Virtuoso EDIF 200 Reader and Writer SKILL Reference

Product Version ICADVM20.1
October 2020

Contents

Preface

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 manual contains reference information for SKILL functions associated with the following Cadence products:

- EDIF 200 In, which translates files from the EDIF 200 format into the Cadence® Design Framework II (DFII) database format.
- EDIF 200 Out, which translates files from the DFII format into the EDIF 200 format.

This manual is aimed at developers and designers of integrated circuits and assumes that you are familiar with the EDIF 200 language.

This preface contains the following topics:

Scope

Unless otherwise noted, the functionality described in this guide can be used in both mature node (for example, IC6.1.8) and advanced node and methodologies (for example, ICADVM20.1) releases.

Label	Meaning
(ICADVM20.1 Only)	Features supported only in the ICADVM20.1 advanced nodes and advanced methodologies release.
(IC6.1.8 Only)	Features supported only in mature node releases.

Licensing Requirements

For information on licensing in the Virtuoso design environment, see *Virtuoso Software Licensing and Configuration Guide*.

Related Documentation

Installation, Environment, and Infrastructure

- *Cadence Installation Guide*
- *Virtuoso Design Environment User Guide*
- *Cadence SKILL Language User Guide*
- *Cadence SKILL Language Reference*

Virtuoso Tools

- *Virtuoso EDIF 200 Reader and Writer User Guide*
- *Virtuoso Schematic Editor User Guide*
- *Virtuoso NC Verilog Environment User Guide*

Additional Learning Resources

Video Library

The Video Library on the Cadence Online Support website provides a comprehensive list of videos on various Cadence products.

To view a list of videos related to a specific product, you can use the *Filter Results* feature available in the pane on the left. For example, click the *Virtuoso Layout Suite* product link to view a list of videos available for the product.

You can also save your product preferences in the Product Selection form, which opens when you click the *Edit* icon located next to *My Products*.

Virtuoso Videos Book

You can access certain videos directly from Cadence Help. To learn more about this feature and to access the list of available videos, see Virtuoso Videos.

Rapid Adoption Kits

Cadence provides a number of Rapid Adoption Kits that demonstrate how to use Virtuoso applications in your design flows. These kits contain design databases and instructions on how to run the design flow.

In addition, Cadence offers the following training courses on SKILL:

- SKILL Language Programming Introduction
- SKILL Language Programming
- Advanced SKILL Language Programming

To explore the full range of training courses provided by Cadence in your region, visit Cadence Training or write to training_enroll@cadence.com.

Note: The links in this section open in a separate web browser window when clicked in Cadence Help.

Help and Support Facilities

Virtuoso offers several built-in features to let you access help and support directly from the software.

- The Virtuoso *Help* menu provides consistent help system access across Virtuoso tools and applications. The standard Virtuoso *Help* menu lets you access the most useful help and support resources from the Cadence support and corporate websites directly from the CIW or any Virtuoso application.
- The Virtuoso Welcome Page is a self-help launch pad offering access to a host of useful knowledge resources, including quick links to content available within the Virtuoso installation as well as to other popular online content.

  The Welcome Page is displayed by default when you open Cadence Help in standalone mode from a Virtuoso installation. You can also access it at any time by selecting *Help – Virtuoso Documentation Library* from any application window, or by clicking the *Home* button on the Cadence Help toolbar (provided you have not set a custom home page).

For more information, see Getting Help in *Virtuoso Design Environment User Guide.*

Customer Support

For assistance with Cadence products:

- Contact Cadence Customer Support

  Cadence is committed to keeping your design teams productive by providing answers to technical questions and to any queries about the latest software updates and training needs. For more information, visit https://www.cadence.com/support.
- Log on to Cadence Online Support

  Customers with a maintenance contract with Cadence can obtain the latest information about various tools at https://support.cadence.com.

Feedback about Documentation

You can contact Cadence Customer Support to open a service request if you:

- Find erroneous information in a product manual
- Cannot find in a product manual the information you are looking for

- Face an issue while accessing documentation by using Cadence Help

You can also submit feedback by using the following methods:

- In the Cadence Help window, click the *Feedback* button and follow instructions.
- On the Cadence Online Support Product Manuals page, select the required product and submit your feedback by using the *Provide Feedback* box.

Understanding Cadence SKILL

Cadence SKILL is a high-level, interactive programming language based on the popular artificial intelligence language, Lisp. It lets you customize and extend your design environment. Using SKILL you can validate the steps of your algorithm incrementally before incorporating them into a larger program.

For more information about the SKILL language, see Getting Started in the *SKILL Language User Guide*.

Using SKILL Code Examples

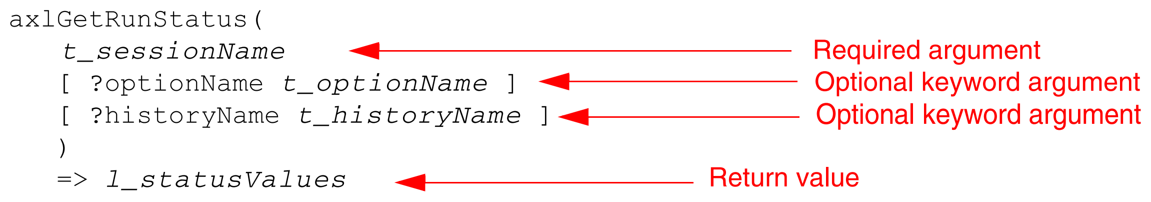
The SKILL APIs in this user manual are explained with illustrative code examples.

You can copy these examples from the manual and paste them directly into the Command Interpreter Window (CIW) or use the code in non-graphical SKILL mode.

Sample SKILL Code

The following code sample shows the syntax of a SKILL API that accepts three arguments.

axlGetRunStatus

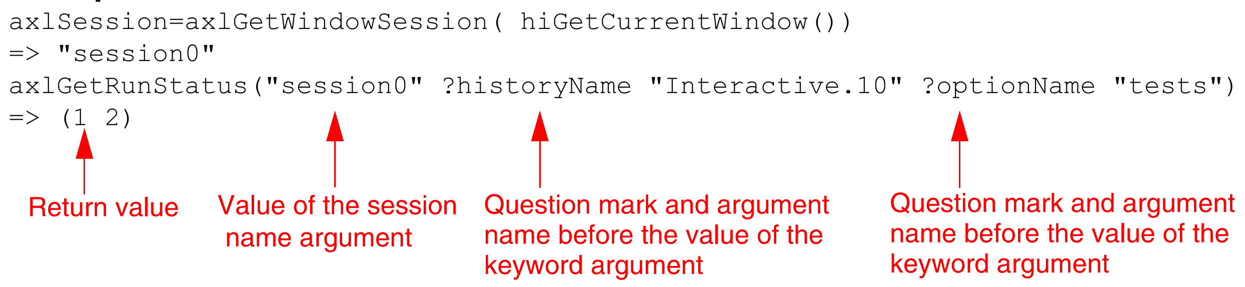


```
axlGetRunStatus(
    t_sessionName
    [ ?optionName t_optionName ]
    [ ?historyName t_historyName ]
    )
    => l_statusValues
```

The first argument `t_sessionName` is a required argument, where `t` signifies the data type of the argument. The second and third arguments `?optionName t_optionName` and `?historyName t_historyName` are optional keyword arguments (identified by a question mark), which are specified in name-value pairs and can be placed in any order during the function call.

The return value is the value that the SKILL API returns after evaluating the expression. In this case, it is a list of status values, *`l_statusValues`*.

Example



```
axlSession=axlGetWindowSession( hiGetCurrentWindow())
=> "session0"
axlGetRunStatus("session0" ?historyName "Interactive.10" ?optionName "tests")
=> (1 2)
```

Return value
Value of the session name argument
Question mark and argument name before the value of the keyword argument
Question mark and argument name before the value of the keyword argument

Accessing API Help

Quick reference information for SKILL APIs is available from the CIW and the SKILL API Finder. To access the reference information for a particular SKILL API, do one of the following:

- Type `help <function_name>` in the CIW.
- Type `startFinder ( [ ?funcName` *`t_functionName`* `] )` in the CIW.
- Start the SKILL API Finder from the CIW by choosing *Tools – Finder* or type `cdsFinder` on the UNIX command line.

  In the *Search in* field of the displayed Cadence SKILL API Finder window, type the SKILL API name for which you want to display the help information and click *Go*.

  The matches for the searched SKILL API appear in the *Results* area.

  To view the complete documentation of the searched SKILL API, select the API name in the *Results* area and click the *More Info* button. The complete documentation of the selected SKILL API appears in a new Cadence Help window.

Typographic and Syntax Conventions

The following typographic and syntax conventions are used in this manual.

text	Indicates names of manuals, menu commands, buttons, and fields.
`text`	Indicates text that you must type as presented. Typically used to denote command, function, routine, or argument names that must be typed literally.
`z_argument`	Indicates text that you must replace with an appropriate argument value. The prefix (in this example, *`z_`*) indicates the data type the argument can accept and must not be typed.
|	Separates a choice of options.
`{ }`	Encloses a list of choices, separated by vertical bars, from which you **must** choose one.
`[ ]`	Encloses an optional argument or a list of choices separated by vertical bars, from which you **may** choose one.
`[ ?argName` *`t_arg`* `]`	Denotes a *key argument*. The question mark and argument name must be typed as they appear in the syntax and must be followed by the required value for that argument.
`...`	Indicates that you can repeat the previous argument. Used with brackets to indicate that you can specify zero or more arguments. Used without brackets to indicate that you must specify at least one argument.
`,...`	Indicates that multiple arguments must be separated by commas.
`=>`	Indicates the values returned by a Cadence® SKILL® language function.
`/`	Separates the values that can be returned by a Cadence SKILL language function.

If a command-line or SKILL expression is too long to fit within the paragraph margins of this document, the remainder of the expression is moved to the next line and indented. In code excerpts, a backslash (\ ) indicates that the current line continues on to the next line.

Identifiers Used to Denote Data Types

Data type identifiers are used to indicate the type of value required by an API argument. These data types are denoted by a single letter that is prefixed to the argument label and is separated from the argument by an underscore; for example, `t` is the data type in *`t_viewName`*. Data types and underscores are used only as identifiers; they must not be typed when specifying the argument in a function.

Prefix	Internal Name	Data Type
a	array	array
A	amsobject	AMS object
b	ddUserType	DDPI object
B	ddCatUserType	DDPI category object
C	opfcontext	OPF context
d	dbobject	Cadence database object (CDBA)
e	envobj	environment
f	flonum	floating-point number
F	opffile	OPF file ID
g	general	any data type
G	gdmSpecIlUserType	generic design management (GDM) spec object
h	hdbobject	hierarchical database configuration object
I	dbgenobject	CDB generator object
K	mapiobject	MAPI object
l	list	linked list
L	tc	Technology file time stamp
m	nmpIlUserType	nmpIl user type
M	cdsEvalObject	cdsEvalObject
n	number	integer or floating-point number
o	userType	user-defined type (other)
p	port	I/O port
q	gdmspecListIlUserType	gdm spec list

Prefix	Internal Name	Data Type
r	defstruct	defstruct
R	rodObj	relative object design (ROD) object
s	symbol	symbol
S	stringSymbol	symbol or character string
t	string	character string (text)
T	txobject	transient object
u	function	function object, either the name of a function (symbol) or a lambda function body (list)
U	funobj	function object
v	hdbpath	hdbpath
w	wtype	window type
sw	swtype	subtype session window
dw	dwtype	subtype dockable window
x	integer	integer number
y	binary	binary function
&	pointer	pointer type

For more information, see *Cadence SKILL Language User Guide*.

1

Customizing EDIF 200 In

You can customize EDIF 200 In output files by creating a SKILL file. A SKILL file contains SKILL procedures that specify changes to object properties in the source design.

This section lists the SKILL procedures you can add to a SKILL file and provides an example of each procedure.

You use the *User-Defined SKILL File* field on the EDIF 200 In form to specify the name of the SKILL file. Specify the full path if it is not in your run directory.

When you start EDIF 200 In, the software automatically looks for the SKILL file in the run directory or in the directory you specify in the *User-Defined SKILL File* field. When the SKILL file is found, the software runs the procedures and modifies the property information that is written to the EDIF 200 In output files.

A SKILL procedure returns a value of `t` when it runs successfully and a value of `nil` when it fails. When a procedure fails, EDIF 200 In processes the property, if possible, as the property is described in the source EDIF file.

See the Cadence SKILL Language User Guide and the Cadence SKILL Language Reference for information about writing SKILL procedures.

edifinDisplay

```
edifinDisplay(
     t_edifFormName
     )
     => t / nil
```

Description

Displays the EDIF 200 In form.

Arguments

`t_edifFormName`	Specifies the Edif200 Import form name.

Value Returned

`t`	Indicates that the command succeeded.
`nil`	Indicates that the command failed.

Example

```
edifinDisplay(transEdifInForm)
```

edifLayerNumMap

```
edifLayerNumMap(
     t_figureGroup
     )
     => list( t_layerName t_purposeName ) / nil
```

Description

Maps the name of `figureGroup` specified in the input EDIF file to the layer-purpose pairs in the technology file.

The technology file is used when libraries are created by EDIF 200 In. The layers defined by `edifLayerNumMap` overwrite the default layers. The procedure returns a list that contains two strings that specify the layer and purpose names.

Arguments

`t_figureGroup`	A `figureGroup` name in the EDIF input file.

Value Returned

`t_layerName`	Returns *`layerName`*, which maps to the specified `figureGroup`.
`t_purposeName`	Returns *`purposeName`*, which maps to the specified `figureGroup`.
`nil`	Returns `nil` if mapping is not possible.

Skeletal Example

```
procedure( edifLayerNumMap( figureGroup "t" )
     prog((flag)
          flag = case( figureGroup
               ("arg1" return( list( "arg2" "arg3" )))
          ) ;end case
          if(!flag then
              return( list( "arg2" "arg3" ))
          );end if
     );prog
);procedure
```

`"arg1"`	Specifies the `figureGroup` name in the input EDIF file.

`"arg2"`	Specifies the DFII layer to which the `figureGroup` is mapped.
`"arg3"`	Specifies the DFII layer-purpose to which the `figureGroup` is mapped.

Example

```
procedure( edifLayerNumMap(figureGroup "t")
     prog((flag)
          flag = case( figureGroup
               ("DEVICE_BODY_FGP" return( list( "device" "drawing" )))
               ("NET_FGP" return( list( "wire" "drawing" )))
               ("PIN_FGP" return( list( "pin" "drawing" )))
               ("PIN_BODY_FGP" return( list( "pin" "drawing" )))
          );end case
          if(!flag then
               return( list( "device" "drawing" ))
          );end if
     );prog
);procedure
```

In this example, any `figureGroup` names that are not among those listed in the `case` statement are mapped to the layer `"device"` with the `"drawing"` purpose. The default layer name for `port` is `"pin"`. The default layer name for `net` is `"wire"`. The default layer name for `net label` is `"wire"` and the `layer-purpose` is `"label"`.

edifinMakeRenameString

```
edifinMakeRenameString(
     t_inputString
     )
     => l_outputString / nil
```

Description

Creates a special string when the string in the EDIF 200 In input file is an illegal name.

The function takes the illegal name string from the EDIF file as an argument and returns a list value that contains the new rename string. You can use `nil` as the `returnString` value. In this case, a `nil` value is returned, which indicates that no new string was provided and the original characters are deleted.

Arguments

`t_inputString`	An illegal name in the EDIF input file.

Value Returned

`l_outputString`	Indicates the list that contains the new rename string.
`nil`	Indicates no new string was provided.

Skeletal Example

```
procedure( edifinMakeRenameString( inputString "t" )
     prog( (returnString)
          case( inputString
               ( "arg1" returnString = "arg2" ))
          return( list( returnString ))
     );prog
);procedure
```

`"arg1"`	Specifies the string in the EDIF file to map.
`"arg2"`	Specifies the translated string in the DFII file.

To map names to a DFII database, you can use any alphanumeric character and any of the following special characters:

_ (underscore)	+ (plus)	- (hyphen)	: (colon)
<> (angle brackets)	{} (braces)	[] (square brackets)	() (parentheses)

If you use an invalid character, EDIF 200 In ignores the name and uses the current EDIF name.

Note: You can use this mapping convention to map the `instanceName`, `portName`, and `netName` constructs. However, when you rename instance names, the `charMapForInstName` function overrides this `edifinMakeRenameString` function.

Example

```
procedure( edifinMakeRenameString( inputString )
     prog( (returnString)
               case( inputString
                    ( "C4" returnString = "C<4>" )
               ); case changes name "C4" into "C<4>"
               return( list( returnString))
     );prog
);procedure
```

The `inputString` is the name of the string in the EDIF file to map. The `returnString` is the name of the DFII database name that is created.

ediFinishStatus

```
ediFinishStatus(
     x_returnCode
     )
```

Description

Provides an exit status number when EDIF 200 In is done.

Arguments

x_returnCode	■ `0`—means there were no errors and no warnings. ■ `-1`—means there were errors and warnings. ■ `1`—means there were warnings but no errors.

Example

```
procedure( ediFinishStatus( returnCode "x")
returnCode = 0 if there are no errors and no warnings
returnCode = -1 if there are errors and warnings
returnCode = 1 is there are warnings and no errors
)
```

2 Customizing EDIF 200 Out

You can customize EDIF 200 Out by creating a SKILL file. A SKILL file contains SKILL procedures that specify changes to object properties in the source design.

This section lists the SKILL procedures you can add to a SKILL file and provides an example of each procedure.

When you start EDIF 200 Out, the software automatically looks for the SKILL file in the run directory or in the directory you specify in the *User-Defined SKILL File* field on the EDIF 200 Out form.

When the SKILL file is found, the software runs the procedures and modifies the property information that is written to the EDIF 200 Out output files.

A SKILL procedure returns a value of `t` when it runs successfully and a value of `nil` when it fails. When a procedure fails, EDIF 200 Out processes the property, if possible, as it is described in the source design.

For information about writing SKILL procedures, see the *Cadence SKILL Language User Guide* and the *Cadence SKILL Language Reference*.

edifDisplay

```
edifDisplay(
     t_edifFormName
     )
     => t / nil
```

Description

Displays the EDIF 200 Out form.

Arguments

t_edifFormName	Specifies the Edif200 Export form name.

Value Returned

`t`	Indicates that the command succeeded.
`nil`	Indicates that the command failed.

Example

```
edifDisplay(transEdifOutForm)
```

edifoutEditProperty

```
edifoutEditProperty(
     d_propId
     d_dbId
     p_outputPort
     )
     => t / nil
```

Description

Customizes user properties.

You can print the property name and value to the EDIF file in any format, or you can skip this property and not print it to the EDIF file.

Arguments

d_propId	Specifies the database property ID.
d_dbId	Specifies the database object ID, which is the ID of the object that contains the property to modify.
p_outputPort	Specifies the port object that is used to access the EDIF 200 Out output file.

Value Returned

`t`	Indicates that the command succeeded.
`nil`	Indicates that the command failed.

Example

```
procedure( edifoutEditProperty( propId dbId outputPort)
     prog( ()
          case( propId~>name
               ("lastSchematicExtraction" return(t))
               ("instancesLastChanged" return(t))
               ("lastModified" return(t))
          ) ; end_case
          return(nil)
     ) ; end_prog
) ; end_proc
```

This routine prevents the properties listed in the case from being written regardless of their context.

edifoutEditLibProperty

```
edifoutEditLibProperty(
     d_propId
     d_dbId
     p_outputPort
     )
     => t / nil
```

Description

Edits properties when EDIF 200 Out is writing `library` constructs.

Arguments

d_propId	Specifies the database property ID.
d_dbId	Specifies the database object ID, which is the ID of the object that contains the property to modify.
p_outputPort	Specifies the port object that is used to access the EDIF 200 Out output file.

Value Returned

`t`	Indicates that the command succeeded.
`nil`	Indicates that the command failed.

Example

```
procedure( edifoutEditLibProperty( propId dbId outputPort)
     prog( ( str )
          if( propId~>valueType == "string" then
               fprintf(outputPort "\n(userData %s \"%s\")"
               propId~>name propId~>value)
               return(t)
          ) ; end_if
          if( propId~>valueType == "int" then
               fprintf(outputPort "\n userData %s %d" propId~>name
               propId~>value)
               return(t)
          ) ; end_if
          return(nil)
     ) ; end_prog
) ; end_proc
```

This procedure writes a `userData` construct for library properties that are either `string` or `integer` types.

edifoutEditCellProperty

```
edifoutEditCellProperty(
     d_propId
     d_dbId
     p_outputPort
     )
     => t / nil
```

Description

Edits properties when EDIF 200 Out is writing `cell` constructs.

Arguments

d_propId	Specifies the database property ID.
d_dbId	Specifies the database object ID, which is the ID of the object that contains the property to modify.
p_outputPort	Specifies the port object that is used to access the EDIF 200 Out output file.

Value Returned

`t`	Indicates that the command succeeded.
`nil`	Indicates that the command failed.

Example

```
procedure( edifoutEditCellProperty( propId dbId outputPort)
     prog( (month day hour min sec year monstr timestr)
          if( propId~>name == "lastModified" then
               sscanf(propId~>value "%s %d %s %d" monstr day
                    timestr year)
               year = year - 1900
               month = getMonth( monstr)
               getTime( timestr hour day sec)
               ; Adjust to UTC (GMT)
               hour = hour + 8
               fprintf( outputPort "\n (status")
               fprintf( outputPort "\n (written")
               fprintf( outputPort
                    "\n (timeStamp %d %d %d %d %d %d )))"
                    year month day hour min sec)
               return(t)
          ) ; end_if
          return(nil)
     ) ; end_prog
) ; end_proc
```

This procedure writes the `lastModified` property of the cell in an EDIF `status` construct. The functions `getMonth` and `getTime` are not shown in this example.

edifoutEditViewProperty

```
edifoutEditViewProperty(
     d_propId
     d_dbId
     p_outputPort
     )
     => t / nil
```

Description

Edits properties when EDIF 200 Out is writing `view` constructs.

Arguments

d_propId	Specifies the database property ID.
d_dbId	Specifies the database object ID, which is the ID of the object that contains the property to modify.
p_outputPort	Specifies the port object that is used to access the EDIF 200 Out output file.

Value Returned

`t`	Indicates that the command succeeded.
`nil`	Indicates that the command failed.

Example

```
procedure( edifoutEditViewProperty( propId dbId outputPort)
     prog( ()
          if(propId~>name == "schType" then
               return(t)
          else
               return(nil)
          ) ; endif
     ) ; end_prog
) ; end_proc
```

This procedure prevents the `schType` property from being written for cellviews.

edifoutEditNetProperty

```
edifoutEditNetProperty(
     d_propId
     d_dbId
     p_outputPort
     )
     => t / nil
```

Description

Edits properties when EDIF 200 Out is writing `net` constructs.

Arguments

d_propId	Specifies the database property ID.
d_dbId	Specifies the database object ID, which is the ID of the object that contains the property to modify.
p_outputPort	Specifies the port object that is used to access the EDIF 200 Out output file.

Value Returned

`t`	Indicates that the command succeeded.
`nil`	Indicates that the command failed.

Example

```
procedure( edifoutEditNetProperty( propId dbId outputPort)
     prog( ()
          if( propId~>name == "criticality" then
               fprintf(outputPort "\n  (criticality %d)"
                    propId~>value)
               return(t)
          ) ; endif
          return(nil)
     ) ; end_prog
) ; end_proc
```

This function generates net criticality.

edifoutEditInstProperty

```
edifoutEditInstProperty(
     d_propId
     d_dbId
     p_outputPort
     )
     => t / nil
```

Description

Edits properties when EDIF 200 Out is writing `instance` constructs.

Arguments

d_propId	Specifies the database property ID.
d_dbId	Specifies the database object ID, which is the ID of the object that contains the property to modify.
p_outputPort	Specifies the port object that is used to access the EDIF 200 Out output file.

Value Returned

t	Indicates that the command succeeded.
nil	Indicates that the command failed.

Example

```
procedure( edifoutEditInstProperty( propId dbId outputPort)
     prog( ()
          if( propId~>name == "connection_expr" then
               fprintf(outputPort
                    "\n (property tap_vector (string %s)
                         (owner \"XYZ\"))" propId~>value)
          return(t)
          ) ; end_if
          return(nil)
     ) ; end_prog
) ; end_proc
```

This function writes out the `connection_expr` property with the name `tap_vector`.

edifoutEditPortProperty

```
edifoutEditPortProperty(
     d_propId
     d_dbId
     p_outputPort
     )
     => t / nil
```

Description

Edits properties when EDIF 200 Out is writing `port` constructs.

Arguments

d_propId	Specifies the database property ID.
d_dbId	Specifies the database object ID, which is the ID of the object that contains the property to modify.
p_outputPort	Specifies the port object that is used to access the EDIF 200 Out output file.

Value Returned

`t`	Indicates that the command succeeded.
`nil`	Indicates that the command failed.

Example

```
procedure( edifoutEditPortProperty( propId dbId port)
     prog( ()
          case( propId~>name
               ("maxFanIn" fprint(outputPort "\n (dcMaxFanIn %d)"
                    propId~>value)return(t))
               ("maxFanOut" fprintf(outputPort "\n (dcMaxFanOut %d)"
                    propId~>value)return(t))
               ("fanInLoad" fprintf(outputPort "\n (dcFanInLoad %d)"
                    propId~>value)return(t))
               ("fanOutLoad" fprintf(outputPort "\n (dcFanOutLoad %d)"
                    propId~>value)return(t))
          ) ; end_case
          return(nil)
     ) ; end_prog
) ; end_proc
```

This function writes the user-defined properties for `port` attributes as EDIF constructs.

edifoutAddNetInfo

```
edifoutAddNetInfo(
     d_anyNetId
     p_outputPort
     )
     => t / nil
```

Description

Adds information about a particular `net` to the EDIF `net` construct.

This procedure writes the string output to the EDIF file.

Arguments

`d_anyNetId`	Specifies the database ID of the net.
`p_outputPort`	Specifies the port object that is used to access the EDIF 200 Out output file.

Value Returned

`t`	Indicates that the command succeeded.
`nil`	Indicates that the command failed.

Example

```
procedure( edifoutAddNetInfo(anyNetId outputPort)
     fprintf(outputPort "\n (property signalRef
                                 (string \"%s\")
                                 (owner \"V210\"))"
          car(anyNetId~>signals~>name)
     ) ; end_fprint
     return(t)
) ; end_proc
```

Adds a property list to the signals of the net.

edifoutAddInstInfo

```
edifoutAddInstInfo(
     d_instId
     p_outputPort
     )
     => t / nil
```

Description

Adds information about a particular instance to the EDIF `instance` construct.

This procedure writes the string output to the EDIF file.

Arguments

`d_instId`	Specifies the database ID of the instance.
`p_outputPort`	Specifies the port object that is used to access the EDIF 200 Out output file.

Value Returned

`t`	Indicates that the command succeeded.
`nil`	Indicates that the command failed.

Example

```
procedure( edifoutAddInstInfo( instId outputPort )
     fprintf(outputPort
          "\n  (property placementStatus (string \"%s\")
          (owner \"Cadence\"))"instId~>status)
     return(t)
) ; end_proc
```

This function adds a property specifying the placement status.

edifoutAddPortInfo

```
edifoutAddPortInfo(
     d_termId
     p_outputPort
     )
     => t / nil
```

Description

Adds information about a terminal (port) to the EDIF `port` construct.

This procedure writes the string output to the EDIF file.

Arguments

`d_termId`	Specifies the database ID of the terminal (port).
`P_outputPort`	Specifies the port object that is used to access the EDIF 200 Out output file.

Value Returned

`t`	Indicates that the command succeeded.
`nil`	Indicates that the command failed.

Example

```
procedure( edifoutAddPortInfo(id outputPort)
     prog(( numstr )
          foreach(x tempId~>prop
               case( x~>name
                    ("maxFanIn" fprintf(outputPort "\n (dcMaxFanIn %d)"
                         x~>value))
                    ("maxFanOut" fprintf(outputPort "\n (dcMaxFanOut
                         %d)"x~>value))
                    ("fanInLoad" fprintf(outputPort "\n (dcFanInLoad
                         %d)"x~>value))
                    ("fanOutLoad" fprintf(outputPort "\n (dcFanOutLoad
                         %d)"x~>value))
                    ("TPHL" numstr = makeEDIFNumber(x~>value)
                         fprintf(outputPort
                              "\n  property t_PHL number %s) (owner
                              \"IEC_TC3\")
                         (comment \"I should be an EDIF attribute\"))"
                              numstr) )
                    ("TPLH" numstr = makeEDIFNumber(x~>value)
                        fprintf(outputPort
                             "\n (property t_PLH (number %s) (owner
                             "\IEC_TC3\")
                        (comment \"I should be an EDIF attribute\"))"
                             numstr) )
                  ) ;end_case
             ) ;end_foreach
             return(t)
        ) ;end_prog
) ;end_proc
```

Another way to add simulation data. Uses function `makeEDIFNumber`, which returns a string of the form *`(e m x)`*.

edifoutAddInterfaceInfo

```
edifoutAddInterfaceInfo(
     d_cellViewId
     p_outputPort
     )
     => t / nil
```

Description

Adds information about a cellview to the EDIF `interface` construct.

This procedure writes the string output to the EDIF file.

Arguments

`d_cellViewId`	Specifies the database ID of the cellview.
`p_outputPort`	Specifies the port object that is used to access the EDIF 200 Out output file.

Value Returned

`t`	Indicates that the command succeeded.
`nil`	Indicates that the command failed.

Example

```
procedure( edifoutAddInterfaceInfo(CVId outputPort)
     prog((min nom max propname pname1 pname2)
          foreach(x CVId~>prop
               index = -1 * strlen(x~>name)
               propname=substring(x~>name index 6)
               case( propname
                    ("tPCKQ_"
                         getNames(x~>name pname1 pname2)
                         fprintf(outputPort "\n (timing")
                         fprintf(outputPort "\n  (derivation
                              REQUIRED)\n (pathDelay")
                         min = 1
                         nom = 2
                         max = 3
                         fprintf(outputPort "\n  (delay mnm %s %s %s)"
                              min nom max)
                         fprintf(outputPort
                              "\n   (event (portRef %s) (transition L
                                   H))" pname1)
                         fprintf(outputPort
                              "\n   (event (portRef %s) (transition L
                              H))))" pname2))
               ) ; end_case
          ) ; end_foreach
     ) ; end_prog
) ; end_proc
```

This function adds the property delay from *`CLK`* to *`Q`* using the range property *`PCKQ_A_B`* where *`A`* and *`B`* are the `portNames` to which the property applies. The function `getNames` is not shown.

edifoutAddViewInfo

```
edifoutAddViewInfo(
     d_cellViewId
     p_outputPort
     )
     => t / nil
```

Description

Adds information about a cellview to the EDIF `view` construct.

This procedure writes the string output to the EDIF file.

Arguments

`d_cellViewId`	Specifies the database ID of the cellview.
`p_outputPort`	Specifies the port object that is used to access the EDIF 200 Out output file.

Value Returned

`t`	Indicates that the command succeeded.
`nil`	Indicates that the command failed.

Example

```
procedure( edifoutAddViewInfo(cvId outputPort)
     fprintf(outputPort "\n  (property fileName (string \"%s\")
          (owner \"Jedi\"))"cvId~>fileName)
     return(t)
) ; end_proc
```

This routine prints the filename for the `cellView` in the property `fileName`.

edifoutAddCellInfo

```
edifoutAddCellInfo(
     d_cellViewId
     p_outputPort
     )
     => t / nil
```

Description

Adds information about a cellview to the EDIF `cell` construct.

This procedure writes the string output to the EDIF file.

Arguments

`d_cellViewId`	Specifies the database ID of the cellview.
`p_outputPort`	Specifies the port object that is used to access the EDIF 200 Out output file.

Value Returned

`t`	Indicates that the command succeeded.
`nil`	Indicates that the command failed.

Example

```
procedure( edifoutAddCellInfo( cellId outputPort)
     fprintf(outputPort "\n  (userData cellOwner \"%s\")"
          cellId~>owner)
     return(t)
) ;end_proc
```

This function prints the owner of the cell as a `userData`.

edifoutAddLibraryInfo

```
edifoutAddLibraryInfo(
     d_libId
     p_outputPort
     )
     => t / nil
```

Description

Adds information about a library to the EDIF `library` construct.

Arguments

`d_libId`	Specifies the database ID of the library.
`p_outputPort`	Specifies the SKILL port that corresponds to the EDIF 200 Out output file.

Value Returned

`t`	Indicates that the command succeeded.
`nil`	Indicates that the command failed.

Example

```
procedure( edifoutAddLibraryInfo(libId outputPort)
     prog( ()
          if(libId~>name == "MOTO_LIB" then
               fprintf(outputPort
                    "\n  (userData GLOBAL_FANOUT
                         \"LMACRO= 14,6,10,-,22\")")
               return(t)
          ) ; end_if
          return(nil)
     ) ; end_prog
) ; end_proc
```

This function adds a `userData` to a library.

edifoutMakeRenameString

```
edifoutMakeRenameString(
     d_id
     p_outputPort
     )
     => t_result / nil
```

Description

Generates a string to use for renaming an object in the EDIF file.

Characters that are illegal in EDIF are mapped to the underscore character (_). Array names are mapped to the string *`n_TO_m`* where *`n`* and *`m`* are the lower and upper bounds of the array indexes, respectively.

Arguments

`d_id`	Specifies the database object ID.
`p_outputPort`	Specifies the SKILL port that corresponds to the EDIF 200 output file.

Value Returned

`t_result`	The string name.
`nil`	Indicates that the command failed.

Example

```
procedure( edifoutMakeRenameString(dbId outputPort)
     return(dbId~>baseName)
) ; end_proc
```

This function returns the base name of the object as the *`rename`* string instead of using the normal EDIF Out mappings.