapname )
              sprintf( mapname "%s_pi" cell )
              return( list( “XST_TARGET_LIB” mapname "layout" ) )
         );prog
);procedure
```

Note: Here, the `XST_TARGET_LIB` name will be replaced with the library name specified using the `-lib` option.

In this example, the special character / in a cell name is replaced with _.

```
procedure( piCellNameMap( cell )
                     prog( ( mapname )
                               cellName=sprintf( nil "%s" cell )
                                rexCompile("/")
                                mapname=rexReplace(cellName "_" 0)
                                println(mapname)
                return( list( "myLib" mapname "layout" ) )
         );prog
);procedure
```

piLayerMap

```
piLayerMap(
     x_layer
     x_dataType
     )
     => l_lpp / nil
```

Description

This is a user-defined function that passes the layer number and datatype for each layer in the input Stream file, when it is called. Stream In interprets the output of the procedure in the same way it interprets the OpenAccess name and purpose. However, Stream In checks for this function only if the layer map file (`-layerMap` option) is not specified. If the layer map file is specified, the `piLayerMap` function is ignored. For more information, see the Layer Map File section.

Arguments

x_layer	Input layer number.
x_dataType	Input datatype number.

Value Returned

l_lpp	A list containing two strings. The first string is the dfII layer name. The second string is the dfII layer purpose.
`nil`	No layer or datatype numbers are found.

Example

In this example, the input layers 1 to 10 are mapped to the Design Framework II layer-purpose pair `text drawing`.

```
procedure( piLayerMap( layer datatype )
    prog( ( lay )
    lay=layer
    if( lay >= 1 && lay <= 10 then
    return( list( "text" "drawing") )
)))
```

piTextMap

```
piTextMap(
     t_label
     )
     => t_changedLabel
```

Description

Modifies the text that is translated from the Stream file to a dfII library.

During Stream In, if you specify a SKILL file having `piTextMap` defined in it, then this function is called for each text object and the text string is passed as an argument to this function. The string returned by this function is used to modify the text in the target dfII library.

Arguments

t_label	Text to be modified.

Value Returned

t_changedLabel	Modified text.

Example

During Stream In, you can replace the character `[` with < and character `]` with > by using the `piTextmap` function.

```
procedure( piTextMap( label )
         prog( ( newLabel )
              rexCompile( "\\[" )
              newLabel = rexReplace( label "<" 1 )
              rexCompile( "\\]" )
              newLabel = rexReplace( newLabel ">" 1 )
              return( newLabel )
         );prog
);procedure
```

poTextMap

```
poTextMap(
     t_label
     )
     => t_changedLabel
```

Description

Modifies the text that is translated from the dfII library to the Stream file.

During Stream Out, if you specify a SKILLfile having `poTextMap` defined in it, then this function is called for all text objects and the text string is passed as an argument to this function.

The string returned by this function is used to create the text object in the Stream file. It is applicable to all the strings present in the dfII library and being translated as text in the Stream file as labels. The string is also applicable to the text translated corresponding to pins when you use the *Convert Pin to* option to translate pins as texts.

Arguments

t_label	Text to be modified.

Value Returned

t_changedLabel	Modified text.

Example

During Stream In, you can replace the character `[` with < and character `]` with > by using the `poTextmap` function.

poParamCellNameMap

```
poParamCellNameMap(
     t_name
     d_cvid
     )
     => t_mapName
```

Description

This is a user-defined function that is called with the parameterized cell name (super-master name) and sub-master cellview identifier. Stream Out interprets the output of the procedure in the same way as it interprets the translated name for the parameterized cell.

Arguments

t_name	The Virtuoso Design Environment parameterized cell name (super-master name) defined by the user.
d_cvid	The Virtuoso Design Environment parameterized cell variant (sub-master) cellview identifier.

Value Returned

t_mapName	The name of the translated parameterized cell.

Example

In this example, the translator appends the cellview identifier to every parameterized cell name, making the cell name unique.

```
procedure( poParamCellNameMap( name ID )
  sprintf( nil "%s_%L" name ID )
);procedure
```

poPreTranslate

```
poPreTranslate(
     lib
     cell
     view
     )
```

Description

During StreamOut, this function is called just before the translation starts.

Caution

Open the OpenAccess designs only in read mode.

Note: XStream Out supports only those SKILL functions that are essential to evaluate a Pcel. These are SKILL Core, `db`, `dd`, `tech`, `cst`, `abe`, `vfo` (vfo.cxt), and `pas` (pdkutils.cxt) SKILL functions. Application specific functions such as `le`, `hi`, `sch`, `lx`, and `via` are not supported.

XStream In supports only following SKILL Core, `db`, `dd`, `tech`, and `cst` SKILL functions. Application specific functions, such as `le`, `hi`, `sch`, `lx`, `via` and so on are not supported.

Arguments

lib	Contains the input library specified by the user.
cell	Contains the topCell specified by the user or empty string if not specified.
view	Contains the view specified by the user or empty string if not specified.

Value Returned

None

Example

In this example, the function will print the values of `lib`, `cell`, and `view`.

```
procedure(poPreTranslate(lib cell view)
let((cv)
```

```
    printf("In PreTranslate of Stream Out\n")
    printf("Library: %s\n" lib)
    printf("Cell: %s\n" cell)
    printf("View: %s\n" view)
    cv=dbOpenCellViewByType(lib cell view "maskLayout" "r")
    ;; modification in cv
);let
)
```

poPostTranslate

```
poPostTranslate(
     lib
     cell
     view
     )
```

Description

During StreamOut, this function is called just after the translation is completed.

Arguments

lib	Contains all libraries created by StreamIn, separated by space.
cell	Contains the topCell specified by the user or empty string if not specified.
view	Contains the view specified by the user or empty string if not specified.

Value Returned

None

Example

In this example, the function will print the values of `lib`, `cell`, and `view`.

```
procedure( poPostTranslate( lib cell view )
    prog( ( )
    printf("In PostTranslate of Stream Out\n")
    printf("Library: %s\n" lib)
    printf("Cell: %s\n" cell)
    printf("View: %s\n" view)
    );prog
)
```

piPreTranslate

```
piPreTranslate(
     lib
     cell
     view
     )
```

Description

During XStream In translation, this function is called just before the translation starts.

Arguments

lib	Contains the destination library specified by the user.
cell	Contains the topCell specified by the user or empty string if not specified.
view	Contains the view specified by the user or empty string if not specified.

Value Returned

None

Example

In this example, the function will print the values of `lib`, `cell`, and `view`.

```
procedure( piPreTranslate( lib cell view )
     prog( ( )
     printf("In PreTranslate of Stream In\n")
     printf("Library: %s\n" lib)
     printf("Cell: %s\n" cell)
     printf("View: %s\n" view)
     );prog

)
```

piPostTranslate

```
piPostTranslate(
     lib
     cell
     view
     )
```

Description

During XStream In translation, this function is called just after the translation is completed. If the `-topCell` option is not specified, `piPostTranslate` is called with all the top cells found during Xstream In translation as a string of space separated names.

Caution

Open the OpenAccess designs only in append mode.

Note: XStream Out supports only those SKILL functions that are essential to evaluate a Pcel. These are SKILL Core, `db`, `dd`, `tech`, `cst`, `abe`, `vfo` (vfo.cxt), and `pas` (pdkutils.cxt) SKILL functions. Application specific functions such as `le`, `hi`, `sch`, `lx`, and `via` are not supported.

XStream In supports only following SKILL Core, `db`, `dd`, `tech`, and `cst` SKILL functions. Application specific functions, such as `le`, `hi`, `sch`, `lx`, `via` and so on are not supported.

Arguments

lib	Contains all the libraries created by StreamIn, separated by space.
cell	Contains the top cell specified by the user or a string of top cells identified by the Xstream In translator, if not specified.
view	Contains the view specified by the user or empty string if not specified.

Value Returned

None

Examples

Example 1

In this example, the function will print the values of `lib`, `cell`, and `view`.

```
procedure(piPostTranslate(lib cell view)
let((cv)
    printf("In PostTranslate of Stream In\n")
    printf("Library: %s\n" lib)
    printf("Cell: %s\n" cell)
    printf("View: %s\n" view)
    cv=dbOpenCellViewByType(lib cell view "maskLayout" "a")
    ;; modification in cv
);let
)
```

Example 2

```
procedure(piPostTranslate(lib cell view)
        let((c sumFile logFile)
            if(listp(cell) == nil then
                printf("\n piPostTranslate is called with topcell = (%s)\n" cell)
             else
                printf("\n piPostTranslate is called with topcell list = (")
                foreach( c cell
                    printf(" %s" c)
                    )
                printf(")\n")
              )))
```

Output:

```
piPostTranslate is called with topcell = (topCell1 topCell2 topCell3)
```

GUI SKILL Functions

xstInOnCancel

```
xstInOnCancel()
```

Description

This is a user-defined function, which is called when user presses the *Cancel* button.

Arguments

None

Value Returned

None

Example

```
procedure( xstInOnCancel()
             sprintf("Translation canceled")
);procedure
```

xstInOnTranslate

```
xstInOnTranslate()
```

Description

This is a user-defined function, which is called when user presses the *Apply* or *Translate* button.

Arguments

None

Value Returned

None

Example

```
procedure( xstInOnTranslate()
             sprintf("Starting Translation")
);procedure
```

xstInOnCompletion

`xstInOnCompletion(x_num)`

Description

This is a user-defined function, which is called when the translation is completed.

Arguments

x_num	It is an integer parameter. If the translation is completed without any error, this value is zero. However, if error occurs during translation, this value is non-zero.

Value Returned

None

Example

```
procedure( xstInOnCompletion( status )

            fprintf(outPort "In xstInOnCompletion of XStream In GUI: %d\n" status)
            if( status == 0 then
            fprintf(outPort "Successfully completed\n" status)
            else
            fprintf(outPort "Error occured during translation\n" status)
            );if

)
```

In this example, if the function is called with status value `0`, it means that translation has completed successfully. However, if the function is called with a non-zero status value, it means that error occurred during translation.

xstOutOnCancel

```
xstOutOnCancel()
```

Description

This is a user-defined function, which is called when user presses the *Cancel* button.

Arguments

None

Value Returned

None

Example

```
procedure( xstOutOnCancel()
             sprintf("Translation canceled")
);procedure
```

xstOutOnTranslate

```
xstOutOnTranslate()
```

Description

This is a user-defined function, which is called when user presses the *Apply* or *Translate* button.

Arguments

None

Value Returned

None

Example

```
procedure( xstOutOnTranslate()
             sprintf("Starting Translation")
);procedure
```

xstOutOnCompletion

`xstOutOnCompletion(x_num)`

Description

This is a user-defined function, which is called when the translation is completed.

Arguments

x_num	It is an integer parameter. If the translation is completed without any error, this value is zero. However, if error occurs during translation, this value is non-zero.

Value Returned

None

Example

```
procedure( xstOutOnCompletion( status )

            fprintf(outPort "In xstOutOnCompletion of XStream Out GUI: %d\n"
            status)
            if( status == 0 then
            fprintf(outPort "Successfully completed\n" status)
            else
            fprintf(outPort "Error occured during translation\n" status)
            );if

)
```

In this example, if the function is called with status value `0`, translation is completed successfully. However, if the function is called with a non-zero status value, it implies that error occurred during translation.

xstInGetField

```
xstInGetField(
     t_optionName
     )
     => t_value / nil
```

Description

Enables you to access GUI field values from the *XStream In* form. You can access all the GUI field values using the appropriate option name.

Arguments

t_optionName	The name of the field that needs to be accessed from the *XStream In* form. Valid values are the command line option name of the *XStream In* form field, `virtualMemory` and `showCompletionMsgBox`. For list of valid values, see XStream In GUI and Template File Options.

Value Returned

t_value	The value returned from the *XStream In* form.
`nil`	The field name specified in the `t_optionName` argument is incorrect.

Example

Using SKILL commands, you can access the values specified in various fields, such as text box, check box, radio button, drop-down menu, and mapping tables, of the *XStream In* form. A few examples are given below:

- **Text box:** To access the value specified in the *Stream File* field of the *XStream In* form, enter the following syntax in CIW:

  ```
  xstInGetField("strmFile")
  => "test.gds"
  ```

- **Check box:** On the *Geometry* tab, if you want to access the value selected in the *Ignore Box Records* option, then you need to enter the following syntax in CIW:

  ```
  xstInGetField("ignoreBoxes")
  => "true"
  ```

This field returns a boolean value.

- **Radio Button:** On the *General* tab, if you want to access the value selected in *Label Case Sensitivity* option, then you need to enter the following syntax in CIW:

```
xstInGetField("labelCase")
=> "upper"
```

- **Drop-down Menu:** On the *General* tab, if you want to access the value selected from the *Text Namespace* drop-down menu, then you need to enter the following syntax in CIW:

```
xstInGetField("strmTextNS")
=> "verilog"
```

This field returns a string value.

- **Mapping Table:** On the *Layer* tab, if you want to access the table with the layer map entries, then you need to enter the following syntax in CIW:

```
xstInGetField("layerMap")
=> "l.map"
```

Note: Here, the `l.map` file contains the layer mapping entries.

xstGetField

```
xstGetField(
          t_optionName
          )
          => t_value / nil
```

Description

Enables you to access GUI field values from the *XStream Out* form. You can access all the GUI field values using the appropriate option name.

Arguments

t_optionName	The name of the field that needs to be accessed from the *XStream Out* form. Valid values are the command line option name of the *XStream Out* form field, `virtualMemory` and `showCompletionMsgBox`. For list of valid values, see XStream Out Option Names in GUI and Template File.

Value Returned

t_value	The value returned from the *XStream Out* form.
`nil`	The field name specified in the `t_optionName` argument is incorrect.

Example

If you want to access the value in the *Stream File* field in the *XStream Out* form, then you need to enter the following syntax in CIW:

```
xstGetField("strmFile")
=> "test.gds"
```

If you want to access the value selected in the *Virtual Memory* check box, then you need to enter the following syntax in CIW:

```
xstGetField("virtualMemory")

=> "true"
```

This field returns a boolean value.

xstInSetField

```
xstInSetField(
     t_optionName
     t_value
     )
     => t / nil
```

Description

Enables you to populate GUI field values in the *XStream In* form. You can populate all the GUI field values using the appropriate option name and value.

Arguments

t_optionName	The name of the field that needs to be populated. Valid values are the command line option name of the *XStream In* form field, `virtualMemory` and `showCompletionMsgBox`. For list of valid values, see XStream In GUI and Template File Options.
t_value	The value by which the field needs to be populated.

Value Returned

`t`	The value is populated in the *XStream In* form.
`nil`	The value is not populated in the *XStream In* form.

Example

Using SKILL commands, you can populate the values in various fields, such as text box, check box, radio button, drop-down menu, and mapping tables, of the *XStream In* form. A few examples are given below:

- **Text box:** If you want to populate the *Stream File* field in the *XStream In* form, then you need to enter the following syntax in CIW:

  ```
  xstInSetField("strmFile" "test.gds")
  ```

- **Check box:** On the *Geometry* tab, if you want to select the *Ignore Box Records* option, then you need to enter the following syntax in CIW:

  ```
  xstInSetField("ignoreBoxes" "true")
  ```

  This field accepts the boolean value.

- **Radio Button:** On the *General* tab, if you want to select the *Label Case Sensitivity* option as *upper*, then you need to enter the following syntax in CIW:

  ```
  xstInSetField("labelCase" "upper")
  ```

- **Drop-down Menu:** On the *General* tab, if you want to select the *verilog* option from the *Text Namespace* drop-down menu, then you need to enter the following syntax in CIW:

  ```
  xstInSetField("strmTextNS" "verilog")
  ```

  This field accepts the string value.

- **Mapping Table:** On the *Layer* tab, if you want to populate the table with the layer map entries, then you need to enter the following syntax in CIW:

  ```
  xstInSetField("layerMap" "l.map")
  ```

  Note: Here, the `l.map` file contains the layer mapping entries.

xstSetField

```
xstSetField(
          t_optionName
          t_value
          )
          => t / nil
```

Description

Enables you to populate GUI field values in the *XStream Out* form. You can populate all the GUI field values using the appropriate option name and value.

Arguments

t_optionName	The name of the field that needs to be populated in the *XStream Out* form. Valid values are the command line option name of the *XStream In* form field, `virtualMemory` and `showCompletionMsgBox`. For list of valid values, see XStream Out Option Names in GUI and Template File.
t_value	The value by which the field needs to be populated.

Value Returned

t	The value is populated in the *XStream Out* form.
`nil`	The value is not populated in the *XStream Out* form.

Example

If you want to populate the *Stream File* field in the *XStream Out* form, then you need to enter the following syntax in CIW:

```
xstSetField("strmFile" "test.gds")
```

However, there are some special cases where you want to strmout a copy of the design present in virtual memory. To do this, you need to select the Virtual Memory check box by using the following syntax in CIW:

```
xstSetField("virtualMemory" "true")
```

xstOutDoTranslate

```
xstOutDoTranslate()
```

Description

Issues the StreamOut command based on the GUI field values. By default, this function is a non-blocking function. However, if the Stream Out From Virtual Memory option is selected, then this function becomes a blocking function.

Note: Before executing the `xstOutDoTranslate` function, you need to either specify the values for both the `Stream File` and `Library` fields or the `Cell List File` field in the Stream Out form.

Argument

None

Value Returned

```
nil
```

Example

In this example, you need to translate library, lib1 to Stream File, out.gds. To do this, you need to execute the following functions:

```
xstSetField("strmFile" "out.gds")
xstSetField("library" "lib1")
xstOutDoTranslate()
```

xstInDoTranslate

```
xstInDoTranslate()
```

Description

Issues the StreamIn command based on the GUI field values. By default, this function is a non-blocking function. However, if the `Stream In to Virtual Memory` option is selected, then this function becomes a blocking function.

Note: Before executing the `xstInDoTranslate` function, you need to specify the value for the `Stream File` and specify the value for either `Library` fields or `Stream Tech File` field in the Stream In form.

Argument

None

Value Returned

```
nil
```

Example

```
In this example, you need to translate Stream File, in.gds to library, lib1. To do
this, you need to execute the following functions:
xstInSetField("strmFile" "in.gds")
xstInSetField("library" "lib1")
xstInDoTranslate()
```

xstInGetVMLibs

```
xstInGetVMLibs(
     )
     => l_vmLibList / nil
```

Description

Returns the list of virtual memory libraries created by XStream In. Only the primary libraries created by XStream In are returned. The additional libraries that are created if the number of cellviews is more than the value specified in the *Maximum Cells in Target Library* field are not returned by this function.

Argument

None

Value Returned

l_vmLibList	The list of virtual libraries created by XStream In.
`nil`	Returns `nil` if the function is not successful.

Example

If XStream In is done in virtual memory mode and "`vmLib`" is the library name, then this library is created in virtual memory. In this case, `xstGetVMLibs()` returns "`vmLib`".

```
xstInGetVMLibs()
=> ("vmLib")
```

xstInSaveVMLib

```
xstInSaveVMLib(
     t_vmLibName
     [ t_path ]
     )
     => t / nil
```

Description

Saves the specified virtual memory library, `t_vmLibName`, to either the specified directory, `t_path`, or the current working directory, if `t_path` is not specified. Multiple libraries are created in a single translation using XStream In if the number of cellviews is more than the value specified in the *Maximum Cells in Target Library* field. All the libraries created during a single XStream In translation are also saved.

Note: When this function is called, the Virtuoso session is blocked. All the layout editor windows corresponding to the virtual memory library cellviews are also closed. The design management (DM) preferences are also not obeyed.

Argument

t_vmLibName	The virtual library created by XStream In.
t_path	The path where the virtual library is saved.

Value Returned

`t`	The operation is successful.
`nil`	The operation is not successful.

Example

If XStream In is done in virtual memory mode and "`vmLib`" is the library name, then this library is created in the virtual memory.

Now, `xstSaveVMLib("vmLib" "/home/user")` saves the library, "`vmLib`" to the directory, "`/home/user`". If the operation is successful, this function returns "`t`".

```
xstSaveVMLib("vmLib" "/home/user")
    => t
```

Callback Functions

xstInOnCancelCB

```
xstInOnCancelCB(
     t_mode
     S_callbackFunction
     )
     => t / nil
```

Description

Registers or unregisters a user-defined callback function when you click the *Cancel* button on the XStream In form.

Arguments

t_mode

Specifies mode to register or unregister the user-defined callback function.

Valid values:

- `"r"`
  Registers the user-defined callback function.
- `"d"`
  Unregisters an already registered user-defined callback function.

S_callbackFunction

Name or function symbol of a user-defined callback function.

Value Returned

`t`	Function is successfully registered or unregistered.
`nil`	Function is not registered or unregistered.

Examples

Register the user-defined callback function `myxstInOnCancel` when the *Cancel* button is clicked on the XStream In form.

```
procedure( myxstInOnCancel()
                print("Cancel button clicked")
)
xstInOnCancelCB("r" 'myxstInOnCancel)
```

or

```
xstInOnCancelCB("r" "myxstInOnCancel")
```

Unregister the user-defined callback function `myxstInOnCancel` when the *Cancel* button is clicked on the XStream In form.

```
procedure( myxstInOnCancel()
                print("Cancel button clicked")
)
xstInOnCancelCB ("d" 'myxstInOnCancel)
```

or

```
xstInOnCancelCB("d" "myxstInOnCancel")
```

xstInOnCompletionCB

```
xstInOnCompletionCB(
     t_mode
     S_callbackFunction
     )
     => t / nil
```

Description

Registers or unregisters a user-defined callback function when the XStream In translation is complete.

Arguments

t_mode

Specifies mode to register or unregister the user-defined callback function.

Valid values:

- `"r"`
  Registers the user-defined callback function.
- `"d"`
  Unregisters an already registered user-defined callback function.

S_callbackFunction

Name or function symbol of a user-defined callback function.

Value Returned

`t`	Function is successfully registered or unregistered.
`nil`	Function is not registered or unregistered.

Examples

Register the user-defined callback function `myxstInOnCompletion` when the XStream In translation is complete.

```
procedure( myxstInOnCompletion()
                print("Translation completed with status %L")
)
```

```
xstInOnCompletionCB("r" 'myxstInOnCompletion)
```

or

```
xstInOnCompletionCB("r" "myxstInOnCompletion")
```

Unregister the user-defined callback function `myxstInOnCompletion` when the XStream In translation is complete.

```
procedure( myxstInOnCompletion()
                print("Cancel button clicked")
)
xstInOnCompletionCB ("d" 'myxstInOnCompletion)
```

or

```
xstInOnCompletionCB("d" "myxstInOnCompletion")
```

xstInOnTranslateCB

```
xstInOnTranslateCB(
     t_mode
     S_callbackFunction
     )
     => t / nil
```

Description

Registers or unregisters a user-defined callback function when you click the *Apply* or *Translate* button on the XStream In form

Arguments

t_mode

Specifies mode to register or unregister the user-defined callback function.

Valid values:

- `"r"`
  Registers the user-defined callback function.
- `"d"`
  Unregisters an already registered user-defined callback function.

S_callbackFunction

Name or function symbol of a user-defined callback function.

Value Returned

`t`	Function is successfully registered or unregistered.
`nil`	Function is not registered or unregistered.

Examples

Register the user-defined callback function `myxstInOnTranslate` when the XStream In translation is complete.

```
procedure( myxstInOnTranslate()
                print("Translate button clicked")
)
```

```
xstInOnTranslateCB("r" 'myxstInOnTranslate)
```

or

```
xstInOnTranslateCB("r" "myxstInOnTranslate")
```

Unregister the user-defined callback function `myxstInOnTranslate` when the XStream In translation is complete.

```
procedure( myxstInOnTranslate()
                print("Translate button clicked")
)
xstInOnTranslateCB ("d" 'myxstInOnTranslate)
```

or

```
xstInOnTranslateCB("d" "myxstInOnTranslate")
```

xstOutOnCancelCB

```
xstOutOnCancelCB(
     t_mode
     S_callbackFunction
     )
     => t / nil
```

Description

Registers or unregisters a user-defined callback function when you click the *Cancel* button on the XStream Out form.

Arguments

t_mode

Specifies mode to register or unregister the user-defined callback function.

Valid values:

- `"r"`
  Registers the user-defined callback function.
- `"d"`
  Unregisters an already registered user-defined callback function.

S_callbackFunction

Name or function symbol of a user-defined callback function.

Value Returned

`t`	Function is successfully registered or unregistered.
`nil`	Function is not registered or unregistered.

Examples

Register the user-defined callback function `myxstOutOnCancel` when you click the *Cancel* button on the XStream Out form.

```
procedure( myxstOutOnCancel()
                print("Cancel button clicked")
)
```

```
xstOutOnCancelCB("r" 'myxstOutOnCancel)
```

or

```
xstOutOnCancelCB("r" "myxstOutOnCancel")
```

Unregister the user-defined callback function `myxstOutOnCancel` when you click the *Cancel* button on the XStream Out form.

```
procedure( myxstInOnTranslate()
                print("Cancel button clicked")
)

xstOutOnCancelCB("d" 'myxstOutOnCancel)
```

or

```
xstOutOnCancelCB("d" "myxstOutOnCancel")
```

xstOutOnCompletionCB

```
xstOutOnCompletionCB(
     t_mode
     S_callbackFunction
     )
     => t / nil
```

Description

Registers or unregisters a user-defined callback function when the XStream Out translation is complete.

Arguments

t_mode

Specifies mode to register or unregister the user-defined callback function.

Valid values:

- `"r"`
  Registers the user-defined callback function.
- `"d"`
  Unregisters an already registered user-defined callback function.

S_callbackFunction

Name or function symbol of a user-defined callback function.

Value Returned

t	Function is successfully registered or unregistered.
nil	Function is not registered or unregistered.

Examples

Register the user-defined callback function `myxstOutOnCompletion` when the XStream Out translation is complete.

```
procedure( myxstOutOnCompletion()
                print("Cancel button clicked")
)
```

```
xstOutOnCompletionCB("r" 'myxstOutOnCompletion)
```

or

```
xstOutOnCompletionCB("r" "myxstOutOnCompletion")
```

Unregister the user-defined callback function `myxstOutOnCompletion` when the XStream Out translation is complete.

```
procedure( myxstInOnCompletion()
                print("Translation completed with status %L")
)
xstOutOnCompletionCB ("d" 'myxstOutOnCompletion)
```

or

```
xstOutOnCompletionCB("d" "myxstOutOnCompletion")
```

xstOutOnTranslateCB

```
xstOutOnTranslateCB(
     t_mode
     S_callbackFunction
     )
     => t / nil
```

Description

Registers or unregisters a user-defined callback function when you click the *Apply* or *Translate* button on the XStream Out form

Arguments

t_mode

Specifies mode to register or unregister the user-defined callback function.

Valid values:

- `"r"`
  Registers the user-defined callback function.
- `"d"`
  Unregisters an already registered user-defined callback function.

S_callbackFunction

Name or function symbol of a user-defined callback function.

Value Returned

`t`	Function is successfully registered or unregistered.
`nil`	Function is not registered or unregistered.

Examples

Register the user-defined callback function `myxstOutOnTranslate` when the XStream Out translation is complete.

```
procedure( myxstOutOnTranslate()
                print("Cancel button clicked")
)
```

```
xstOutOnTranslateCB("r" 'myxstOutOnTranslate)
```

or

```
xstOutOnTranslateCB("r" "myxstOutOnTranslate")
```

Unregister the user-defined callback function `myxstOutOnTranslate` when the XStream Out translation is complete.

```
procedure( myxstOutOnTranslate()
                print("Cancel button clicked")
)
xstOutOnTranslateCB ("d" 'myxstOutOnTranslate)
```

or

```
xstOutOnTranslateCB("d" "myxstOutOnTranslate")
```

2

XOasis Functions

This section provides syntax, descriptions, and examples for the SKILL functions associated with the XOasis translator.

There are three types of SKILL functions in XOasis:

- User-defined SKILL Functions
- GUI SKILL Functions
- Callback Functions

User-defined SKILL Functions

There are two types of user-defined SKILL functions in XOasis:

- User-defined Command Line SKILL Functions
- User-defined GUI SKILL Functions

User-defined Command Line SKILL Functions

poCellNameMap

For information related to this function, see poCellNameMap.

poLayerMap

For information related to this function, see poLayerMap.

poTextMap

For information related to this function, see poTextMap.

poParamCellNameMap

For information related to this function, see poParamCellNameMap.

poPreTranslate

For information related to this function, see poPreTranslate.

poPostTranslate

For information related to this function, see poPostTranslate.

piCellNameMap

For information related to this function, see piCellNameMap.

piLayerMap

For information related to this function, see piLayerMap.

piTextMap

For information related to this function, see piTextMap.

piPreTranslate

For information related to this function, see piPreTranslate.

piPostTranslate

For information related to this function, see piPostTranslate.

User-defined GUI SKILL Functions

xoasInOnCancel

```
xoasInOnCancel()
```

Description

This is a user-defined function, which is called when you press the *Cancel* button.

Value Returned

None

Example

```
procedure( xoasInOnCancel()
             sprintf("Translation canceled")
);procedure
```

xoasInOnTranslate

```
xoasInOnTranslate()
```

Description

This is a user-defined function, which is called when you press the *Apply* or *Translate* button.

Value Returned

None

Example

```
procedure( xoasInOnTranslate()
             sprintf("Starting Translation")
);procedure
```

xoasInOnCompletion

```
xoasInOnCompletion(
     t_num
     )
     => t_num
```

Description

This is a user-defined function, which is called when the translation is completed.

Arguments

t_num	It is an integer parameter.

Value Returned

t_num	Returns an integer value. If the translation is completed with error then it returns a non-zero value. However, if the translation is completed without any error then it returns zero.

Example

In this example, if the function returns `0`, the translation is completed successfully. However, if the function returns any non-zero value, then errors occurred during translation.

```
procedure( xoasInOnCompletion( status )
            prog( ( )
            fprintf(outPort "In xoasInOnCompletion of XOasis In GUI: %d\n" status)
            if( status == 0 then
            fprintf(outPort "Successfully completed\n" status)
            else
            fprintf(outPort "Error occured during translation\n" status)
            );if
            );prog
)
```

xoasOutOnCancel

```
xoasOutOnCancel()
```

Description

This is a user-defined function, which is called when you press the *Cancel* button.

Arguments

None

Value Returned

None

Example

```
procedure( xoasOutOnCancel()
             sprintf("Translation canceled")
);procedure
```

xoasOutOnTranslate

```
xoasOutOnTranslate()
```

Description

This is a user-defined function, which is called when you press the *Apply* or *Translate* button.

Arguments

None

Value Returned

None

Example

```
procedure( xoasOutOnTranslate()
             sprintf("Starting Translation")
);procedure
```

xoasOutOnCompletion

```
xoasOutOnCompletion(
     t_num
     )
     => t_num
```

Description

This is a user-defined function, which is called when the translation is completed.

Arguments

t_num	It is an integer parameter.

Value Returned

t_num	Returns an integer value. If the translation is completed with error then it returns a non-zero value. However, if the translation is completed without any error then it returns 0.

Example

In this example, if the function returns `0`, the translation is completed successfully. However, if the function returns any non-zero value, then the error has occurred during translation.

```
procedure( xoasOutOnCompletion( status )
            prog( ( )
            fprintf(outPort "In xoasOutOnCompletion of XOasis Out GUI: %d\n"
            status)
            if( status == 0 then
            fprintf(outPort "Successfully completed\n" status)
            else
            fprintf(outPort "Error occured during translation\n" status)
            );if
            );prog
)
```

GUI SKILL Functions

xoasInGetField

```
xoasInGetField(
     t_optionName
     )
     => t_value / nil
```

Description

Enables you to access GUI field values from the *XOasis In* form. You can access all the GUI field values using the appropriate option name.

Arguments

t_optionName	The name of the field that needs to be accessed from the *XOasis In* form. Valid values are the command line option name of the *XOasis In* form field, `virtualMemory` and `showCompletionMsgBox`. For list of valid values, see XOasis In Option Names in GUI and Template File.

Value Returned

t_value	The value returned from the *XOasis In* form
`nil`	If the field name specified in the `t_optionName` argument is incorrect

Example

Using SKILL commands, you can access the values specified in various fields, such as text box, check box, radio button, drop-down menu, and mapping tables, of the *XOasis In* form. A few examples are given below:

- **Text box:** If you want to access the value in the *OASIS File* field in the *XOasis In* form, then you need to enter the following syntax in CIW:

  ```
  xoasInGetField("oasisFile")
  => "test.gds"
  ```

- **Check box:** On the *Geometry* tab, if you want to access the value selected in the *Ignore Box Records* option, then you need to enter the following syntax in CIW:

  ```
  xoasInGetField("ignoreBoxes")
  => "true"
  ```

  This field returns a Boolean value.

- **Radio Button:** On the *General* tab, if you want to access the value selected in the *Label Case Sensitivity* option, then you need to enter the following syntax in CIW:

  ```
  xoasInGetField("labelCase")
  => "upper"
  ```

- **Mapping Table:** On the *Layer* tab, if you want to access the table with the layer map entries, then you need to enter the following syntax in CIW:

  ```
  xoasInGetField("layerMap")
  => "l.map"
  ```

Here, the `l.map` file contains the layer mapping entries.

xoasInSetField

```
xoasInSetField(
     t_optionName
     t_value
     )
     => t / nil
```

Description

Enables you to populate GUI field values in the *XOasis In* form. You can populate all the GUI field values using the appropriate option name and value.

Arguments

t_optionName	The name of the field that needs to be populated in *XOasis In* form. Valid values are the command line option name of the *XOasis In* form field, `virtualMemory` and `showCompletionMsgBox`. For list of valid values, see XOasis In Option Names in GUI and Template File.
t_value	The value by which the field needs to be populated.

Value Returned

`t`	The value is populated in the *XOasis In* form
`nil`	If the value is not populated in the *XOasis In* form

Example

Using SKILL commands, you can populate the values in various fields, such as text box, check box, radio button, drop-down menu, and mapping tables, of the *XOasis In* form. A few examples are given below:

- **Text box:** If you want to populate the *OASIS File* field in the *XOasis In* form, then you need to enter the following syntax in CIW:

  ```
  xoasInSetField("oasisFile" "test.gds")
  ```

- **Check box:** On the *Geometry* tab, if you want to select the *Ignore Box Records* option, then you need to enter the following syntax in CIW:

```
xoasInSetField("ignoreBoxes" "true")
```

This field accepts the Boolean value.

- **Radio Button:** On the *General* tab, if you want to select the *Label Case Sensitivity* option as *upper*, then you need to enter the following syntax in CIW:

  ```
  xoasInSetField("labelCase" "upper")
  ```

- **Mapping Table:** On the *Layer* tab, if you want to populate the table with the layer map entries, then you need to enter the following syntax in CIW:

  ```
  xoasInSetField("layerMap" "l.map")
  ```

  Note: Here, the `l.map` file contains the layer mapping entries.

xoasInDoTranslate

```
xoasInDoTranslate()
```

Description

Issues the XOasis In command based on the GUI field values. By default, this function is a non-blocking function. However, if the *Import OASIS into Virtual Memory* option is selected, then this function becomes a blocking function.

Note: Before executing the `xoasInDoTranslate` function, you need to specify the value for the `OASIS File` and specify the value for either `Library` fields or the `OASIS File` field in the XOasis In form.

Argument

None

Value Returned

`nil`

Example

In this example, you translate OASIS File, `in.gds`, to library, `lib1`. To do this, you need to execute the following functions:

```
xoasInSetField("oasisFile" "in.gds")
xoasInSetField("library" "lib1")
xoasInDoTranslate()
```

xoasOutGetField

```
xoasOutGetField(
     t_optionName
     )
     => t_value / nil
```

Description

Enables you to access GUI field values from the *XOasis Out* form. You can access all the GUI field values using the appropriate option name.

Arguments

t_optionName	The name of the field that needs to be accessed from the *XOasis Out* form. Valid values are the command line option name of the *XOasis Out* form field, `virtualMemory` and `showCompletionMsgBox`. For list of valid values, see XOasis Out Option Names in GUI and Template File.

Value Returned

t_value	The value returned from the *XOasis Out* form
`nil`	If the field name specified in the `t_optionName` argument is incorrect

Example

If you want to access the value in the *OASIS File* field in the *XOasis Out* form, then you need to enter the following syntax in CIW:

```
xoasOutGetField("strmFile")
=> "test.gds"
```

If you want to access the value selected in the *Export OASIS from Virtual Memory* check box, then you need to enter the following syntax in CIW:

```
xoasOutGetField("virtualMemory")

=> "true"
```

This field returns a Boolean value.

xoasOutSetField

```
xoasOutSetField(
     t_optionName
     t_value
     )
     => t / nil
```

Description

Enables you to populate GUI field values in the *XOasis Out* form. You can populate all the GUI field values using the appropriate option name and value.

Arguments

t_optionName	The name of the field that needs to be populated in the *XOasis Out* form. Valid values are the command line option name of the *XOasis Out* form field, `virtualMemory` and `showCompletionMsgBox`. For list of valid values, see XOasis Out Option Names in GUI and Template File.
t_value	The value by which the field needs to be populated.

Value Returned

`t`	The value is populated in the *XOasis Out* form.
`nil`	If the value is not populated in the *XOasis Out* form.

Example

If you want to populate the *OASIS File* field in the *XOasis Out* form, then you need to enter the following syntax in CIW:

```
xoasOutSetField("oasisFile" "test.gds")
```

However, there are some special cases where you want to translate a copy of the design present in the virtual memory. To do this, you need to select the *Export OASIS from Virtual Memory* check box by using the following syntax in CIW:

```
xoasOutSetField("virtualMemory" "true")
```

xoasOutDoTranslate

```
xoasOutDoTranslate()
```

Description

Issues the XOasis Out command based on the GUI field values. By default, this function is a non-blocking function. However, if the *Export OASIS from Virtual Memory* option is selected, then this function becomes a blocking function.

Note: Before executing the `xoasOutDoTranslate` function, you need to either specify the values for both the *OASIS File* and *Library* fields or the *Cell List File* field in the XOasis Out form.

Argument

None

Value Returned

`nil`

Example

In this example, you translate library, `lib1`, to OASIS file, `out.gds`. To do this, you need to execute the following functions:

```
xoasOutSetField("oasisFile" "out.gds")
xoasOutSetField("library" "lib1")
xoasOutDoTranslate()
```

Callback Functions

xoasInOnCancelCB

```
xoasInOnCancelCB(
     t_mode
     S_callbackFunction
     )
     t / nil
```

Description

Registers or unregisters a user-defined callback function when you click the *Cancel* button on the XOasis In form.

Arguments

t_mode

Specifies mode to register or unregister the user-defined callback function.

Valid values:

- `"r"`
  Registers the user-defined callback function.
- `"d"`
  Unregisters an already registered user-defined callback function.

S_callbackFunction

Name or function symbol of a user-defined callback function.

Value Returned

`t`	Function is successfully registered or unregistered.
`nil`	Function is not registered or unregistered.

Examples

Register the user-defined callback function `myxoasInOnCancel` when the *Cancel* button is clicked on the XOasis In form.

```
procedure( myxoasInOnCancel()
                print("Cancel button clicked")
)
xoasInOnCancelCB("r" 'myxoasInOnCancel)
```

or

```
xoasInOnCancelCB("r" "myxoasInOnCancel")
```

Unregister the user-defined callback function `myxoasInOnCancel` when the *Cancel* button is clicked on the XOasis In form.

```
procedure( myxstInOnCancel()
                print("Cancel button clicked")
)
xoasInOnCancelCB ("d" 'myxoasInOnCancel)
```

or

```
xoasInOnCancelCB("d" "myxoasInOnCancel")
```

xoasInOnCompletionCB

```
xoasInOnCompletionCB(
     t_mode
     S_callbackFunction
     )
     => t / nil
```

Description

Registers or unregisters a user-defined callback function when the XOasis In translation is complete.

Arguments

t_mode

Specifies mode to register or unregister the user-defined callback function.

Valid values:

- `"r"`
  Registers the user-defined callback function.
- `"d"`
  Unregisters an already registered user-defined callback function.

S_callbackFunction

Name or function symbol of a user-defined callback function.

Value Returned

`t`	Function is successfully registered or unregistered.
`nil`	Function is not registered or unregistered.

Examples

Register the user-defined callback function `myxoasInOnCompletion` when the XOasis In translation is complete.

```
procedure( myxoasInOnCompletion()
                print("Translation completed with status %L")
)
```

```
xoasInOnCompletionCB("r" 'myxoasInOnCompletion)
```

or

```
xoasInOnCompletionCB("r" "myxoasInOnCompletion")
```

Unregister the user-defined callback function `myxoasInOnCompletion` when the XOasis In translation is complete.

```
procedure( myxoasInOnCompletion()
                print("Cancel button clicked")
)
xoasInOnCompletionCB ("d" 'myxoasInOnCompletion)
```

or

```
xoasInOnCompletionCB("d" "myxoasInOnCompletion")
```

xoasInOnTranslateCB

```
xoasInOnTranslateCB(
     t_mode
     S_callbackFunction
     )
     => t / nil
```

Description

Registers or unregisters a user-defined callback function when you click the *Apply* or *Translate* button on the XOasis In form.

Arguments

t_mode

Specifies mode to register or unregister the user-defined callback function.

Valid values:

- `"r"`
  Registers the user-defined callback function.
- `"d"`
  Unregisters an already registered user-defined callback function.

S_callbackFunction

Name or function symbol of a user-defined callback function.

Value Returned

`t`	Function is successfully registered or unregistered.
`nil`	Function is not registered or unregistered.

Examples

Register the user-defined callback function `myxoasInOnTranslate` when the XOasis In translation is complete.

```
procedure( myxoasInOnTranslate()
                print("Translate button clicked")
)
```

```
xoasInOnTranslateCB("r" 'myxoasInOnTranslate)
```

or

```
xoasInOnTranslateCB("r" "myxoasInOnTranslate")
```

Unregister the user-defined callback function `myxoasInOnTranslate` when the XOasis In translation is complete.

```
procedure( myxstInOnTranslate()
                print("Cancel button clicked")
)
xoasInOnTranslateCB("d" 'myxoasInOnTranslate)
```

or

```
xoasInOnTranslateCB("d" "myxoasInOnTranslate")
```

xoasOutOnCancelCB

```
xoasOutOnCancelCB(
     t_mode
     S_callbackFunction
     )
     => t / nil
```

Description

Registers or unregisters a user-defined callback function when you click the *Cancel* button on the XOasis Out form.

Arguments

t_mode

Specifies mode to register or unregister the user-defined callback function.

Valid values:

- `"r"`
  Registers the user-defined callback function.
- `"d"`
  Unregisters an already registered user-defined callback function.

S_callbackFunction

Name or function symbol of a user-defined callback function.

Value Returned

`t`	Function is successfully registered or unregistered.
`nil`	Function is not registered or unregistered.

Examples

Register the user-defined callback function `myxoasOutOnCancel` when you click the *Cancel* button on the XOasis Out form.

```
procedure( myxoasOutOnCancel()
                print("Cancel button clicked")
)
```

```
xoasOutOnCancelCB("r" 'myxoasOutOnCancel)
```

or

```
xoasOutOnCancelCB("r" "myxoasOutOnCancel")
```

Unregister the user-defined callback function `myxoasOutOnCancel` when you click the *Cancel* button on the XOasis Out form.

```
procedure( myxoasOutOnCancel()
                print("Cancel button clicked")
)

xoasOutOnCancelCB("d" 'myxoasOutOnCancel)
```

or

```
xoasOutOnCancelCB("d" "myxoasOutOnCancel")
```

xoasOutOnCompletionCB

```
xoasOutOnCompletionCB(
     t_mode
     S_callbackFunction
     )
     => t / nil
```

Description

Registers or unregisters a user-defined callback function when the XOasis Out translation is complete.

Arguments

t_mode

Specifies mode to register or unregister the user-defined callback function.

Valid values:

- `"r"`
  Registers the user-defined callback function.
- `"d"`
  Unregisters an already registered user-defined callback function.

S_callbackFunction

Name or function symbol of a user-defined callback function.

Value Returned

`t`	Function is successfully registered or unregistered.
`nil`	Function is not registered or unregistered.

Examples

Register the user-defined callback function `myxoasOutOnCompletion` when the XOasis Out translation is complete.

```
procedure( myxoasOutOnCompletion()
                print("Translation completed with status %L")
)
```

```
xoasOutOnCompletionCB("r" 'myxoasOutOnCompletion)
```

or

```
xoasOutOnCompletionCB("r" "myxoasOutOnCompletion")
```

Unregister the user-defined callback function `myxoasOutOnCompletion` when the XOasis Out translation is complete.

```
procedure( myxoasOutOnCompletion()
                print("Cancel button clicked")
)
xoasOutOnCompletionCB ("d" 'myxoasOutOnCompletion)
```

or

```
xoasOutOnCompletionCB("d" "myxoasOutOnCompletion")
```

xoasOutOnTranslateCB

```
xoasOutOnTranslateCB(
     t_mode
     S_callbackFunction
     )
     => t / nil
```

Description

Registers or unregisters a user-defined callback function when you click the *Apply* or *Translate* button on the XOasis Out form

Arguments

t_mode

Specifies mode to register or unregister the user-defined callback function.

Valid values:

- `"r"`
  Registers the user-defined callback function.
- `"d"`
  Unregisters an already registered user-defined callback function.

S_callbackFunction

Name or function symbol of a user-defined callback function.

Value Returned

`t`	Function is successfully registered or unregistered.
`nil`	Function is not registered or unregistered.

Examples

Register the user-defined callback function `myxoasOutOnTranslate` when the XOasis Out translation is complete.

```
procedure( myxoasOutOnTranslate()
                print("Translate button clicked")
)
```

```
xoasOutOnTranslateCB("r" 'myxoasOutOnTranslate)
```

or

```
xoasOutOnTranslateCB("r" "myxoasOutOnTranslate")
```

Unregister the user-defined callback function `myxoasOutOnTranslate` when the XOasis Out translation is complete.

```
procedure( myxoasOutOnTranslate()
                print("Cancel button clicked")
)
xoasOutOnTranslateCB ("d" 'myxoasOutOnTranslate)
```

or

```
xoasOutOnTranslateCB("d" "myxoasOutOnTranslate")
```

3

SpiceIn Function

This section provides syntax, descriptions, and examples for the SKILL function associated with the SpiceIn translator.

spcinGuiDisplay

```
spcinGuiDisplay()
     => t / nil
```

Description

Displays the Virtuoso SpiceIn form.

An alternative way of launching the SpiceIn GUI is by selecting *File — Import — Spice* from a workbench CIW that includes the SpiceIn menu. The workbenches from which SpiceIn can be launched are `icde`, `icds`, `icms`, `virtuoso`, `layout`, `layoutPlus`, and `msfb`.

Arguments

None

Value Returned

`t`	Returns `t` on successful execution of the function.
`nil`	Returns `nil` when the function encounters an error.

4

CDL Out Functions

This section provides syntax, descriptions, and examples for the SKILL functions associated with the CDL Out translator.

hnlCDLPrintBJTElement

```
hnlCDLPrintBJTElement()
```

Description

Prints the CDL syntax of an instance of the `BJT` element in the netlist. An example of the syntax is shown below; if you are viewing this description in the SKILL API Finder, click *More Info* to see it.

```
QInstanceName C B E [SUB] cellName $EA=@area $L=@l $W=@w {$SUB=@sub}
@off ic=@ic m=@m
```

If you create a library element similar to the `BJT` element, set the following property in the CDL view of the element:
`hnlCDLFormatInst = hnlCDLPrintBJTElement()`

Arguments

None

Value Returned

None

hnlCDLPrintGeneralElement

```
hnlCDLPrintGeneralElement()
```

Description

Prints the CDL syntax of an instance of any general element. An example of the syntax is shown below; if you are viewing this description in the SKILL API Finder, click *More Info* to see it.

InstanceName O1 O2 ... *I1 I2* ... *OT1 OT2* ...

where,

Keyword	Description
O1 O2...	Nets on output terminals
I1 I2...	Nets on input terminals
OT1 OT2...	Nets on other terminals

To use this function, set the following property in the CDL view of the element:

```
hnlCDLFormatInst = hnlCDLPrintGeneralElement()
```

Arguments

None

Value Returned

None

hnlCDLPrintICIsrcElement

`hnlCDLPrintICIsrcElement()`

Description

Prints the CDL syntax of an instance of an `ICIsrc` element in the netlist. An example of the syntax is shown below; if you are viewing this description in the SKILL API Finder, click *More Info* to see it.

`F`*`InstanceName N+ N-`* `V`*`controlVoltage @value`*

If you create a library element similar to the `ICIsrc` element, set the following property in the CDL view of the element:
`hnlCDLFormatInst = hnlCDLPrintICIsrcElement()`

Arguments

None

Value Returned

None

hnlCDLPrintICVsrcElement

```
hnlCDLPrintICVsrcElement()
```

Description

Prints the CDL syntax of an instance of an `ICVsrc` element in the netlist. An example of the syntax is shown below; if you are viewing this description in the SKILL API Finder, click *More Info* to see it.

H*InstanceName* *N+* *N-* V*controlVoltage* @*value*

If you create a library element similar to the `ICVsrc` element, set the following property in the CDL view of the element:

```
hnlCDLFormatInst = hnlCDLPrintICVsrcElement()
```

Arguments

None

Value Returned

None

hnlCDLPrintCds_Thru

```
hnlCDLPrintCds_Thru()
```

Description

Prints the CDL syntax of an instance of a `Cds_Thru` element in the netlist. An example of the syntax is shown below; if you are viewing this description in the SKILL API Finder, click *More Info* to see it.

```
RInstanceName src dst @ns/100.0m $[cellName]
```

If you create a library element similar to the `Cds_Thru` element, set the following property in the CDL view of the element：

```
hnlCDLFormatInst = hnlCDLPrintCds_Thru()
```

Arguments

None

Value Returned

None

hnlCDLPrintInductorElement

```
hnlCDLPrintInductorElement()
```

Description

Prints the CDL syntax of an instance of an `Inductor` element in the netlist. An example of the syntax is shown below; if you are viewing this description in the SKILL API Finder, click *More Info* to see it.

L*InstanceName PLUS MINUS @l @tc1 @tc2 @nt* ic=*@ic*

If you create a library element similar to the `Inductor` element, set the following property in the CDL view of the element:

```
hnlCDLFormatInst = hnlCDLPrintInductorElement()
```

Arguments

None

Value Returned

None

hnlCDLPrintIsrcElement

```
hnlCDLPrintIsrcElement()
```

Description

Prints the CDL syntax of an instance of an `Isrc` element in the netlist. An example of the syntax is shown below; if you are viewing this description in the SKILL API Finder, click *More Info* to see it.

```
IInstanceName N+ N- @DCValue @TRANValue @ACMag @ACPhase
```

If you create a library element similar to the `Isrc` element, set the following property in the CDL view of the element:

```
hnlCDLFormatInst = hnlCDLPrintIsrcElement()
```

Arguments

None

Value Returned

None

hnlCDLPrintJfetElement

`hnlCDLPrintJfetElement()`

Description

Prints the CDL syntax of an instance of a `Jfet` element in the netlist. An example of the syntax is shown below; if you are viewing this description in the SKILL API Finder, click *More Info* to see it.

JInstanceName D G S cellName w=*@w* l=*@l @off* ic=*@ic* m=*@m*

If you create a library element similar to the `Jfet` element, set the following property in the CDL view of the element:
`hnlCDLFormatInst = hnlCDLPrintJfetElement()`

Arguments

None

Value Returned

None

hnlCDLPrintNMOSfetElement

```
hnlCDLPrintNMOSfetElement()
```

Description

Prints the CDL syntax of an instance of a `NMOSfet` element in the netlist. An example of the syntax is shown below; if you are viewing this description in the SKILL API Finder, click *More Info* to see it.

M*InstanceName* *D* *G* *S* *global* *gnd*, *cellName* w=*@w* l=*@l* ad=*@ad* as=*@as* pd=*@pd* ps=*@ps* nrd=*@nrd* nrs=*@nrs* *@off* ic=*@ic* m=*@m* $LDD[*@LDD*]

If you create a library element similar to the `NMOSfet` element, set the following property in the CDL view of the element:
`hnlCDLFormatInst = hnlCDLPrintNMOSfetElement()`

Arguments

None

Value Returned

None

hnlCDLPrintNPNElement

```
hnlCDLPrintNPNElement()
```

Description

Prints the CDL syntax of an instance of a `NPN` element in the netlist. An example of the syntax is shown below; if you are viewing this description in the SKILL API Finder, click *More Info* to see it.

`Q`*`InstanceName C B E cellName`* `M=@m $EA=`*`@area`*

If you create the library element similar to the `NPN` element, set the following property in the CDL view of the element:

```
hnlCDLFormatInst = hnlCDLPrintNPNElement()
```

Arguments

None

Value Returned

None

hnlCDLPrintPMOSfetElement

```
hnlCDLPrintPMOSfetElement()
```

Description

Prints the CDL syntax of an instance of a `PMOSfet` element in the netlist . An example of the syntax is shown below; if you are viewing this description in the SKILL API Finder, click *More Info* to see it.

```
MInstanceName DGS global_Vdd cellName w=@w l=@l ad=@ad as=@as pd=@pd
ps=@ps nrd=@nrd nrs=@nrs @off ic=@ic m=@m $LDD[@LDD]
```

If you create a library element similar to the `PMOSfet` element, set the following property in the CDL view of the element:
`hnlCDLFormatInst = hnlCDLPrintPMOSfetElement()`

Arguments

None

Value Returned

None

hnlCDLPrintPNPElement

```
hnlCDLPrintPNPElement()
```

Description

Prints the CDL syntax of an instance of a `PNP` element in the netlist. An example of the syntax is shown below; if you are viewing this description in the SKILL API Finder, click *More Info* to see it.

```
QInstanceName C B E cellName M=@m $EA=@area
```

If you create a library element similar to the `PNP` element, set the following property in the CDL view of the element:

```
hnlCDLFormatInst = hnlCDLPrintPNPElement()
```

Arguments

None

Value Returned

None

hnlCDLPrintResistorElement

```
hnlCDLPrintResistorElement()
```

Description

Prints the CDL syntax of an instance of a `Resistor` element in the netlist. An example of the syntax is shown below; if you are viewing this description in the SKILL API Finder, click *More Info* to see it.

```
RInstanceName PLUS MINUS @ns|@r $[cellName] m=@m {$SUB=@sub} $w=@w
$l=@l @ns @tc1 @tc2 @scale @rsh ac=@ac {$SUB=@sub}
```

If you create a library element similar to the `Resistor` element, set the following property in the CDL view of the element:
`hnlCDLFormatInst = hnlCDLPrintResistorElement()`

Arguments

None

Value Returned

None

hnlCDLPrintSchottkyTranElement

```
hnlCDLPrintSchottkyTranElement()
```

Description

Prints the CDL syntax of an instance of a `SchottkyTran` element in the netlist. An example of the syntax is shown below; if you are viewing this description in the SKILL API Finder, click *More Info* to see it.

```
CInstanceName PLUS MINUS @c $[cellName] @ns @tc1 @tc2 @scale @cj
ic=@ic m=@m {$SUB=@sub} Q name.1 C B E @NP cellname Q name.2 B C cellname
```

If you create a library element similar to the `SchottkyTran` element, set the following property in the CDL view of the element:
`hnlCDLFormatInst = hnlCDLPrintSchottkyTranElement()`

Arguments

None

Value Returned

None

hnlCDLPrintTlineElement

`hnlCDLPrintTlineElement()`

Description

Prints the CDL syntax of an instance of a `Tline` element in the netlist. An example of the syntax is shown below; if you are viewing this description in the SKILL API Finder, click *More Info* to see it.

`T`*`InstanceName N1 N2 N3 N4`* `z0=`*`@z0`* `td=`*`@td`* `f=`*`@f`* `nl=`*`@nl`* `ic=`*`@ic`*

If you create a library element similar to the `Tline` element, set the following property in the CDL view of the element:
`hnlCDLFormatInst = hnlCDLPrintTlineElement()`

Arguments

None

Value Returned

None

hnlCDLPrintVCIsrcElement

```
hnlCDLPrintVCIsrcElement()
```

Description

Prints the CDL syntax of an instance of a `VCIsrc` element in the netlist. An example of the syntax is shown below; if you are viewing this description in the SKILL API Finder, click *More Info* to see it.

```
GInstanceName N+ N- NC+ NC- @value
```

If you create a library element similar to the `VCIsrc` element, set the following property in the CDL view of the element:
`hnlCDLFormatInst = hnlCDLPrintVCIsrcElement()`

Arguments

None

Value Returned

None

hnlCDLPrintVCVsrcElement

`hnlCDLPrintVCVsrcElement()`

Description

Prints the CDL syntax of an instance of a `VCVsrc` element in the netlist. An example of the syntax is shown below; if you are viewing this description in the SKILL API Finder, click *More Info* to see it.

`E`*`InstanceName N+ N- NC+ NC- @value`*

If you create a library element similar to the `VCVsrc` element, set the following property in the CDL view of the element:
`hnlCDLFormatInst = hnlCDLPrintVCVsrcElement()`

Arguments

None

Value Returned

None

hnlCDLPrintVsrcElement

`hnlCDLPrintVsrcElement()`

Description

Prints the CDL syntax of an instance of a `Vsrc` element in the netlist. An example of the syntax is shown below; if you are viewing this description in the SKILL API Finder, click *More Info* to see it.

`V`*`InstanceName N+ N- @DCValue @TRANValue @ACMag @ACPhase`*

If you create a library element similar to the `Vsrc` element, set the following property in the CDL view of the element:
`hnlCDLFormatInst = hnlCDLPrintVsrcElement()`

Arguments

None

Value Returned

None

hnlCDLPrintMultiCNPNElement

```
hnlCDLPrintMultiCNPNElement()
```

Description

Prints the CDL syntax of an instance of a `MultiCNPN` element in the netlist. An example of the syntax is shown below; if you are viewing this description in the SKILL API Finder, click *More Info* to see it.

```
QInstanceName.1 C1 B E cellName
.
QInstanceName.n Cn B E cellName
```

If you create a library element similar to the `MultiCNPN` element, set the following property in the CDL view of the element:

```
hnlCDLFormatInst = hnlCDLPrintMultiCNPNElement()
```

Arguments

None

Value Returned

None

hnlCDLPrintMultiCPNPElement

```
hnlCDLPrintMultiCPNPElement()
```

Description

Prints the CDL syntax of an instance of a `MultiCPNP` element in the netlist. An example of the syntax is shown below; if you are viewing this description in the SKILL API Finder, click *More Info* to see it.

```
QInstanceName.1 C1 B E cellName
.
QInstanceName.n Cn B E cellName
```

If you create a library element similar to the `MultiCPNP` element, set the following property in the CDL view of the element:

```
hnlCDLFormatInst = hnlCDLPrintMultiCPNPElement()
```

Arguments

None

Value Returned

None

hnlCDLPrintMultiENPNElement

```
hnlCDLPrintMultiENPNElement()
```

Description

Prints the CDL syntax of an instance of a `MultiENPN` element in the netlist. An example of the syntax is shown below; if you are viewing this description in the SKILL API Finder, click *More Info* to see it.

```
QInstanceName.1 C1 B E cellName
.
QInstanceName.n Cn B En cellName
```

If you create a library element similar to the `MultiENPN` element, set the following property in the CDL view of the element:

```
hnlCDLFormatInst = hnlCDLPrintMultiENPNElement()
```

Arguments

None

Value Returned

None

hnlCDLPrintMultiEPNPElement

```
hnlCDLPrintMultiEPNPElement()
```

Description

Prints the CDL syntax of an instance of a `MultiEPNP` element in the netlist. An example of the syntax is shown below; if you are viewing this description in the SKILL API Finder, click *More Info* to see it.

```
QInstanceName.1 C1 B E cellName
.
QInstanceName.n Cn B En cellName
```

If you create a library element similar to the `MultiEPNP` element, set the following property in the CDL view of the element:

```
hnlCDLFormatInst = hnlCDLPrintMultiEPNPElement()
```

Arguments

None

Value Returned

None

hnlCDLPrintCapElement

```
hnlCDLPrintCapElement()
```

Description

Prints the CDL syntax of an instance of a `cap` element in the netlist. An example of the syntax is shown below; if you are viewing this description in the SKILL API Finder, click *More Info* to see it.

```
CInstanceName Y global_gnd @c m=@m $[cellName] {$SUB=@sub} @ns @tc1
@tc2 @scale @cj ic=@ic area=@area l=@l w=@w
```

If you create a library element similar to the `cap` element, set the following property in the CDL view of the element:

```
hnlCDLFormatInst = hnlCDLPrintCapElement()
```

Arguments

None

Value Returned

None

hnlCDLPrintCapacitorElement

```
hnlCDLPrintCapacitorElement()
```

Description

Prints the CDL syntax of an instance of a `capacitor` element in the netlist. An example of the syntax is shown below; if you are viewing this description in the SKILL API Finder, click *More Info* to see it.

format:
`C`*`InstanceName PLUS MINUS @c`* `$[`*`cellName`*`]` {*`$SUB=@sub`*} *`@ns @tc1 @tc2 @scale @cj`* `ic=`*`@ic`* `m=`*`@m`* `{$SUB=`*`@sub`*`}` *`area=@area l=@l w=@w`*

If you create a library element similar to the `capacitor` element, set the following property in the CDL view of the element：
```
hnlCDLFormatInst = hnlCDLPrintCapacitorElement()
```

Arguments

None

Value Returned

None

hnlCDLPrintDiodeElement

```
hnlCDLPrintDiodeElement()
```

Description

Prints the CDL syntax of an instance of a `diode` element in the netlist. An example of the syntax is shown below; if you are viewing this description in the SKILL API Finder, click *More Info* to see it.

```
DInstanceName PLUS MINUS cellName AREA=@area PJ=@pj w=@w l=@l
wp=@wp lp=@lp wm=@wm @off ic=@ic {@area} {@periphery}
{$SUB=@sub}
```

If you create a library element similar to the `diode` element, set the following property in the CDL view of the element:
`hnlCDLFormatInst = hnlCDLPrintDiodeElement()`

Arguments

None

Value Returned

None

hnlCDLPrintBSIM3SOIElement

`hnlCDLPrintBSIM3SOIElement()`

Description

Prints the CDL syntax of an instance of a `BSIM3SOI` element in the netlist. An example of the syntax is shown below; if you are viewing this description in the SKILL API Finder, click *More Info* to see it.

M<*name*> <D> <G> <S> <E> <model> l=*@l* w=*@w* ad=*@ad* as=*@as* pd=*@pd* ps=*@ps* nrs=*@nrs* nrd=*@nrd* nrb=*@nrb* *@off* bjtoff=*@bjtoff* ic=*@ic* rtho=*@rtho* ctho=*@ctho* debug=*@debug* nbc=*@nbc* nseg=*@nseg* pdbcp=*@pdbcp* psbcp=*@psbcp* agbcp=*@agbcp* aebcp=*@aebcp* vbsusr=*@vbsusr* tnodeout=*@tnodeout*

If you create a library element similar to the `BSIM3SOI` element, set the following property in the CDL view of the element:
`hnlCDLFormatInst = hnlCDLPrintBSIM3SOIElement()`

Arguments

None

Value Returned

None

hnlCDLPrintResElement

```
hnlCDLPrintResElement()
```

Description

Prints the CDL syntax of an instance of a `Res` element in the netlist. An example of the syntax is shown below; if you are viewing this description in the SKILL API Finder, click *More Info* to see it.

```
RInstanceName A Y @ns|@r $[cellName] $SUB=@sub $w=@w $l=@l @ns @tc1
@tc2 @scale @rsh ac=@ac m=@m
```

If you create a library element similar to the `Res` element, set the following property in the CDL view of the element:
`hnlCDLFormatInst = hnlCDLPrintResElement()`

Arguments

None

Value Returned

None

hnlCDLPrintInstPropVal

```
hnlCDLPrintInstPropVal(
     t_propName
     )
```

Description

Prints an instance property value in the CDL netlist during netlisting.

Note: This function must be called only from the user-defined instance formatting procedures.

Arguments

t_propName	The instance property name to print in CDL netlist.

Value Returned

None

Examples

```
hnlCDLPrintInstPropVal(" $EA=%s" "area")
```

Prints `area` as the value of the property `$EA`.

```
hnlCDLPrintInstPropVal(" $L=%s" "l")
```

Prints `1` as the value of the property `$L`.

```
hnlCDLPrintInstPropVal(" $W=%s" "w")
```

Prints `w` as the value of the property `$W`.

```
hnlCDLPrintInstPropVal(" $SUB=%s" "sub")
```

Prints `sub` as the value of the property `$SUB`.

```
hnlCDLPrintInstPropVal(" off" "off")
```

Prints `off` as the value of the property `off`.

```
hnlCDLPrintInstPropVal(" ic=%s" "ic")
```

Prints `ic` as the value of the property `ic`.

transCdlOutDisplay

```
transCdlOutDisplay()
```

Description

Invokes the *CDL Out* GUI form.

Arguments

None

Value Returned

`t`	Returns t when Ok button is clicked in the CDL Out form.

5

LEF/DEF Functions

This section provides syntax, descriptions, and examples for the SKILL functions associated with the LEF/DEF translator.

ldtrLefReadOA

```
ldtrLefReadOA(
     t_fileName
     t_libName
     [ t_libPath ]
     [ t_techName ]
     [ t_techPath ]
     [ g_overwriteTech { t | nil } ]
     [ g_shared { t | nil } ]
     [ t_viewName ]
     [ t_logName logName ]
     [ t_layermapFileName ]
     [ t_commentChar ]
     [ t_templateFileName ]
     [ t_pinpurp ]
     [ t_textLayer ]
     [ t_textHeight ]
     [ t_techRefs ]
     [ g_compress { t | nil }]
     [ t_compressLevel ]
     [ g_mapConflicts { t | nil }]
     [ g_pnrLibDataOnly { t | nil } ]
     [ g_useFoundryInnovus { t | nil } ]
     [ g_useTextLayerFromPin {t |nil} ]
     [ t_textPurposeName ]
     [ g_keepOrigDisplayDrf {t |nil} ]
     [ g_createFixedViaDefs ]
     [ t_verilogStubFile ]
     [ g_lockColorData { t | nil } ]
     [ t_suffix ]
     [ g_widthAndRatio { t | nil } ]
     )
     => t / nil
```

Description

Reads the specified LEF file into an OpenAccess library. `t_lockColorData` is an advanced nodes only argument.

Arguments

t_fileName	Name of the LEF file to be read into the OpenAccess library.
t_libName	Name of the output OpenAccess library. If the output library does not exist, a new library is created in the current directory and contains the translated technology database and macros. If the output library exists, it must contain a technology database or refer to the technology database of another library. In addition, any existing macros, if redefined in the LEF file, are overwritten and new cells are created for the new macros.
t_libPath	Complete path where the OpenAccess library is to be created.
t_techName	Name of the technology library that is to be created.
t_techpath	Path to technology library.
g_overwriteTech	Overwrite the existing technology file. Specify `nil` to update the technology file.
g_shared	Share the OA library with other applications while the current application is running.
t_viewName	View name for the translated macros. The default view name is abstract.
t_logName	Name of the log file for storing the error messages.
t_layermapFileName	Name of the layer map file. While creating layers, the layer numbers specified in this file are used.
t_commentChar	Special character that identifies comments in the LEF file.
t_templateFileName	Name of the template file that contains all the command options.
t_pinpurp	Purpose for pin geometries.
t_textLayer	Defines the text layer name to use for pin labels.

t_textHeight	Defines the text height to use for pin labels.
t_techRefs	List of libraries that contain master technology databases.
g_compress	Allow libraries to be compressed.
t_compressLevel	Define the compression level to use. Default value: 1
g_mapConflicts	Map certain conflicts between incremental and base technologies to constraints and constraint parameter definitions.
g_pnrLibDataOnly	Filter the Place and Route information. The information can be stored only as incremental technology database on the top of existing referenced technology database. Therefore, the *t_techRefs* option needs to be specified. For more information, see General Processing Rules of -pnrLibDataOnly. This option cannot be specified along with *g_useFoundryInnovus* option. **Note:** The *g_pnrLibDataOnly* argument works for 20 nm and larger designs. Cadence recommends using the *g_useFoundryInnovus* argument, instead.
g_useFoundryInnovus	Enables Innovus to see the same design rules that are described in LEF without impacting the Virtuoso tools. For more information, see General Processing Rules of useFoundryInnovus. This option cannot be specified along with *g_pnrLibDataOnly* option.
g_useTextLayerFromPin	Specify the text layer name from pin.
t_textPurposeName	Specify the purpose name.
g_keepOrigDisplayDrf	Specify whether the original `display.drf` file is retained.
g_createFixedViaDefs	Enables creation of fixed via definitions.

t_verilogStubFile	Specify list of input Verilog files
g_lockColorData	Lock color data of all shapes created using the PORT and PIN definitions and are associated with a MASK construct.
t_suffix	Appends the specified suffix to the names of the standard constraint groups, `LEFDefaultRouteSpec`, `LEFSpecialRouteSpec`, and `foundry_innovus`. These suffixed constraint groups are then used during LEF In translation, instead of the standard constraint groups. **Note:** When you use this option, you can import the LEF file into different constraint group of existing technology without using `-techRefs` option.
g_widthAndRatio	Specifies whether values specified for the LEF MACRO pin property LEF58_ANTENNADIFFAREA should be translated as width and ratio. The default is `nil`.

Value Returned

`t`	Returns `t` on successful execution of the function.
`nil`	Returns `nil` when the function encounters an error.

Example

```
ldtrLefReadOA("tech.lef macro1.lef macro2.lef macro3.lef" "techPlusDesignLib" ""
"" "" nil nil "abstract" "" "layer.map" "" "" "" "" "" "" t "7" )
```

In the above example, four LEF files are read into the `techPlusDesignLib` library with compress option enabled and compress level set to `7`.

ldtrLefWriteOA

```
ldtrLefWriteOA(
     t_fileName
     t_libName
     [ t_cellNames ]
     [ t_cellListFileName ]
     [ t_viewNames ]
     [ t_logName ]
     [ g_noTech { t | nil } ]
     [ t_version ]
     [ g_techOnly { t | nil } ]
     [ t_templateFileName ]
     [ g_useFoundryInnovus { t | nil}]
     [ g_lockedColorOnly { t | nil }]
     [ t_suffix ]
     [ g_widthAndRatio { t | nil } ]
     )
     => t / nil
```

Description

Writes a LEF file from a specified OpenAccess library. `t_lockedColorOnly` is an advanced nodes only argument.

Arguments

t_fileName	Name of the LEF file to be created from the OpenAccess library.
t_libName	Name of the source OpenAccess library.
t_cellNames	Name of the cells to be translated as a macro. You can specify more than one cell name.
t_cellListFileName	Name of the file containing a list of cell names to be translated as macros. The cell list file contains the list of library names and cell names separated by a space. In a cell list file for Virtuoso design environment on OpenAccess 2.2 you do not need to specify the view name. The cell names in a cell list file appear in the following format: *<libName> <cellName>*

A sample cell list file is shown below:

```
 #libName          cellName
designLib1        ACCSHCINX2
designLib1        ACCSHCINX4
designLib1        ACCSHCONX2
```

If you specify the cell names as well as a cell list file as arguments, then the cell list file is used. The specified cell names are ignored and a warning message is displayed.

t_viewNames View names for the input cell names.

t_logName Name of the log file for storing the error messages.

g_noTech Specify `t` if you do not want to output the technology information.

t_version Specify the version of LEF. Supported versions are 6.0, 5.8, 5.7, 5.6, 5.5, and 5.4.

g_techOnly Output only the technology information and cell, cell list, and cell view options are ignored.

t_templateFileName Name of the file that contains all the command options.

g_useFoundryInnovus

Output the `foundry_innovus` group constraint as a LEF LAYER LEF58 property.

g_lockedColorOnly Output colors for only those shapes that have their locked state set to true.

This includes pathSegs in nets, rects and polygons, and fill shapes and vias in specialnets.

t_suffix Appends the specified suffix to the names of the standard constraint groups, `LEFDefaultRouteSpec`, `LEFSpecialRouteSpec`, and `foundry_innovus` when searching for the standard constraint groups.

g_widthAndRatio Specifies whether values specified for the LEF MACRO pin property LEF58_ANTENNADIFFAREA should be created as width and ratio from constraint.

The default is `nil`.

Value Returned

`t`	Returns `t` on successful execution of the function.
`nil`	Returns `nil` when the function encounters an error.

Example

```
ldtrLefWriteOA("out.lef.tech" "techPlusDesignLib" "" "" "" "" nil "5.4" t)
```

In the above example, a LEF file, `out.lef.tech`, is created from the `techPlusDesignLib` library.

ldtrDefReadOA

```
ldtrDefReadOA(
     t_fileName
     t_libName
     [ t_libPath ]
     [ t_cellName ]
     [ t_viewName ]
     [ t_techName ]
     [ t_viewNameList ]
     [ t_masterLibs ]
     [ g_shared  { t | nil } ]
     [ g_noRouting  { t | nil } ]
     [ t_logName ]
     [ g_useCustomVias  { t | nil } ]
     [ g_overwrite  { t | nil } ]
     [ g_createModHier  { t | nil } ]
     [ t_commentChar commentChar ]
     [ t_templateFileName ]
     [ t_layermapFileName ]
     [ t_techRefs ]
     [ t_pinpurp ]
     [ g_compress { t | nil } ]
     [ t_compressLevel ]
     [ g_ignoreDrcFill { t | nil } ]
     [ t_userSkillFile ]
     [ g_lockColorData { t | nil } ]
     [ t_oaMapFile ]
     [ t_alternateFoundry constraintGroupName ]
     )
     => t / nil
```

Description

Reads the specified DEF file into an OpenAccess library. `t_lockColorData` is an advanced nodes only argument.

Arguments

t_fileName	Name of the DEF file to be read into the OpenAccess library.
t_libName	Name of the target OpenAccess library.
t_libPath	Complete path where the OpenAccess library is to be created.
t_cellName	Output cell name.
t_viewName	View name for the translated design.
t_techName	Name of the technology library to be attached to the output OpenAccess library.
t_viewNameList	List of view names for the master cells, which can be searched for the DEF `COMPONENTS` construct record.
t_masterLibs	List of reference libraries that contain master cells.
g_shared	Shares the input-output library with other applications while the current application is running.
g_noRouting	Specify `t` to ignore any routing data including the vias section.
t_logName	Name of the log file for storing the error messages.
g_useCustomVias	Create custom vias for all DEF vias.
g_overwrite	Overwrite the existing design in the target library.
g_createModHier	Specify this as `nil` to create only the physical data. No module hierarchy is derived from the DEF hierarchical names.
t_commentChar	Special character that identifies comments in the DEF file.
t_templateFileName	Name of the template file that contains all the command options.

t_layermapFileName	Name of the layer map file. While creating layers, the layer numbers specified in this file are used.
t_techRefs	List of libraries that contain master technology databases.
t_pinpurp	Purpose for pin geometries.
g_compress	Allow libraries to be compressed.
t_compressLevel	Define the compression level to use. Default value: 1
g_ignoreDrcFill	Map all shapes specified with the `+SHAPE DRCFILL` tag in the `SPECIALNETS` section to the `drawing` purpose. If this option is not specified, such shapes are mapped to the `gapFill` purpose. **Note:** When *`lockColorData`* is specified, the `ldtrDefReadOA` SKILL function is incompatible with ICADV12.2 ISR6 and earlier versions. For more information, see example.
t_userSkillFile	Specifies a file that consists of user-defined SKILL routines.
g_lockColorData	Lock color data of all shapes created using the PORT and PIN definitions and are associated with a MASK construct.
t_oaMapFile	Uses the specified mapping file to post-process the design before it is saved. For more information, see Using the oaMapFile option for different OpenAccess Database Formats in *Design Data Translators Reference*.
t_alternateFoundry	Uses the specified constraint group instead of the default foundry constraint group. Constraint group specified using this option, overwrites the name specified in the .cdsenv file using the AlternateFoundryCG environment variable.

Value Returned

`t`	Returns `t` on successful execution of the function.
`nil`	Returns `nil` when the function encounters an error.

Examples

Example 1

In the following example, three DEF files are read into the `definLib` library with compress option enabled and compressLevel set to `8`.

```
ldtrDefReadOA("sroutes01.def sroutes02.def sroutes03.def" "definLib"
"" "defCell" "layout" "" "layout abstract autoLayout route" "techLib
techPlusDesignLib" t t "" t t t "" "" "" "" t "8")
```

Example 2

The following example shows how the code changes when *`t_ignoreDrcFill`* is specified in a code that already specifies *`t_lockedColorData`*.

In the following ICADV12.2 ISR6 or earlier version code, **`t`** is the 22nd value that specifies value for *`t_lockedColorData`*.

```
ldtrDefReadOA("defin.def" "lib" "" "design" "layout" "" "" "" nil nil "defin.log"
nil nil nil "" "" "" "" "" nil "" t)
```

From ICADV12.2 ISR7 onward, the function also supports *`t_ignoreDrcFill`*. Therefore, the code that previously specified *`t_lockedColorData`* will have to be changed. As you can see below, in the updated code, the 22nd value **`nil`** now specifies the value for *`t_ignoreDrcFill`*, and the 23rd value **`t`** specifies value for *`t_lockedColorData`*.

```
ldtrDefReadOA("defin.def" "lib" "" "design" "layout" "" "" "" nil nil "defin.log"
nil nil nil "" "" "" "" "" nil "" nil t)
```

The updated code is incompatible with ICADV12.2 IRSR6 and earlier versions.

ldtrDefWriteOA

```
ldtrDefWriteOA(
     t_fileName
     t_libName
     t_cellName
     t_viewName
     [ t_logName ]
     [ t_version ]
     [ t_templateFileName ]
     [ g_skipNoneOrViaCellType ]
     [ t_mapDividerChar ]
     [ t_dividerChar ]
     [ t_busbitChars ]
     [ g_outputFloatingShapes { t | nil } ]
     [ g_warnOnNotPlacedInsts { t | nil } ]
     [ g_skipPhysOnlyInsts { t | nil }]
     [ g_gdsCompatible ]
     [ t_userSkillFile ]
     [ t_noDefOnError { t | nil } ]
     [ g_lockedColorOnly { t | nil } ]
     [ g_skipTrimShapes { t | nil } ]
     [ g_skipTrimProductShapes { t | nil } ]
     [ g_genBridgeMetalShapes { t | nil } ]
     [ g_outputTrimSegsAsNets { t | nil } ]
     [ t_maskShiftLayer ]
     [ t_oaMapFile ]
     [ g_skipTrimmedShapes { t | nil } ]
     [ t_errorOnGrayShapes { t | nil } ]
     [ t_errorOnUnlockedShapes { t | nil } ]
     )
     => t / nil
```

Description

Creates a DEF file from a specified OpenAccess library.

Arguments

t_fileName	Name of the DEF file to be created from the OpenAccess library.
t_libName	Name of the source OpenAccess library.
t_cellName	Name of the cell to be translated.
t_viewName	View name for the cell name to be translated.

t_logName	Name of the log file for storing the error messages.
t_version	Specify the version of DEF. Supported versions are 6.0, 5.8, 5.7, 5.6, 5.5, and 5.4.
t_templateFileName	Name of the file that contains all the command options.
g_skipNoneOrViaCellType	Cells of type `none` or `via` will not be written in output DEF file.
t_mapDividerChar	Specifies a single character to be replaced in instance and net names by the DEF.
t_dividerChar	Specifies a new divider character to be used during output.
t_busbitChars	Specifies new busbit character pair to be used during output.
g_outputFloatingShapes	Outputs floating shapes into the `SPECIALNETS` section. **Note:** Floating shapes are shapes that do not have any connectivity information. All shapes on `tsvMetal` and `padMetal` layers that do not have any connectivity are also considered as floating shapes. When specified during DEF OUT translation, any drawing shape on a layer, which is not a member of a net is exported on a net in the SPECIALNETS section named `_FLOATING_DRAWING_SHAPES_RESERVED`.
g_warnOnNotPlacedInsts	Adds warnings in the log file for the instances in the DEF Out output that have no placement status.
g_skipPhysOnlyInsts	Prevents output of physical instances. If this option is not specified, physical instances are placed in the `DEF COMPONENTS` section and have the `+ SOURCE DIST` statement added to them.
g_gdsCompatible	Generates a DEF file with limits that make it compatible with GDS.
t_userSkillFile	Specifies a file that consists of user-defined SKILL routines.

t_noDefOnError	Prevents the DEF output file from being generated if there are errors during translation.
g_lockedColorOnly'	Outputs colors for only those shapes that have their locked state set to true. This includes pathSegs in nets, rects and polygons, and fill shapes and vias in specialnets.
g_skipTrimShapes	Prevents output of trim layers shapes into the FILLS section. **Note:** This option works only when the `-outputFloatingShapes` option is specified.
g_skipTrimProductShapes	Prevents output of trim product shapes into the SPECIALNETS section.
g_genBridgeMetalShapes	Outputs the trim gap fill shapes (bridge metal shapes) into the `SPECIALNETS` section.
g_outputTrimSegsAsNets	Outputs trim product shapes into NETS section. *Important* Arguments *t_skipTrimProductShapes* and *t_outputTrimSegsAsNets* cannot set to `t` together.
t_maskShiftLayer	Specifies list of layers used in COMPONENTMASKSHIFT statement.
t_oaMapFile	Uses the specified mapping file to pre-process the design before it is exported to DEF. For more information, see Using the oaMapFile option for different OpenAccess Database Formats in *Design Data Translators Reference*.
g_skipTrimmedShapes	Prevents output of trim layers shapes into the `SPECIALNETS` section. **Note:** This option works only when the `g_outputFloatingShapes` option is specified. ‘

t_errorOnGrayShapes	Displays errors if gray or uncolored shapes or vias are found during translation. **Note:** (Virtuoso Advanced Node for Layout Only) When the technology library contains preColoredLayers constraint and in the VirtuosoMPTSetup constraint group, the layers list from the first preColoredLayers constraint is not considered for error checking. Additionally, colorless layers and colorless LPPs are not considered for error checking.
t_errorOnUnlockedShapes	Displays errors if any colored shapes or vias with unlocked status are found during translation. **Note:** (Virtuoso Advanced Node for Layout Only) When the technology library contains preColoredLayers constraint and in the VirtuosoMPTSetup constraint group, the layers list from the first preColoredLayers constraint is not considered for error checking. Additionally, colorless layers and colorless LPPs are not considered for error checking.

Value Returned

t	Returns t on successful execution of the function.
nil	Returns nil when the function encounters an error.

Examples

Example 1

```
ldtrDefWriteOA("out.def" "definLib" "defCell" "layout" "defout.log" "5.6")
```

In this example, a DEF file, out.def, is created from the definLib library.

Example 2

```
ldtrDefWriteOA("out.def" "definLib" "defCell" "layout" "defout.log "5.8" "" nil ""
"" "" t nil nil t t t nil)
```

In the above example, a DEF file, `out.def`, is created from the `definLib` library with boolean options `t_allowFloatingShapes`, `t_skipTrimShapes`, `t_skipTrimProductShapes`, and `t_genBridgeMetalShapes` set to `t`.

This example is valid only for ICADV12.3 ISR9 version. It is incompatible with older and newer versions of this function. Argument `t_allowFloatingShapes`, which was fifteenth argument in previous releases has been moved to twelfth place. Therefore, a block of code that already specified one or more of the following options will have to be modified to work correctly in ICADV12.3 ISR9:

- *`g_lockedColorOnly`* (Moved to thirteenth place from twelfth)
- *`g_skipTrimShapes`* (Moved to fourteenth place from thirteenth)
- *`g_skipTrimProductShapes`* (Moved to fifteenth place from fourteenth)
- *`g_genBridgeMetalShapes`*
- *`g_outputTrimSegsAsNets`*

Example 3

```
ldtrDefWriteOA("out.def" "definLib" "defCell" "layout" "defout.log "5.8" "" nil ""
"" "" nil t)
```

In the above example, a DEF file, `out.def`, is created from the `definLib` library with Boolean option `t_warnOnNotPlacedInsts` set to `t`.

This example is valid only for ICADV12.3 ISR10. It is incompatible with the older versions of this function. Argument `t_warnOnNotPlacedInsts` has been added as the thirteenth argument. Therefore, a block of code that already specifies one or more of the following options will have to be modified to work correctly ICADV12.3 ISR10 onward:

- *`g_lockedColorOnly`* (Moved to fourteenth place from thirteenth)
- *`g_skipTrimShapes`* (Moved to fifteenth place from fourteenth)
- *`g_skipTrimProductShapes`* (Moved to sixteenth place from fifteenth)
- *`g_genBridgeMetalShapes`* (Moved to seventeenth place from sixteenth)
- *`g_outputTrimSegsAsNets`* (Moved to eighteenth place from seventeenth)

Command-Line SKILL Functions

defoutPreTranslate

```
defoutPreTranslate(
     lib
     cell
     view
     )
```

Description

During DEF Out, this function is called just before the translation starts.

Arguments

lib	Contains the input library specified by the user.
cell	Contains the cell specified by the user.
view	Contains the view specified by the user.

Value Returned

None

Example

```
procedure( defoutPreTranslate( lib cell view )
 prog( ( )
   printf("In PreTranslate of defout")
   printf("Library: %s\n" lib)
   printf("Cell: %s\n" cell)
   printf("View: %s\n" view)
    );prog
  )
```

defoutPostTranslate

```
defoutPostTranslate(
     lib
     cell
     view
     )
```

Description

During DEF Out, this function is called just after the translation is completed.

Arguments

lib	Contains the input library specified by the user.
cell	Contains the cell specified by the user.
view	Contains the view specified by the user.

Value Returned

None

Example

```
procedure( defoutPostTranslate( lib cell view )
    prog( ( )
    printf("In PostTranslate of defout\n")
    printf("Library: %s\n" lib)
    printf("Cell: %s\n" cell)
    printf("View: %s\n" view)
    );prog
 )
```

definPreTranslate

```
definPreTranslate(
     lib
     cell
     view
     )
```

Description

During DEF In, this function is called just before the translation starts.

Arguments

lib	Contains the destination library specified by the user.
cell	Contains the cell specified by the user or empty string if not specified.
view	Contains the view specified by the user or the default view `layout` is used.

Value Returned

None

Example

```
procedure( definPreTranslate( lib cell view )
  prog( ( )
   printf("In PreTranslate of defin\n")
   printf("Library: %s\n" lib)
   printf("Cell: %s\n" cell)
   printf("View: %s\n" view)
  );prog
)
```

definPostTranslate

```
definPostTranslate(
     lib
     cell
     view
     )
```

Description

During DEF In translation, this function is called just after the translation is completed.

Arguments

lib	Contains the destination library specified by the user.
cell	Contains the cell specified by the user or empty string if not specified.
view	Contains the view specified by the user or the default view `layout` is used.

Value Returned

None

Example

```
procedure( definPostTranslate( lib cell view )
  prog( ( )
   printf("In PostTranslate of DefIn\n")
   printf("Library: %s\n" lib)
   printf("Cell: %s\n" cell)
   printf("View: %s\n" view)
  );prog
)
```