cādence®

Virtuoso® AMS Designer Environment SKILL Reference

Product Version 6.1.6
August 2014

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.

Scope of this Manual

The SKILL functions described in this manual can be used in either IC6.1.6, ICADV12.1, or both of these releases. Functions that are supported only in a particular release are identified using the **(ICADV12.1 ONLY)** or **(IC6.1.6 ONLY)** text at the beginning of the function description. All other functions are supported in both releases.

Important

Only the functions and arguments described in this manual are supported for public use. All other functions, and undocumented aspects of the functions described here, are private and subject to change at any time.

Related Documents for AMS Designer Environment SKILL Functions

The SKILL programming language is often used with other Virtuoso products or requires knowledge of a special language. The following documents give you more information about these tools and languages.

- If you want to use the SKILL language functions, the Virtuoso SKILL++™ functions, and the SKILL++ object system (for object-oriented programming), you need to read the *Cadence SKILL Language User Guide*.
- If you want to see descriptions, syntax, and examples for the SKILL and SKILL++ functions, you need to read the *Cadence SKILL Language Reference*.
- If you want to see descriptions, syntax, and examples for the object system functions, you need to read the *Cadence SKILL++ Object System Reference*.

- *Virtuoso Design Environment SKILL Functions Reference* provides detailed information about the SKILL functions that interface to applications in the Virtuoso Design Environment.
- If you want to design and simulate AMS Designer environment, you need to read the *VirtuosoAMS Designer Environment User Guide*.

Typographic and Syntax Conventions

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

`text`	Indicates text you must type exactly as it is presented.
`z_argument`	Indicates text that you must replace with an appropriate argument. The prefix (in this case, *`z_`*) indicates the data type the argument can accept. Do not type the data type or underscore.
`[ ]`	Denotes an optional argument. When used with vertical bars, they enclose a list of choices from which you can choose one.
`{ }`	Used with vertical bars, they denote a list of choices from which you must choose one.
|	Separates a choice of options.
...	Indicates that you can repeat the previous argument.
=>	Precedes the values returned by a Cadence® SKILL language function.
`/`	Separates the possible values that can be returned by a Cadence SKILL language function.
text	Indicates names of manuals, menu commands, form buttons, and form fields.

Identifiers Used to Denote Data Types

The Cadence SKILL language supports different data types to identify the type of value you can assign to an argument.

Data types are identified by a single letter followed by an underscore; for example, `t` is the data type in *`t_viewNames`* and denotes that the argument in question accepts a character string. Data types and the underscore are used as identifiers only; they should not be typed.

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	gdm spec
h	hdbobject	hierarchical database configuration object
K	mapiobject	MAPI object
l	list	linked list
L	tc	Technology file time stamp
m	nmpIlUserType	nmpIl user type
M	cdsEvalObject	—
n	number	integer or floating-point number
o	userType	user-defined type (other)
p	port	I/O port
q	gdmspecListIlUserType	gdm spec list
r	defstruct	defstruct
R	rodObj	relative object design (ROD) object
s	symbol	symbol
S	stringSymbol	symbol or character string

Prefix	Internal Name	Data Type
`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	—
`w`	wtype	window type
`x`	integer	integer number
`y`	binary	binary function
`&`	pointer	pointer type

Scope of This Manual

This manual may contain a combination of SKILL functions that are appropriate for use in either IC6.1.6, ICADV12.1, or both of these releases.

By default, any function’s usage should be considered as being applicable to both IC6.1.6 and ICADV12.1. Where a function’s usage is applicable to only one of these releases it will be indicated as such in the abstract paragraph of that function. For example, the function will be marked as being applicable to “(ICADV12.1 ONLY)” or “(IC6.1.6 ONLY”).

Additional Learning Resources

Cadence provides various Rapid Adoption Kits that you can use to learn how to employ Virtuoso applications in your design flows. These kits contain workshop databases, designs, and instructions to run the design flow.

Cadence offers the following training courses on the SKILL programming language, which you can use to customize, extend, and automate your design environment:

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

For further information on the training courses available in your region, visit the Cadence Training portal. You can also write to training_enroll@cadence.com.

Note: The links in this section open in a new browser. They initially display the requested training information for North America, but if required, you can navigate to the courses available in other regions.

1

AMS Designer SKILL Functions

The following table lists the public SKILL functions associated with the AMS Designer environment. See the cross-references for syntax, descriptions, and examples.

SKILL Function	For information, see
`amsCheckCV`	amsCheckCV on page 12
`amsIsPresent`	amsIsPresent on page 14
`amsNetlist`	amsNetlist on page 15
`amsProcessCellViews`	amsProcessCellViews on page 18
`amsUpdateTextviews`	amsUpdateTextviews on page 21
`amsUIOptionsForm`	amsUIOptionsForm on page 23
`amsUIRunNetlisterForm`	amsUIRunNetlisterForm on page 24
`ddsCvtAMSTranslateCell`	ddsCvtAMSTranslateCell on page 25
`ddsCvtAMSTranslateLib`	ddsCvtAMSTranslateLib on page 28
`ddsCvtToolBoxAMS`	ddsCvtToolBoxAMS on page 30
`vmsUpdateCellViews`	vmsUpdateCellViews on page 31

amsCheckCV

```
amsCheckCV(
     d_cvId g_genNetlist
     [ s_markerFuncSym ]
     )
     => l_numCount
```

Description

Runs AMS checks on the given cellview. The exact nature of checks and severity of violations is set by AMS Designer environment variables. This function checks the cellview only if the `amsDirect.vlog checkOnly` environment variable is set to `t`.

Arguments

d_cvId	The cellview to run AMS checks upon.
g_genNetlist	If `t`, specifies that a netlist is to be generated.
s_markerFuncSym	If not nil, attaches markers to database objects that violate AMS checks. The syntax of the marker function is **markerFunc(** *d_id t_severity t_text***)** where *d_id* is the database ID of the offending object, *t_severity* is either `"error"` or `"warning"`, and *t_text* is a string containing the text of the error.

Value Returned

l_numCount	A list of two integers: the number of errors, and the number of warnings encountered while running AMS checks.

Example

To run AMS checks and netlist a previously opened cellview, you might use

```
amsCheckCV( cv t )
```

The number of errors and warnings is returned as a list, and a `verilog.vams` netlist file is also generated for the cellview.

To run AMS checks on a previously opened cellview and enable the markers,

amsIsPresent

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

Description

Determines whether AMS netlisting capability is included as part of an executable.

Arguments

None.

Value Returned

`t`	AMS netlisting capability is included in the executable.
`nil`	AMS netlisting capability is *not* included in the executable.

Example

You can test for the presence of the AMS netlisting capability like this:

```
if( isCallable( ’amsIsPresent )
then
;; Yes, AMS Netlisting capability is included
...
else
;; No, AMS Netlisting capability is not present
...
)
```

amsNetlist

```
amsNetlist(
     t_libName
     [t_cellName]
     [t_viewName]
     [ ?checkOnly g_checkOnly ]
     [ ?netlist g_netlist ]
     [ ?netlistMode s_netlistMode ]
     [ ?compile g_compile ]
     )
     => t/nil
```

Description

Runs the AMS netlister on the specified cellviews and, depending upon the passed arguments, performs one or more of the following operations: 1) checks cellviews; 2) checks and netlists cellviews; 3) checks, netlists and compiles cellviews; 4) compiles cellviews.

To generate a netlist, the `amsNetlist` function calls the following netlist procedures, in the order given.

1. `amsPrintComments`
2. `amsPrintHeaders`
3. `amsPrintModule`
4. `amsPrintFooters`

You cannot override the `amsNetlist` function, so you cannot change the order in which the procedures are called. You can, however, override the individual procedures.

Arguments

t_libName	A string, which is the name of the library to process.
t_cellName	A string, which is the name of the cell to process. If *t_cellName* is left blank (with just `""`), all the cells in the library are processed.
t_viewName	A string, which is the name of the view to process. If *t_viewName* is left blank (with just `""`), all the views are processed.

g_checkOnly	The value `t` or `nil`. If `t` is specified, the checks run. If `nil` is specified, the checks do not run. If no value is specified, the value defaults to that of the `amsDirect.vlog checkOnly` environment variable.
g_netlist	The value `t` or `nil`. If `t` is specified, a Verilog-AMS netlist is generated. If *g_netlist* is `nil`, no netlist is generated. If no value is specified, the value defaults to that of the `amsDirect.vlog checkAndNetlist` environment variable.
s_netlistMode	A symbol with the value `'incr` or `'all`. If `'incr` is specified, only new or revised cellviews are netlisted. For example, changing a symbol or the CDF for a device on a Schematic and then requesting netlisting triggers netlisting for only affected cells When `'all` is specified and netlisting is requested, every cell is netlisted. This is the default value.
g_compile	The value `t` or `nil`. If `t` is specified, the generated Verilog-AMS netlist is compiled. If *g_compile* is `nil`, the netlist is not compiled. If no value is specified, the default value depends on the value of the `amsDirect.vlog prohibitCompile` environment variable. When the value of the `prohibitCompile` variable is `t`, the default value for *g_compile* is `nil`. When the value of the `prohibitCompile` variable is `nil`, the default value for *g_compile* is `t`.

Value Returned

`t`	The function was successful.
`nil`	The function failed.

Example

To netlist and compile `mylib.mycell:schematic`:

```
amsNetlist( "mylib" "mycell" "schematic" ?netlist t ?compile t)
```

To netlist and compile all eligible views of `mycell`:

```
amsNetlist( "mylib" "mycell" "" ?netlist t ?compile t)
```

To compile all the cellviews in `mylib`:

```
amsNetlist( "mylib" "" "" ?compileAll t)
```

amsProcessCellViews

```
amsProcessCellViews(
     t_libName
     [t_cellName]
     [t_viewName]
     [ ?checkOnly g_checkOnly ]
     [ ?netlist g_netlist ]
     [ ?compile g_compile ]
     [ ?netlistNode s_netlistMode ]
     [ ?compileMode s_compileMode ]
     )
     => t/nil
```

Description

Performs, depending upon the passed arguments, one or more of the following operations: 1) checks cellviews; 2) checks and netlists cellviews; 3) checks, netlists, and compiles netlisted cellviews; 4) compiles Verilog-AMS, Verilog (digital), Verilog-A, VHDL (digital), and VHDL-AMS files in cellviews.

Arguments

t_libName	A string, which is the name of the library to process.
t_cellName	A string, which is the name of the cell to process. If *t_cellName* is left blank (with just `""`), all the cells in the library are processed.
t_viewName	A string, which is the name of the view to process. If *t_viewName* is left blank (with just `""`), all the views are processed.
g_checkOnly	The value `t` or `nil`. If `t` is specified, the checks run. If `nil` is specified, the checks do not run. If no value is specified, the value defaults to that of the `amsDirect.vlog checkOnly` environment variable.
g_netlist	The value `t` or `nil`. If `t` is specified, a Verilog-AMS netlist is generated. If `nil` is specified, no netlist is generated. If no value is specified, the value defaults to that of the `amsDirect.vlog checkAndNetlist` environment variable.

g_compile	The value `t` or `nil`. If `t` is specified, the generated Verilog-AMS netlist is compiled. If `nil` is specified, the netlist is not compiled. If no value is specified, the default value depends on the values of the `amsDirect.vlog prohibitCompile` and the `amsDirect.vhdl prohibitCompile` variables, as shown in the following table. When the value of both `amsDirect.vlog prohibitCompile` and `amsDirect.vhdl prohibitCompile` are set to `t`, the default value for *g_compile* is `nil`. When the value of one or both of `amsDirect.vlog prohibitCompile` and `amsDirect.vhdl prohibitCompile` are set to `nil,` the default value for *g_compile* is `t`.
s_netlistMode	A symbol with the value `'incr` or `'all`. If `'incr` is specified, only new or revised cellviews are netlisted. For example, changing a symbol or the CDF for a device on a Schematic and then requesting netlisting triggers netlisting for only affected cells. When `'all` is specified and netlisting is requested, every cell is netlisted. This is the default value.
s_compileMode	A symbol with the value `'whenNetlist` or `'all`. The `'whenNetlist` value specifies that only cellviews that are netlisted are compiled. The `'whenNetlist` value is the default. The `'all` value specifies that all cellviews are compiled, whether newly netlisted or not.

Value Returned

`t`	The function was successful.
`nil`	The function failed.

Examples

To netlist and compile all eligible views of `mycell`:

```
amsProcessCellViews( "mylib" "mycell" "" ?netlist t ?compile t)
```

To compile all the cellviews in `mylib` without netlisting:

```
amsProcessCellViews( "mylib" "" "" ?netlist nil ?compile t ?compileMode 'all )
```

To netlist and compile all the cellviews in `mylib`:

```
amsProcessCellViews( "mylib" "" "" ?netlist t ?compile t ?compileMode ‘all )
```

amsUpdateTextviews

```
amsUpdateTextviews(
     t_libName
     [t_cellName]
     [t_viewName]
     [?incremental g_incremental]
     )
     => t/nil
```

Description

Creates a Virtuoso database, depending on the passed arguments, for the following:

- for all the text views in the library
- for all the text views of the specified cell in the library
- for all the text views in the library that have the specified view name
- for the specified text view, given that the view is a text view
- for all the text views in the configuration, given that the specified view is a config view
- for all the text views that do not have an existing Virtuoso database or have a database with an older timestamp than the specified text view

Arguments

t_libName	A string, which is the name of the library to process.
t_cellName	A string, which is the name of the cell to process.
t_viewName	A string, which is the name of the view to process.
g_incremental	If `t`, the Virtuoso database is created only for text views that do not have an existing Virtuoso database or have a database with an older timestamp than the text view.If `nil`, the database is created for all the text views. The default value is `t`.

Value Returned

`t`	The function was successful in creating the Virtuoso database.
`nil`	The function failed in creating the Virtuoso database.

Notes

If you have text views in read-only libraries, you must set an environment variable in `.cdsinit` file. It specifies the directory where the Virtuoso database for such text views must be created. You should set the environment variable as shown below:

```
envSetVal("ams.netlisterOpts" "amsTempDirForShadows" 'string’"<pathToDirectory>")
```

Automatic creation of the Virtuoso database for read-only Verilog-A and VHDL-AMS text views is not supported.

Examples

To create a Virtuoso database for all the text views in `myLib`:

```
amsUpdateTextviews("myLib")
```

To create a Virtuoso database for all the text views of `mycell` in `mylib`:

```
amsUpdateTextviews("myLib" ?cellName "mycell" )
```

To create a Virtuoso databse for all the text views in the `config` view of `mycell` in `mylib`:

```
amsUpdateTextviews("myLib" ?cellName "mycell" ?viewName "config" ?incremental nil
)
```

amsUIOptionsForm

```
amsUIOptionsForm(
      )
```

Description

Pops up the AMS Options form, which is used to set environment variables.

Arguments

None.

Value Returned

None.

Example

```
amsUIOptionsForm()
```

amsUIRunNetlisterForm

```
amsUIRunNetlisterForm(
      )
```

Description

Pops up the AMS Netlister form, which is used to run the AMS Netlister on specified cellviews.

Arguments

None.

ddsCvtAMSTranslateCell

```
ddsCvtAMSTranslateCell(
     b_cellId
     g_overwriteAMS
     l_viewList
     [ ?setPrimitive g_setPrimitive ]
     )
```

Description

Given a DDPI cell ID, this function translates any existing Spectre simulation information for the cell to AMS simulation information. In the process, the `otherParameters`, `instParameters`, `termOrder`, `componentName`, and `propMapping` fields of the Spectre simulation information are copied to the AMS simulation information.

In addition, this function might create the following new fields:

- `stringParameters`
- `referenceParameters`
- `enumParameters`
- `arrayParameters`
- `extraTerminals`

This function categorizes parameters into `stringParameters`, `referenceParameters`, `enumParameters`, `arrayParameters`, and `extraTerminals` by examining the `netlistProcedure` listed in the Spectre simulation information of the cell.

This function can also set the *isPrimitive* field in the AMS simulation information.

Arguments

`b_cellId`	The cell ID obtained using DDPI.
`g_overwriteAMS`	If `t`, existing AMS simulation is overwritten. If `nil`, existing AMS simulation information is not modified.
`l_viewList`	A list of view names, for example, `'("spectre")`. Spectre simulation information is translated to AMS simulation information only if the cell has at least one view from this list.

`g_setPrimitive`	`nil`: Do not set *isPrimitive*. `'model`: set *isPrimitive* if `model*` exists in AMS simulation information..`'spectreView`: set *isPrimitive* if cell has a spectre view.`'modelAndSpectreView`: set *isPrimitive* if cell has a spectre view and `model*` exists in the AMS simulation information

Notes

Converting simulation information usually requires editing the AMS simulation information. This function does not fill in the `isPrimitive` field of the AMS simulation information. Editing is definitely required if the `netlistProcedure` specified in the Spectre section of the simulation information is not one of the following:

- `spectreCCPrim`
- `spectreFsrcPrim`
- `spectreMindPrim`
- `spectreNportPrim`
- `spectrePolyCntrlPrim`
- `spectrePortPrim`
- `spectrePortSrcPrim`
- `spectrePwlsrcPrim`
- `spectreSCCPrim`
- `spectreSVCPrim`
- `spectreSrcPrim`
- `spectreVandISourcePrim`
- `spectreWindingPrim`

See Updating Legacy SimInfo for Analog Primitives for more details about AMS simulation information.

Examples

To convert the Spectre simulation information of `mylib.mycell` to AMS simulation information:

```
cellId = ddGetObj( "mylib" "mycell" )
ddsCvtAMSTranslateCell( cellId nil nil )
```

ddsCvtAMSTranslateLib

```
ddsCvtAMSTranslateLib(
      t_libName
      g_overwriteAMS
      t_fileName
      l_viewList
      [ ?setPrimitive g_setPrimitive ]
      )
      => t/nil
```

Description

Translates any existing Spectre simulation information for all the cells in *`t_libName`* to AMS simulation information. In the process, the `otherParameters`, `instParameters`, `termOrder`, `componentName`, and `propMapping` fields of the Spectre simulation information are copied to the AMS simulation information.

In addition, this function might create the following new fields:

- `stringParameters`
- `referenceParameters`
- `enumParameters`
- `arrayParameters`
- `extraTerminals`

This function categorizes parameters into `stringParameters`, `referenceParameters`, `enumParameters`, `arrayParameters`, and `extraTerminals` by examining the `netlistProcedure` listed in the Spectre simulation information of each cell.

Arguments

`t_libName`	The library name.
`g_overwriteAMS`	If `t`, existing AMS simulation is overwritten. If `nil`, existing AMS simulation information is not modified.
`t_fileName`	Existing simulation information for the cells of the library are written to this file. To restore, type `load("`*`t_fileName`*`")`in the CIW.

`l_viewList`	A list of view names, for example, `'("spectre")`. Spectre simulation information is translated to AMS simulation information only for cells that have at least one view from this list.
`g_setPrimitive`	`nil`: Do not set *isPrimitive*. `'model`: set *isPrimitive* if `model*` exists in AMS simulation information..`'spectreView`: set *isPrimitive* if cell has a spectre view.`'modelAndSpectreView`: set *isPrimitive* if cell has a spectre view and `model*` exists in the AMS simulation information

Examples

The next example converts Spectre simulation information to AMS simulation information for all the cells in `mylib` that have a Spectre view, without modifying existing AMS simulation information.

```
ddsCvtAMSTranslateLib( "mylib" nil "/tmp/old_siminfo" '("spectre") )
```

ddsCvtToolBoxAMS

```
ddsCvtToolBoxAMS(
      )
```

Description

Pops up the Create AMS from Spectre form, which can be used to translate Spectre simulation information for cells in a library to AMS simulation information.

The actual conversion is done using the `ddsCvtAMSTranslateLib` function.

Arguments

None.

Examples

```
ddsCvtToolBoxAMS()
```

vmsUpdateCellViews

```
vmsUpdateCellViews(
     [ ?lib lt_lib ]
     [ ?cell lt_cell ]
     [ ?view lt_view ]
     [ ?viewt t_viewType ]
     )
     => t/nil
```

Description

Updates AMS Designer information with the current state of verilog, veriloga, verilogams and vhdl text views. You might use this function, for example, when you have updated a Verilog-AMS source file outside of the AMS Designer environment. You might also use it when you receive a Verilog-AMS library in a single source file, bring it into the Library.Cell:View structure using `ncvlog -use5x`, and then need to prepare the library for use in the AMS Designer environment. See also customization variable `vmsDoNotCheckMasterFileWritable`.

Note: If you run this function without any arguments, a pop-up appears asking for lib/cell/view and viewType information.

Arguments

lt_lib	A string, which is the name of a library or a list of library names to look in for cellviews to update. If this argument is not specified (with just `""`) or is specified as `nil`, all libraries defined in the `cds.lib` file are searched.
lt_cell	A string, which is the name of a cell or a list of cell names to be searched for update in the libraries. If this argument is not specified (with just `""`) or is specified as `nil`, all cells are searched.
lt_view	A string, which is the name of a cellview or a list of cellview names to be searched for update. If this argument is not specified (with just `""`) or is specified as `nil`, all views are searched.
t_viewType	The type of view that you want to update. Valid Values:

`text.ahdl`	Analog HDL text view
`text.veriloga`	

	Verilog-A text view
`VHDLAMSText`	VHDL-AMS text view
`vhdl`	VHDL text view
`text.v`	Verilog text view
`VerilogAMSText`	Verilog-AMS text view

Value Returned

`t`	The function ran successfully.
`nil`	The function failed.

Examples

This example updates the specified text cellview.

```
vmsUpdateCellViews(?lib "myLib" ?cell "myCell" ?view "verilogAMS"
?viewt "VerilogAMSText" )
```

The next example updates verilogAMS views in all the cells in the `myLib` library.

```
vmsUpdateCellViews(?lib "myLib" ?view "verilogAMS" ?viewt "VerilogAMSText" )
```

2

SKILL Functions Supported for Netlisting

The SKILL functions specifically developed for use in custom netlisting procedures can be divided into those that replicate the default netlisting behavior and those that perform lower-level helping functions. These two varieties are listed in the following tables and are described in detail in the remainder of this appendix.

The default netlisting procedures reproduce the default behavior of the AMS netlister. For example, if you want the netlister to print the default headers in the default format, leave the `headersProc` field of the formatter object set to the default netlisting procedure, `amsPrintHeaders`.

The default netlisting procedures take into account information that might be required to create a netlist. For that reason, customized netlist procedures often run default netlisting procedures after setting up the appropriate data.

Table 2-1 Default Netlisting Procedures

Procedure	For more information, see...
`amsPrintAliases`	amsPrintAliases on page 58
`amsPrintAttributes`	amsPrintAttributes on page 62
`amsPrintInstance`	amsPrintInstance on page 64
`amsPrintInstanceMasterName`	amsPrintInstanceMasterName on page 66
`amsPrintInstanceParameters`	amsPrintInstanceParameters on page 70
`amsPrintInstancePorts`	amsPrintInstancePorts on page 72
`amsPrintIOs`	amsPrintIOs on page 75
`amsPrintParameters`	amsPrintParameters on page 77
`amsPrintPorts`	amsPrintPorts on page 81
`amsPrintWires`	amsPrintWires on page 85

The netlisting helper functions provide specific behaviors that you can combine to create the overall custom behavior that you need.

Table 2-2 Netlisting Helper Functions

Function	For more information, see...
`amsError`	amsError on page 35
`amsGetInstanceName`	amsGetInstanceName on page 37
`amsGetNetlister`	amsGetNetlister on page 39
`amsGetPortExpr`	amsGetPortExpr on page 41
`amsGetUniqueName`	amsGetUniqueName on page 43
`amsInfo`	amsInfo on page 44
`amsMapName`	amsMapName on page 46
`amsMtlinePrintParams`	amsMtlinePrintParams on page 48
`amsMtlineTermOrder`	amsMtlineTermOrder on page 50
`amsNportTermOrder`	amsNportTermOrder on page 51
`amsPrint`	amsPrint on page 52
`amsPrintAlias`	amsPrintAlias on page 56
`amsPrintAttribute`	amsPrintAttribute on page 60
`amsPrintInstanceParameter`	amsPrintInstanceParameter on page 68
`amsPrintIO`	amsPrintIO on page 74
`amsPrintParameter`	amsPrintParameter on page 76
`amsPrintPort`	amsPrintPort on page 79
`amsPrintWire`	amsPrintWire on page 83
`amsSpectreToVams`	amsSpectreToVams on page 87
`amsWarning`	amsWarning on page 88

amsError

```
amsError(
     A_formatterId
     t_msg
     )
     => t/nil
```

Description

Helper function that prints *t_msg* in the form of an error message and increments the error count. The message is added to the log file. Calling this function causes netlisting to fail, although processing to detect further netlisting problems continues.

Arguments

A_formatterId	ID of the formatter object.
t_msg	Message to be printed. If you want newline characters to appear in the message, you must include them in the message.

Value Returned

`t`	String was printed.
`nil`	String was not printed.

Example

You enter the following code in your netlist procedures override file. The code defines an instance parameter netlist procedure that includes the `amsError` function.

```
;; Function that checks a custom structure. The structure can have
;; n fingers, where n must be between 1 and 10. This function only checks--
;; the parameter is actually printed by amsPrintInstanceParameters.
(defun MyFingersProc (formatter cellview instance)
   (let (fingers)
      (setq fingers instance->id->numFingers)
      (when (or (lessp fingers 1)
                (greaterp fingers 10)
                ) ; or
           (amsError formatter
                (sprintf nil
                   "Number of fingers (%d) must be between 1 and 10 (%s)\n"
                    fingers instance->name
                    ) ; sprintf
                ) ; amsError
          ) ; when
      ;; Just print the parameters
      (amsPrintInstanceParameters formatter cellview instance )
      ) ; let
   ) ; defun
```

If the number of fingers is outside the range, this function generates error messages like the following ones:

```
Error: Number of fingers (12) must be between 1 and 10 (I2)
Error: Number of fingers (-2) must be between 1 and 10 (I1)
```

amsGetInstanceName

```
amsGetInstanceName(
     A_formatterId
     A_instanceId
     [x_iteration]
     )
     => t_instanceName/nil
```

Description

Helper function that returns the print name of the instance specified by *`A_instanceId`* or the print name of the specified iteration of the instance.

Arguments

`A_formatterId`	ID of the formatter object.
`A_instanceId`	ID of the instance object.
`x_iteration`	Iteration number specifying a particular iterated instance.

Value Returned

`t_instanceName`	Name of the specified instance.
`nil`	Name was not retrieved.

Example

This netlisting procedure uses the `amsGetInstanceName` function to retrieve the print name of the instance so that it can be used by the `amsPrintInstanceParameter` function and written to the netlist.

```
;; Customize the parameter "r" for resistor to be 4K always.
(defun MYPrintInstanceParameters (formatterId cvId instanceId)

  (if (instanceId->masterName == "resistor") then
         (amsPrint formatterId "#(")

         ;; Go through the list of parameters for resistor
         (foreach param instanceId->parameters

            (unless (equal param (car instanceId->parameters))
              (amsPrint formatterId ",")
            ); unless
            ;; Print only the parameter called "r"
            (if (param->name == "r") then

              ;; Compute my_new_value
              my_new_value = "4K"

              ;; Set the value of "r" to new value
              param->value = my_new_value

              instanceName = (amsGetInstanceName formatterId instanceId)

              ;; Call the helper function to print the parameter
              amsPrintInstanceParameter(formatterId instanceName param)
            )

         );foreach

         (amsPrint formatterId ")")
  ); if

  ;; For any instance whose masterName is NOT "resistor", print its
  ;; parameters in the default way using the default print function.
  ;;
  (if (instanceId->masterName != "resistor") then
       amsPrintInstanceParameters(formatterId cvId instanceId)
  )

); defun
```

amsGetNetlister

```
amsGetNetlister(
     )
     => A_netlisterId/nil
```

Description

Returns the ID of the top-level netlister object. This netlister object contains the global options applicable to the AMS netlister. The object is available throughout the life of the UNIX process that is running the AMS netlister and is a unique object for that process.

For information about the netlister object, see Netlister Object.

Arguments

None.

Value Returned

`A_netlisterId`	The ID of the current netlister.
`nil`	The ID was not obtained.

Example 1

You enter the following code in your netlist procedures override file. The code uses the `amsGetNetlister` function to obtain the information necessary to implement other functions.

```
netlisterId = amsGetNetlister()

;; Get the Verilog-AMS formatterId.
vlogFormId = netlisterId->vlog

;; Override the default comment printing function.
vlogFormId->commentsProc = 'MyCommentsProc
```

Example 2

You enter the following into the CIW.

```
netlisterID=amsGetNetlister()
```

AMS Designer returns the ID, in a format similar to

```
ams:28164120
```

You list the information and values contained in the netlister object by typing the following command in the CIW.

```
netlisterID->??
```

AMS Designer returns a list of settings, in a format similar to

```
(lsbMsb nil scalarizeInstances t includeInstCDFParams
    nil excludeParams nil expScalingFactor no
    modifyParamScope no vlog ams:28164140 vhdl
    nil
)
```

amsGetPortExpr

```
amsGetPortExpr( A_formatterId A_portId [x_iteration] )
     => t_portExpr/nil
```

Description

Helper function that gets the port expression for the passed port object. It can also get the expression for the port of an iterative instance at the passed index.

Arguments

A_formatterId	ID of the formatter object.
a_portId	ID of the port object.
x_iteration	Iteration index of the iterative instance.

Value Returned

t_portExpr	The formatted string for the connection to the port.
nil	The formatted string for the connection to the port was not generated.

Example

This code prints the port name and its expression as a message in the log file. It uses the `amsGetPortExpr` helper function to get the port expression.

```
(defun MYPrintPorts (formatterId cellviewId)
(let (ports)
    (setq ports cellviewId->ports
     ) ; setq
    (amsPrint formatterId "( ")
    (foreach port ports
        ;; cellview ports can have null port->expr
        (if (port->expr == nil) then
       sprintf(portExpr "No Port Expr")
         else
            sprintf(portExpr "%s" port->expr)
        )
        sprintf(tempStr "Port Name: %s, Port Expr: %s\n", port->name,
                amsGetPortExpr(formatterId port) )
        amsInfo(formatterId tempStr)
        (unless (equal port (car ports))
          (amsPrint formatterId ", ")
        ); unless
        (amsPrintPort formatterId port)
    ) ; foreach
    (amsPrint formatterId " );")
  ) ; let

);;defun

;; Set up the custom netlist procedure
netId = amsGetNetlister()
vlog = netId->vlog
vlog->portsProc = ‘MYPrintPorts
```

amsGetUniqueName

```
amsGetUniqueName( A_formatterId s_objectType )
     => t_objectName/nil
```

Description

Helper function that generates a unique, legal Verilog-AMS name for the specified object type. (A unique name is a name that is not already in the database.) If you need to insert a new object in the netlist, you can call this function to obtain a non-conflicting, unique name.

Arguments

A_formatterId	ID of the formatter object.
s_objectType	Kind of object to be named. Valid values: `'net`, `'instance`, `'alias`

Value Returned

t_objectName	The generated unique name.
`nil`	Unique name was not generated.

Example

This example uses `amsGetUniqueName` to create a name for a new node.

```
;; Check for property CUSTOM_GROUND on the CBN
;;
(when cvId->id->CUSTOM_GROUND
  ;; add a new ground node to the CBN
  gndName = amsGetUniqueName(formatterId 'net)
  amsPrint(formatterId strcat("\n\nelectrical " gndName ";\n") )
  amsPrint(formatterId strcat("ground " gndName ";\n") )
); when
```

amsInfo

```
amsInfo(
     A_formatterId t_msg
     )
     => t/nil
```

Description

Helper function that prints *t_msg* in the form of an informational message added to the log file.

Arguments

A_formatterId	ID of the formatter object.
t_msg	Message to be printed. If you want newline characters to appear in the message, you must include them in the message.

Value Returned

`t`	Message was printed.
`nil`	Message was not printed.

Example

You enter the following code in your netlist procedures override file. The code includes the `amsInfo` function as shown.

```
netlisterId = amsGetNetlister()

;; Get the Verilog-AMS formatterId.
vlogFormId = netlisterId->vlog

;; Override the default comment printing function.
vlogFormId->commentsProc = ‘MyCommentsProc

;; My function to print comments
(defun MyCommentsProc (formatterId cellViewId)
(amsInfo formatterId "Formatting with MyFormatter.\n      June 23,2003.\n")
;; We’ve overridden the original comment printing function, so next
;; line writes the original comments to the netlist.
(amsPrint formatterId formatterId->comments)
);defun
```

After you load the netlist procedures override file, the following message appears in the log when you netlist.

```
Info: Formatting with MyFormatter.
      June 23, 2003.
```

amsMapName

```
amsMapName(
     A_formatterId
     A_cellViewId
     t_name
     [ s_objectType ]
     )
     => t_mappedName/nil
```

Description

Helper function that checks whether the specified name is a legal Verilog-AMS identifier. If the name is legal, the function returns the name. If the name is not a valid Verilog-AMS identifier, the identifier is mapped.

Only the net, instance, and alias types are collision mapped. A name to be used as an instance master name does not need to be collision mapped so it is only checked for validity and, if necessary, mapped to a legal Verilog-AMS name.

Arguments

A_formatterId	ID of the formatter object.
A_cellViewId	Cellview object to which the name belongs.
t_name	Name to be checked and, if necessary, mapped.
s_objectType	Kind of object referred to by the name to be checked. If *s_objectType* is omitted, name collision checks are not performed. Valid values: `'net`, `'instance`, `'alias`, `'other` Default value: `'other`

Value Returned

t_mappedName	The name passed in, or, if necessary, a mapped transformation of the name.
`nil`	No name was returned.

Example

This example uses the `amsMapName` function to check and, if necessary, map the `newParam` name so that it does not clash with the name of an existing parameter.

```
;; Display the listed parameters
       (while (newParam != nil)
         amsPrint(formatterId "\nparameter ")
         mappedParamName = amsMapName(formatterId cvId newParam ’parameter)
         amsPrint(formatterId mappedParamName)
         amsPrint(formatterId "= 0;")
         paramList = cdr(paramList)
         newParam = car(paramList)
       ); while
```

amsMtlinePrintParams

```
amsMtlinePrintParams(
     A_formatterId
     A_cellViewId
     A_instanceId
     )
     => t/nil
```

Description

Helper function that prints the `r`, `l`, `g`, `c`, `rskin`, and `gdloss` parameters in the matrix format used by the mtline component. The `n` and `subcktfile` parameters are not printed but all other parameters are printed as they are by the `amsPrintParameters` function.

To use this function, it is normally provided as an element of a list in the `netlistProcedure` field in the ams simulation information (simInfo) section of the mtline device. The entry in that field would be as follows:

```
nil params amsMtlinePrintParams
```

The code for the `amsMtlinePrintParams` function is defined in the `analogLib/nportProcs.il` which is loaded by the `libInit.il` file in the `dfII/etc/cdslib/artist/analogLib` directory. If necessary, you can modify the function to meet specific needs.

Arguments

A_formatterId	ID of the formatter object.
A_cellViewId	ID of the cellview.
A_instanceId	ID of the instance.

Value Returned

`t`	Parameters were written in the mtline matrix format.
`nil`	Parameters were not correctly written.

Example

If the number of transmission lines is 2, the netlist generated by the `netlistProcedure` field entry given above might have a format like this:

```
mtline   #(.c({118E-12,-1,1}), .l({2.97E-7,1,1}), .g({0.236E-11,-1,1})
, .rskin({0.53E-3,1,1}), .r({2.03,1,1}), .gdloss({14.8E-12,-1,1}), .len(543.9664m)
) (*
integer library_binding = "analogLib";
 *)
wtrace1 ( net010,
net011, net_1, net_3, cds_globals.\gnd! , cds_globals.\gnd!  );
```

amsMtlineTermOrder

```
amsMtlineTermOrder(
     A_instanceId
     )
     => l_termOrder/nil
```

Description

Helper function that generates an appropriately ordered list of pins for the mtline component. The mtline component provided in `analogLib` is a parameterized cell (pcell) where the number of terminals depends upon the number of transmission lines specified by the user. The transmission lines, which are specified as the property value `n` on the schematic instance, determine the number of pins for the instance. The `amsMtlineTermOrder` function generates a list of these pins in the order necessary for an AMS simulation. To use the function, it is normally provided as the value in the CDF `termOrder` field in the `ams` simulation information (simInfo) section of the mtline device.

Arguments

`A_instanceId`	ID of the instance of the mtline primitive.

Value Returned

`l_termOrder`	The `termOrder` value required by the AMS simulator for the mtline primitive.
`nil`	The `termOrder` was not generated.

Example

If the number of transmission lines is 2, the `l_termOrder` return value is

```
(in1 out1 in2 out2 inref outref)
```

amsNportTermOrder

```
amsNportTermOrder(
     A_instanceId
     )
     => l_termOrder/nil
```

Description

Helper function that generates an appropriately ordered list of pins for the nport component. The nport component provided in `analogLib` is a parameterized cell (pcell) where the number of ports is specified by the user as the property value `p` on the schematic instance. The `amsNportTermOrder` function generates a list of these ports in the order necessary for an AMS simulation. To use the function, it is normally provided as the value in the CDF `termOrder` field in the `ams` simulation information (simInfo) section of the nport device.

This function is defined in the `analogLib/nportProcs.il` which is loaded via the `libInit.il` file in the `dfII/etc/cdslib/artist/analogLib` directory.

Arguments

`A_instanceId`	ID of the instance of the nport component.

Value Returned

`l_termOrder`	The `termOrder` value, which is a list of ordered terminals, required by the AMS simulator for the nport component.
`nil`	The `termOrder` was not generated.

Example

If the number of ports is 3, the `l_termOrder` return value is

```
(p1 m1 p2 m2 p3 m3)
```

amsPrint

```
amsPrint(
     A_formatterId
     t_msg
     [ s_sectionId ]
     )
     => t/nil
```

Description

Helper function that writes the specified string to the netlist. This function uses the AMS netlister internal IO buffering method, which generates automatic line breaks to ensure that line widths are reasonable.

Arguments

A_formatterId	ID of the formatter object.
t_msg	String to be written to the netlist. If you want newline characters to appear in the netlist, you must include them in the string.
s_sectionId	Symbol representing the buffer into which *t_msg* is printed. Valid values: `'INCLUDES_LIST`, `'MODULE_INTERFACE`, `'PORT_DECLARATION`, `'PARAMETER_DECLARATION`, `'SIGNAL_DECLARATION`, `'VLOG_INSTANCES`, `'END_MODULE` If you omit the *s_sectionId*, the default value is determined from the netlist procedure field being overwritten. For a list of the default *s_sectionId* value for each field see Table 2-3 on page 53.

Value Returned

`t`	Message string was written to the netlist.
`nil`	Message string was not written to the netlist.

Default s_sectionId Values

If you omit the *`s_sectionId`* option, the default value is determined from the *`formatterId`* field being overwritten. The defaults are listed in the following table. Because the default function for the `moduleProc` field (`amsPrintModule`) cannot be overridden, that field is not included in the table. The corresponding netlist sections listed in the third column are illustrated in the Identifying the Sections of a Netlist.

Table 2-3 Default s_sectionId Values

If the formatterId field being overridden is...	Then the default value for s_sectionId is...	Corresponding to this netlist section...
`attributesProc`	`'VLOG_INSTANCES` when *`A_objectId`* is an instanceId.	Instances
`commentsProc`	`'INCLUDES_LIST`	Includes list
`headersProc`	`'INCLUDES_LIST`	Includes list
`instanceMasterNameProc`	`'VLOG_INSTANCES`	Instances
`instanceParametersProc`	`'VLOG_INSTANCES`	Instances
`instancePortsProc`	`'VLOG_INSTANCES`	Instances
`instanceProc`	`'VLOG_INSTANCES`	Instances
`parametersProc`	`'PARAMETER_DECLARATION`	Parameter declarations

Example 1

You enter the following code in your netlist procedures override file. The code includes the `amsPrint` function as shown.

```
netlisterId = amsGetNetlister()
;; Get the Verilog-AMS formatterId.
vlogFormId = netlisterId->vlog
;; Override the default comment printing function.
;; Overriding the commentsProc field means the default
;; for s_sectionId is 'INCLUDES_LIST.
vlogFormId->commentsProc = ‘MyCommentsProc
;; My function to print comments
(defun MyCommentsProc (formatterId cellViewId)
(amsPrint formatterId "// Formatted with MyFormatter.\n// June 23, 2003.\n")
);defun
```

After you load the netlist procedures override file, new netlists contain the specified comments, followed by an empty line. The default comments provided by AMS Designer no longer appear.

```
// Formatted with MyFormatter.
// June 23, 2003.
```

Example 2

You enter the following code in your netlist procedures override file. The code includes the `amsPrint` function as shown.

```
netlisterId = amsGetNetlister()

;; Get the Verilog-AMS formatterId.
vlogFormId = netlisterId->vlog

; The default s_sectionId of MYPrintInstance is set to 'VLOG_INSTANCES'
; because MYPrintInstance is set on the instanceProc field.
vlog->instanceProc = `MYPrintInstance

;; My function to print instances
(defun MYPrintInstance (formatterId cellViewId instanceId)
   (amsPrint formatterId "\n//Comment 1 in 'VLOG_INSTANCES\n")
   (amsPrint formatterId "\n//Comment 2 in 'INCLUDES_LIST\n" 'INCLUDES_LIST)
   (amsPrintInstance formatterId cellViewId instanceId)
);defun
```

The custom netlist procedure `MYPrintInstance` overrides the default netlist procedure for the field `instanceProc`. Therefore, `'VLOG_INSTANCES` becomes the default *`s_sectionId`* for any use of the `amsPrint` function within the `MYPrintInstance` function.

In this example, the first `amsPrint` function call does not specify the *`s_sectionId`* so the default `'VLOG_INSTANCES` value is used to print `Comment 1`. The second `amsPrint` function call explicitly specifies the *`s_sectionId`* as `'INCLUDES_LIST` so the `'INCLUDES_LIST` value is used to print `Comment 2`. (Comment 2 appears twice in the netlist because it is printed for each instance and there are two instances.) As a result, the netlist includes comments as shown here.

```
// Verilog-AMS netlist generated by the AMS netlister, version 5.0.33.110.
// Cadence Design Systems, Inc.

`include "disciplines.vams"
`include "constants.vams"

//Comment 2 in 'INCLUDES_LIST

//Comment 2 in 'INCLUDES_LIST

module comparator ( inp,inn,out );

input   inp;
input   inn;
output   out;

parameter chivalue=5.0;

//Comment 1 in 'VLOG_INSTANCES
```

```
pmos4 #(.l(3u), .region("triode"), .w(40u))
(* integer library_binding = "amslib";
integer cds_net_set[0:0]= {"bulk_n"};
integer bulk_n = "cds_globals.\\vdd! ";  *)
M11 ( net92, cds_globals.„nd! , net79, cds_globals.\vdd!  );
```

```
//Comment 1 in ’VLOG_INSTANCES
```

```
isource #(.type("dc"), .dc(cds_globals.idc))  (*
integer library_binding = "analogLib";  *) I3 ( vref1,
cds_globals.„nd!  );
```

```
endmodule
```

amsPrintAlias

```
amsPrintAlias(
     g_formatterId
     g_aliasId
     )
     => t/nil
```

Description

Generates instances of the `cds_alias` module with the format illustrated by the following examples.

```
cds_alias #(.width(1))
    (*   integer library_binding = "basic";
         integer view_binding = "functional"; *)
    ams_alias_inst_0 (net015, net014[0]);
cds_alias #(.width(2))
    (*   integer library_binding = "basic";
         integer view_binding = "functional"; *)
    ams_alias_inst_1 ({ net06[0],net06[1] }, {net09[3],net09[7] });
```

You can use this function to print particular aliases. For example, to effectively suppress some aliases, you can override the `amsPrintAliases` function, then iterate over the aliases using this function to print just the ones you want.

If you override the `amsPrintAliases` function and choose not to use this `amsPrintAlias` helper function, you must ensure that the `library_binding` and `view_binding` attributes are printed properly. The elaborator cannot resolve `cds_alias` instantiations without these attributes.

Arguments

`g_formatterId`	ID of the formatter object. The formatter object holds information about the netlist procedures supported for the formatter.
`g_aliasId`	ID of the alias object.

Value Returned

`t`	Instance of the `cds_alias` module was written.
`nil`	Instance of the `cds_alias` module was not written.

Example

In this example, the `amsPrintAlias` helper function is used to write the alias instance in the netlist as usual. In addition, the name and association list of every alias object is printed as an informational message in the log file.

```
(defun MYPrintAliases (formatterId cellviewId)
  (let (aliases)
    aliases = cellviewId->aliases
    ;; Now, print all the alias association list, one by one
    (foreach alias aliases
    sprintf(tempStr "Alias Name: %s Assocs: %L\n\n\n" alias->name alias->assocs)
    amsInfo(formatterId tempStr)
    amsPrintAlias(formatterId alias)
    )
  )
);;

;; Set up the custom netlist procedure
netId = amsGetNetlister()
vlog = netId->vlog
vlog->aliasesProc = ‘MYPrintAliases
```

amsPrintAliases

```
amsPrintAliases(
     A_formatterId
     A_cellViewId
     A_instanceId
     )
     => t/nil
```

Description

Default netlisting procedure to generate the alias declarations for the cellview and print them one by one.

Arguments

A_formatterId	ID of the formatter object.
A_cellViewId	ID of the cellview.
A_instanceId	ID of the instance.

Value Returned

`t`	Alias declarations were printed.
`nil`	Alias declarations were not printed.

Example

In this example, the `amsPrintAliases` default netlisting procedure writes the alias instances into the netlist as usual. The name and association list of every alias object is printed as an informational message in the log file.

```
(defun MYPrintAliases (formatterId cellviewId)
   (let (aliases)
      aliases = cellviewId->aliases
      ;; Print the alias association list, one by one
      (foreach alias aliases
      sprintf(tempStr "Alias Name: %s Assocs: %L\n\n\n" alias->name alias->assocs)
      amsInfo(formatterId tempStr)
      )
      amsPrintAliases(formatterId cellviewId)
   )
);;
```

```
;; Set up the custom netlist procedure
netId = amsGetNetlister()
vlog = netId->vlog
vlog->aliasesProc = ‘MYPrintAliases
```

amsPrintAttribute

```
amsPrintAttribute(
     g_formatterId
     g_attributeId
     )
     => t/nil
```

Description

Helper function that prints the attribute specified by *`g_attributeId`*.

Arguments

`g_formatterId`	ID of the formatter object. The formatter object holds information about the netlist procedures supported for the formatter.
`g_attributeId`	ID of the attribute object.

Value Returned

`t`	Attribute object was printed.
`nil`	Attribute object was not printed.

Example

This example uses the `amsPrintAttribute` helper function to write the attribute in the netlist.

```
(defun MYPrintAttributes (formatterId objectId)
  (let (attrs attr)
    (if (objectId->attributes) then
       attrs = objectId->attributes
       (if (formatterId->ifdefLanguageExtensions) then
          amsPrint(formatterId "‘ifdef INCA\n")
       )
       amsPrint(formatterId "(*\n")
       (foreach attr attrs
           amsPrintAttribute(formatterId attr)
       )
       amsPrint(formatterId " *)\n")
       (if (formatterId->ifdefLanguageExtensions) then
          amsPrint(formatterId "‘endif\n")
       )
    );;endif
  ); let
);;defun

;; Set up the custom netlist procedure
netId = amsGetNetlister()
vlog = netId->vlog
vlog->attributesProc = ‘MYPrintAttributes
```

amsPrintAttributes

```
amsPrintAttributes(
     A_formatterId
     A_objectId
     )
     => t/nil
```

Description

Default netlisting procedure for printing the attributes of the object specified by *`A_objectId`*.

Be aware that attributes play an important role in controlling elaboration and simulation. If you are netlisting for the AMS simulator, for example, omitting attributes can have the following consequences:

Omitting this attribute...	Can result in...
`library_binding`	Binding an instance to a library other than that intended by the schematic
`view_binding`	Binding an instance to a view other than that intended by the schematic
`cds_net_set`, `inh_conn_prop_name`, `inh_conn_def_value`	Generating incorrect or incomplete inherited connections specifications
`passed_mfactor`	Using incorrect multiplication factors during simulation
`elaboration_binding`	Causing errors during elaboration because components cannot be found
`supplySensitivity`, `groundSensitivity`	Using incorrect power and ground values during simulation

Different simulators use different attributes, so the consequences of omitting or incorrectly setting an attribute depend on the simulator.

To help avoid problems such as those listed in the table, use the default `amsPrintAttributes` function as a helper function in your customized netlisting procedure when you override the `instancePortProc`. That way, the AMS netlister continues to generate the attributes for the netlist.

Arguments

A_formatterId	ID of the formatter object.
A_objectId	ID of the object. Currently, an *instanceId* is the only value supported for *A_objectId*.

Value Returned

t	Attributes of the object were printed.
nil	Attributes of the object were not printed.

Example

In this example, you have a function called `isDigitalGate`, which determines whether a gate is a digital gate. You use that function to determine whether to print the attributes to the netlist. If the attributes do need to be printed, use the default function `amsPrintAttributes` to do the work.

```
;; Do not print attributes for instances of digital gates.
(defun DontPrintInstanceAttributesForDigitalGates (formatterId objectId)

  ;; Determine if it is an instance of a digital gate.
  digitalGate = isDigitalGate(objectId->masterName)

  (if (digitalGate == nil) then
    amsPrintAttributes(formatterId objectId)
  );; if

)

;; ====================================================================
;; Set up area
;; ====================================================================
netlisterId = amsGetNetlister()
formatterId = netlisterId->vlog
formatterId->attributesProc = 'DontPrintInstanceAttributesForDigitalGates
```

amsPrintInstance

```
amsPrintInstance(
     A_formatterId
     A_cellViewId
     A_instanceId
     )
     => t/nil
```

Description

Default netlisting procedure for printing an instance, including the instance master name, the instance parameters override list, the attributes for the instance, the name of the instance, and the instance port list.

If you override this function, you must

- Develop an algorithm for obtaining the master name
- Handle the `library_binding`, `view_binding`, `elaboration_binding`, `inh_conn_prop_name`, `inh_conn_def_value`, `passed_mfactor`, `supplySensitivity`, `groundSensitivity`, and `cds_net_set` attributes
- Determine how to print instance parameters

To help meet these requirements, you can make use of the `amsPrintInstanceMasterName`, `amsPrintInstanceParameters`, `amsPrintInstancePorts`, and `amsPrintAttributes` functions, as well as the various helper functions. In fact, the simplest and easiest-to-maintain approach to achieving your goals might be to override just these functions, leaving the `amsPrintInstance` function running as it does by default.

Arguments

`A_formatterId`	ID of the formatter object.
`A_cellViewId`	ID of the cellview object.
`A_instanceId`	ID of the instance object.

Value Returned

`t`	Instance was printed.

`nil`	Instance was not printed.

Example

In this example, the default `amsPrintInstance` function is called to print the instance as usual, then additional monitoring code is inserted in the netlist automatically.

```
;; ========================================================================
;; A custom netlist procedure to generate $display for listed signals.
;; ========================================================================
(defun MYInstanceSignalMonitor (formatterId cellviewId instanceId)

  ;; Call the default instance procedure
  amsPrintInstance(formatterId cellviewId instanceId)

  (progn

     ;; Check for property CLOCK_MONITOR on the instance
     ;;
     (when instanceId->id->CLOCK_MONITOR

         (setq signal_list (parseString instanceId->id->CLOCK_MONITOR))

         monitor_signal = car(signal_list)
         print_debug_signals = 0

         (if (monitor_signal != nil) then

           amsPrint(formatterId "\n// Debug signals at every clock transition")
           amsPrint(formatterId "always @(posedge(clock) or negedge(clock))\n")
           amsPrint(formatterId "begin")
           amsPrint(formatterId "  $display($stime, 'Signal values are:');")
           print_debug_signals = 1
         ); if

         ;; Display the listed signals
         (while (monitor_signal != nil)
           display_signal = strcat("  $display('" monitor_signal ": %b', " mon
itor_signal ");  ")
           amsPrint(formatterId display_signal)
           signal_list = cdr(signal_list)
           monitor_signal = car(signal_list)
         )

         (if (print_debug_signals == 1) then
           amsPrint(formatterId "\nend \n")
         ); if

     ) ; when

  ) ; progn

) ; defun

;; ========================================================================
;; Set up area
;; ========================================================================
netlisterId = amsGetNetlister()
formatterId = netlisterId->vlog

;; Override the printing of instance netlist procedure
formatterId->instanceProc = 'MYInstanceSignalMonitor
```

amsPrintInstanceMasterName

```
amsPrintInstanceMasterName(
     A_formatterId A_cellViewId
     A_instanceId
     )
     => t/nil
```

Description

Default netlisting procedure for printing the name of an instance master. By overriding this function, you can modify the name of the instance master.

Arguments

A_formatterId	ID of the formatter object.
A_cellViewId	ID of the attribute object.
A_instanceId	ID of the instance object.

Value Returned

t	Name of the instance master was printed.
nil	Name of the instance master was not printed.

Example

You enter the following code in your netlist procedures override file.

```
(setq MYNetlister (amsGetNetlister))
;; Override the function that prints the master name.
MYNetlister->vlog->instanceMasterNameProc = ‘MYInstMasterNameProc

(defun MYInstMasterNameProc ( formatterId cellViewId instanceId )
    ;; Check the name of the instance "fingers" property. Use the value
    ;; to generate the name of the master.

    (setq numFingers instanceId->id->fingers)

    (if (numFingers != nil) then
       ;; print the custom instance master name.
       (amsPrint formatterId (sprintf nil "\n%s_%d"
            instanceId->masterName numFingers) )
     else
       ;; print using the default instance master name netlist procedure.
       amsPrintInstanceMasterName(formatterId cellViewId instanceId)
    ); if

t
) ; defun
```

The `MYInstMasterNameProc` procedure generates and prints master names like `nmos_1` or `nmos_3` if the instance of `nmos` has 1 or 3 fingers and a master name like `capacitance` if the instance of `capacitance` has no property called `fingers`.

amsPrintInstanceParameter

```
amsPrintInstanceParameter(
     A_formatterId
     t_instanceName
     A_parameterId
     )
     => t/nil
```

Description

Helper function that prints the instance parameter specified by *`A_parameterId`*.

You can use this function to filter the parameters of an instance. For example, you can override the `amsPrintInstanceParameters` function, then iterate over the parameters using this `amsPrintInstanceParameter` function to print the parameters you want to retain.

Arguments

`A_formatterId`	ID of the formatter object.
`t_instanceName`	Name of the instance that has the parameter to be printed.
`A_parameterId`	ID of the parameter to be printed.

Value Returned

`t`	Instance parameter was printed.
`nil`	Instance parameter was not printed.

Example

This example prepares and prints a list of parameters, calling the `amsPrintInstanceParameter` helper function to write each one.

```
;; Customize the parameter "r" for resistor to be 4K always.
(defun MYPrintInstanceParameters (formatterId cvId instanceId)
   (if (instanceId->masterName == "resistor") then
       (amsPrint formatterId "#(")
       ;; Go through the list of parameters for resistor
       (foreach param instanceId->parameters
         (unless (equal param (car instanceId->parameters))
           (amsPrint formatterId ",")
         ); unless

         ;; Print only the parameter called "r"
         (if (param->name == "r") then

           ;; Compute my_new_value
           my_new_value = "4K"

           ;; Set the value of "r" to new value
           param->value = my_new_value

           instanceName = (amsGetInstanceName formatterId instanceId)

           ;; Call the helper function to print the parameter
           amsPrintInstanceParameter(formatterId instanceName param)
         )

      );foreach

      (amsPrint formatterId ")")
  ); if

  ;; For any instance whose masterName is NOT "resistor", print its
  ;; parameters in the default way using the default print function.
  ;;
  (if (instanceId->masterName != "resistor") then
       amsPrintInstanceParameters(formatterId cvId instanceId)
  )

); defun
```

amsPrintInstanceParameters

```
amsPrintInstanceParameters(
     A_formatterId
     A_cellViewId
     A_instanceId
     )
     => t/nil
```

Description

Default netlisting procedure for printing instance parameters.

If possible, do not override this function. Instead, consider changing the parameters list directly and then calling the `amsPrintInstanceParameters` function to print the changed list.

Arguments

A_formatterId	ID of the formatter object.
A_cellViewId	ID of the cellview object.
A_instanceId	ID of the instance object.

Value Returned

`t`	Instance parameters were printed.
`nil`	Instance parameters were not printed.

Example

This example illustrates how you can change the individual parameter values and then use the `amsPrintInstanceParameters` function to print the changed list.

```
(defun MYPrintInstanceParameters ( formatterId cellViewId instanceId )
    (foreach parameter instanceId->parameters
        ;; Change the individual parameter values
        parameter->value = <newVal>
        ...
        ) ; foreach
    ;; Delete parameters in the parameter list
    instanceId->parameters = newList
    ;; But call the default procedure
    (amsPrintInstanceParameters formatterId cellViewId instanceId)
  ) ; defun
```

amsPrintInstancePorts

```
amsPrintInstancePorts(
     A_formatterId
     A_instanceId
     [x_iteration]
     )
     => t/nil
```

Description

Default netlisting procedure for printing the port list of an instance or of a particular iteration of an instance. The ports in the port list are arranged according to the value specified by the CDF `termOrder` property, if the `termOrder` property exists for the instance. Otherwise, the order of the ports is undetermined. The `amsPrintInstancePorts` function prints the port list by order or by named port maps, according to the effective options and settings.

The `amsPrintInstancePorts` functions does not print punctuation at the end of the port list, nor does it insert newline characters to break lines. However, the underlying implementation of `amsPrint` can insert newline characters at appropriate places to control line lengths.

Arguments

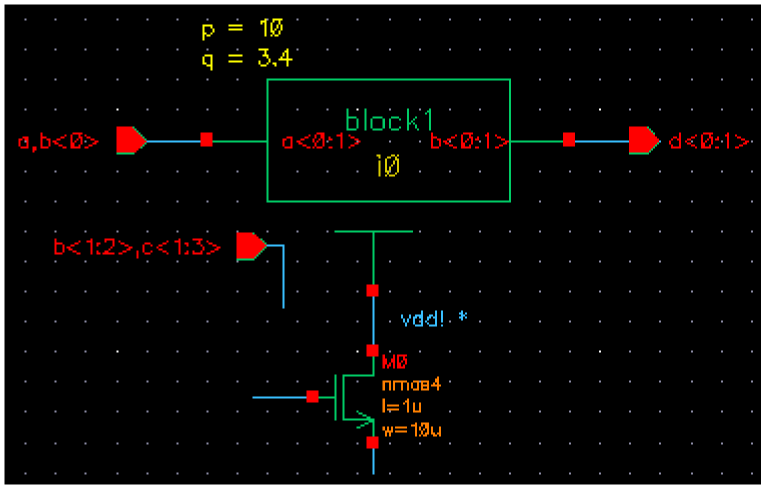
A_formatterId	ID of the formatter object.
A_instanceId	ID of the instance object.
x_iteration	Iteration number specifying a particular iterated instance. A value of `-1` indicates that the port list of an iterated instance is not to be split among the iterations of an instance. The default value for this argument is `-1`.

Value Returned

`t`	Port list was printed.
`nil`	Port list was not printed.

Example

You have the following schematic to be netlisted.



The instance terminal connections for instance `i0` of master `block1` are:

- Instance terminal `a<0:1>` is connected to net `a,b<0>`
- Instance terminal `b<0:1>` is connected to net `d<0:1>`

Calling the `amsPrintInstancePorts` function on this schematic generates a port list as follows:

```
( .b( d[0:1] ), .a( { a,b[0] } ) )
```

amsPrintIO

```
amsPrintIO(
     A_formatterId
     A_ioId
     )
     => t/nil
```

Description

Helper function that prints the port specified by *A_ioId*. If *A_ioId* has an inherited net expression, the expression is printed as an attribute.

Arguments

A_formatterId	ID of the formatter object.
A_ioId	ID of the port.

Value Returned

t	The port was printed.
nil	The port was not printed.

Example

This example illustrates a function that prints IOs sorted by name.

```
(defun MYIONetProc ( formatterId cellViewId)
   ;; Prints the IO list
   (foreach io (sort cellViewId->IOs (lambda (a b) (alphalessp a->name b->name)))
      (amsPrintIO formatterId io)
      ) ; foreach
   t
) ; defun

;; Set up the custom netlist procedure
netId = amsGetNetlister()
vlog = netId->vlog
vlog->iosProc = 'MYIONetProc
```

amsPrintIOs

```
amsPrintIOs(
     A_formatterId
     A_cellViewId
     )
     => t/nil
```

Description

Default procedure to generate the IO declarations for the port list of the cellview and print them one by one. Also prints inherited net expressions as attributes.

Arguments

A_formatterId	ID of the formatter object.
A_cellViewId	ID of the cellview object.

Value Returned

t	IO declarations were printed.
nil	IO declarations were not printed.

Example

To print IOs sorted by name you might write a function like this.

```
(defun MYIONetProc ( formatterId cellViewId)
    ;; Prints the IO list
    (foreach io (sort cellViewId->IOs (lambda (a b) (alphalessp a->name b->name)))
        (amsPrintIO formatterId io)
        ) ; foreach
    t
    ) ; defun

;; Set up the custom netlist procedure
netId = amsGetNetlister()
vlog = netId->vlog
vlog->iosProc = 'MYIONetProc
```

amsPrintParameter

```
amsPrintParameter(
     A_formatterId
     A_parameterId
     )
     => t/nil
```

Description

Helper function that prints the parameter specified by *`A_parameterId`*.

Arguments

`A_formatterId`	ID of the formatter object.
`A_parameterId`	ID of the parameter to be printed.

Value Returned

`t`	Parameter was printed.
`nil`	Parameter was not printed.

Example

This example changes the default value for the `frequency` parameter, then uses the `amsPrintParameter` function to write the parameter to the netlist.

```
/* Change the default value only for the parameter named "frequency" */
  (foreach param cvId->parameters
           (if (param->name == "frequency") then
             param->value = "0"
           ); if

           (amsPrintParameter formatterId param)

  ); foreach
```

amsPrintParameters

```
amsPrintParameters(
     A_formatterId
     A_cellViewId
     )
     => t/nil
```

Description

Default netlisting procedure for generating the parameter declarations for a cellview and printing the declarations one by one. The parameters are obtained from the base cell CDF and from parameters on the instances of the cellview being netlisted that have pPar references. The actual list of parameters is determined by the `ams` section of the simInfo (which can be used to specify parameters to include and parameters to exclude) as well as by library and cell CDF.

Arguments

`A_formatterId`	ID of the formatter object.
`A_cellViewId`	ID of the cellview object.

Value Returned

`t`	Parameter declarations were printed.
`nil`	Parameter declarations were not printed.

Example

This example code first calls the default `amsPrintParameters` function to write the regular parameters to the netlist. The remainder of the code checks for custom parameters, and if they exist, uses the `amsPrint` function to write them to the netlist too.

```
;; print the default CBN parameters
  amsPrintParameters(formatterId cvId)
  (let (newParam printInfo)
     ;; Check for property CUSTOM_PARAMS on the CBN
     ;;
     (when cvId->id->CUSTOM_PARAMS
       (setq paramList (parseString cvId->id->CUSTOM_PARAMS))
       newParam = car(paramList)
       printInfo = 0

       ;; Send informative messages to the log file
       (if (newParam != nil) then
         (amsInfo formatterId "Adding custom CBN parameters to the netlist..\n")
         printInfo = 1
       ); if

       ;; Display the listed parameters
       (while (newParam != nil)
         amsPrint(formatterId "\nparameter ")
         mappedParamName = amsMapName(formatterId cvId newParam ’parameter)
         amsPrint(formatterId mappedParamName)
         amsPrint(formatterId "= 0;")
         paramList = cdr(paramList)
         newParam = car(paramList)
       ); while

       (if (printInfo == 1) then
         (amsInfo formatterId "Done.\n")
       ); if

     ); when
 ); let
```

amsPrintPort

```
amsPrintPort(
     g_formatterId
     g_portId
     [ x_iteration ]
     )
     => t/nil
```

Description

Helper function that prints the port specified by `g_portId`. If the instance master is a primitive device, this function prints ports with connections specified by order by simply printing the port expression. If the instance master is not a primitive device, but there *is* a port expression involved, the function prints ports with connections specified by order. If the instance master is not a primitive device, and there *is no* port expression involved, the function prints ports with connections specified by name.

Arguments

g_formatterId	ID of the formatter object.
g_portId	ID of the port.
x_iteration	Iteration index of the iterative instance.

Value Returned

`t`	Port object was printed.
`nil`	Port object was not printed.

Example

This code prints the port name and its expression as a message in the log file. It uses the `amsPrintPort` helper function to netlist the port in the cellview.

```
(defun MYPrintPorts (formatterId cellviewId)
(let (ports)
   (setq ports cellviewId->ports
      ) ; setq
   (amsPrint formatterId "( ")
   (foreach port ports
      ;; cellview ports can have null port->expr
      (if (port->expr == nil) then
         sprintf(portExpr "No Port Expr")
      else
      sprintf(portExpr "%s" port->expr)
      )
      sprintf(tempStr "Port Name: %s, Port Expr: %s\n", port->name, portExpr)
         amsInfo(formatterId tempStr)
      (unless (equal port (car ports))
      (amsPrint formatterId ", ")
      ); unless
      (amsPrintPort formatterId port)
   ) ; foreach
   (amsPrint formatterId " );")
   ) ; let

);;defun

;; Set up the custom netlist procedure
netId = amsGetNetlister()
vlog = netId->vlog
vlog->portsProc = ‘MYPrintPorts
```

amsPrintPorts

```
amsPrintPorts(
     A_formatterID
     A_cellViewId
     )
     => t/nil
```

Description

Default netlisting procedure for generating the port list of a cellview and printing the ports one by one. The ports in the list are arranged according to the termOrder, termMap, or portOrder, if those characteristics are specified. The port list also contain any ports specified in the extraTerminals section of the ams simulation information (simInfo).

Arguments

A_formatterId	ID of the formatter object.
A_cellViewId	ID of the cellview object.

Value Returned

`t`	The port list was printed.
`nil`	The port list was not printed.

Example

This example code prints the port name and its expression as a message in the log file. It then uses the `amsPrintPorts` default netlist procedure to netlist the ports of the cellview.

```
(defun MYPrintPorts (formatterId cellviewId)
(let (ports)
    (setq ports cellviewId->ports
    ) ; setq
    (foreach port ports
        (if (port->expr == nil) then
        sprintf(portExpr "No Port Expr")
         else
            sprintf(portExpr "%s" port->expr)
        )
        sprintf(tempStr "Port Name: %s, Port Expr: %s\n", port->name, portExpr)
        amsInfo(formatterId tempStr)
    ) ; foreach
    amsPrintPorts(formatterId cellviewId)
```

```
  ) ; let
);;defun

;; Set up the custom netlist procedure
netId = amsGetNetlister()
vlog = netId->vlog
vlog->portsProc = ‘MYPrintPorts
```

amsPrintWire

```
amsPrintWire(
     g_formatterId
     g_wireId
     )
     => t/nil
```

Description

Prints the wire (or other type if net type or net discipline are defined) specified by `g_wireID`. If necessary, this function also prints a wire declaration.

Arguments

g_formatterId	ID of the formatter object. The formatter object holds information about the netlist procedures supported for the formatter.
g_wireId	ID of the wire object.

Value Returned

`t`	Wire object was printed.
`nil`	Wire object was not printed.

Examples

This example collects and prints out information about the characteristics of each wire and then writes the wire to the netlist in the usual way.

```
(defun MYPrintWires (formatterId cellviewId)
  (let (wires)
    wires = cellviewId->wires
    (foreach wire wires
        (if (car(wire->range) == nil) then
           lsb = "nil"
         else
           sprintf(lsb "%d" car(wire->range))
         )
        (if (cadr(wire->range) == nil) then
          msb = "nil"
          else
          sprintf(msb "%d" cadr(wire->range) )
        )
        (if (wire->type == nil) then
          type = "nil"
         else
          type = wire->type
        )
        (if (wire->discipline == nil) then
          discipline = "nil"
         else
          discipline = wire->discipline
        )
        sprintf(tempStr "Wire Name: %s, Range: [%s:%s], Type: %s, Discipline: %s\n"
                wire->name lsb msb type discipline)
        amsInfo(formatterId tempStr)
       amsPrintWire(formatterId wire)
    ) ; foreach
) ; let
);;defun

;; Set up the custom netlist procedure
netId = amsGetNetlister()
vlog = netId->vlog
vlog->wiresProc = 'MYPrintWires
```

amsPrintWires

```
amsPrintWires(
     A_formatterId
     A_cellViewId
     )
     => t/nil
```

Description

Default netlist procedure to generate the wire declarations for the cellview and print them one by one. Individual wires are collapsed and merged where necessary to form a single declaration. If holes exist in the ranges (sparse buses), the wires are over-declared.

Arguments

A_formatterId	ID of the formatter object.
A_cellViewId	ID of the cellview object.

Value Returned

`t`	The wire declarations were printed.
`nil`	The wire declarations were not printed.

Example

This code prints the wire name, its range, type and discipline as a message in the log file. It then uses the `amsPrintWires` default netlist procedure to netlist the required wires in the cellview.

```
(defun MYPrintWires (formatterId cellviewId)
  (let (wires)
    wires = cellviewId->wires
    (foreach wire wires
         (if (car(wire->range) == nil) then
            lsb = "nil"
          else
            sprintf(lsb "%d" car(wire->range))
          )
         (if (cadr(wire->range) == nil) then
           msb = "nil"
           else
           sprintf(msb "%d" cadr(wire->range) )
         )
```

```
        (if (wire->type == nil) then
          type = "nil"
         else
          type = wire->type
        )
        (if (wire->discipline == nil) then
          discipline = "nil"
         else
          discipline = wire->discipline
        )
        sprintf(tempStr "Wire Name: %s, Range: [%s:%s], Type: %s, Discipline: %s\n"
wire->name lsb msb type discipline)
        amsInfo(formatterId tempStr)
    ) ; foreach
    amsPrintWires(formatterId cellviewId)
  ) ; let
);;defun

;; Set up the custom netlist procedure
netId = amsGetNetlister()
vlog = netId->vlog
vlog->wiresProc = 'MYPrintWires
```

amsSpectreToVams

```
amsSpectreToVams(
     s_netlistDirectory
     )
     => t/nil
```

Description

Reads `netlist.oss` in the specified directory and writes it to `netlist.vams` in the same directory after translating the Spectre-language statements between `_ANALOG_BEGIN` and `_ANALOG_END` to the Verilog-AMS language. If `netlist.vams` already exists, this function overwrites it.

Note: You can edit `netlist.oss` and call this function to translate to Verilog-AMS.

Argument

`s_netlistDirectory`	Absolute path to the directory containing `netlist.oss`.

Value Returned

`t`	Successful translation.
`nil`	Unsuccessful translation.

Example

```
amsSpectreToVams("/cds/user123/simulation/my_cell/ams/config_ams/netlist/")
```

Translates `netlist.oss` in `/cds/user123/simulation/my_cell/ams/config_ams/netlist/` from Spectre to Verilog-AMS and writes `netlist.vams`.

amsWarning

```
amsWarning(
     A_formatterId
     t_msg
     )
     => t/nil
```

Description

Helper function that prints the given string in the form of a warning added to the log file.

Arguments

A_formatterId	ID of the formatter object.
t_msg	The warning message to be printed. If you want newline characters to appear in the message, you must include them in the message.

Value Returned

`t`	Warning message was printed.
`nil`	Warning message was not printed.

Example

You enter the following code in your netlist procedures override file. The code includes the `amsWarning` function as shown.

```
netlisterId = amsGetNetlister()
;; Get the Verilog-AMS formatterId.
vlogFormId = netlisterId->vlog
;; Override the default comment printing function.
vlogFormId->commentsProc = 'MyCommentsProc
;; My function to print comments
(defun MyCommentsProc (formatterId cellViewId)
(amsWarning formatterId "Too many closing parentheses.\n      Ignoring extra
parentheses and continuing.\n")
);defun
```

After you load the netlist procedures override file, the following message appears when you netlist.

```
Warning: Too many closing parentheses.
      Ignoring extra parentheses and continuing.
```