cādence®

Virtuoso® AMS Designer Environment User Guide

Product Version 6.1.6
August 2014

Contents

Preface

The Virtuoso® AMS Designer environment provides a framework for developing, simulating, and debugging mixed-signal design blocks. Using the AMS Designer environment, you can

- Design mixed-signal, mixed-language blocks in an environment that supports both schematic and text data entry
- Access AMS forms and features from the Cadence Hierarchy Editor, the schematic editor, or the command interpreter window
- Set up, netlist, and run your AMS simulation automatically from the schematic environment or completely standalone from the command line
- Create netlists:
    - a (full or incremental) netlist for an entire design
    - a netlist for an entire library
- Simulate using the Spectre, UltraSim or APS solver of the AMS Designer simulator
- Import and use Tcl command and analog simulation control files
- Interactively debug your design and investigate the results
- Display analog, digital, and mixed-signal waveforms in one enviroment

Note: Cadence offers the Virtuoso Analog Design Environment (ADE) for analog-oriented designs. ADE shares many of the same forms, functionality, state files, and default mechanisms with the AMS Designer environment but caters to analog designs.

To get started, see Chapter 1, "Getting Started with AMS Designer."

See the following topics for additional information in this preface:

- Scope of this Guide on page 22
- Licensing for the AMS Designer Environment on page 22
- Related Documents for the AMS Designer Environment on page 22
- Third-Party Software for Viewing Video Clips on page 23
- Typographic and Syntax Conventions on page 23

- Data Type Prefixes for SKILL Arguments on page 26
- Additional Learning Resources on page 27

Scope of this Guide

All the functionality described in this guide is available in IC6.1.6 and ICADV12.1 onward unless otherwise noted. Features that are supported only in a particular release are identified using (ICADV12.1 only) and (IC6.1.6 only) labels.

Licensing for the AMS Designer Environment

There is no licensing information specific to the Virtuoso® AMS Designer environment. General licensing information for the Virtuoso design environment is available in the *Virtuoso Software Licensing and Configuration User Guide*.

Related Documents for the AMS Designer Environment

For more information about the AMS simulator and related products, consult the sources listed below.

- *Cadence Application Infrastructure User Guide*
- *Cadence Hierarchy Editor User Guide*
- *Virtuoso AMS Designer Environment SKILL Reference*
- *Cadence Library Manager User Guide*
- *Cadence Verilog-AMS Language Reference*
- *Cadence VHDL-AMS Overview*
- *Component Description Format User Guide*
- *IEEE Std 1076.1*. Available from IEEE.
- *Instance-Based View Switching Application Note*
- *Virtuoso NC Verilog Environment User Guide*
- *Verilog-AMS Language Reference Manual*. Available from Open Verilog International.

- *Virtuoso AMS Designer Simulator User Guide*
- *Virtuoso Analog Design Environment User Guide*
- Virtuoso Mixed-Signal Circuit Design Environment User Guide
- *Virtuoso Schematic Editor User Guide*
- *Virtuoso Spectre Circuit Simulator Reference*
- *Virtuoso Spectre Circuit Simulator and Accelerated Parallel Simulator User Guide*
- *Virtuoso UltraSim Simulator User Guide*
- *Virtuoso Parasitic Simulation User Guide*

For information about problems, see the *Virtuoso AMS Designer Environment Known Problems and Solutions*.

Third-Party Software for Viewing Video Clips

To view any `.swf` multimedia files (which appear as Show Me hot links), you need:

- Access to the Cadence Online Support website.

  Contact your CAD department or your local Cadence representative for information about obtaining access to the Cadence Online Support website.

  Note: For access to the entire CIC Video Library, click here.
- Flash-enabled web browser such as Internet Explorer 5.0 or later, Netscape 6.0 or later, or Mozilla Firefox 1.6 or later. Alternatively, you can download Flash Player (version 6.0 or later) directly from the Adobe website.
- Speakers and a sound card for videos that have audio narration.

Typographic and Syntax Conventions

In general, the text in this book follow these typographic and syntax conventions:

`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.

For other more specialized text, the following typographical conventions apply:

- The definition operator, `::=` , defines more complex elements of the Verilog-AMS language in terms of less complex elements.
- Lowercase words represent syntactic categories. For example,

  `module_declaration`

  Some names begin with a part that indicates how the name is used. For example,

  `node_identifier`

  represents an identifier that is used to declare or reference a node.
- Boldface words represent elements of the syntax that must be used exactly as presented (except as noted below). Such items include keywords, operators, and punctuation marks. For example,

  `endmodule`

  Sometimes options can be abbreviated. The shortest permitted abbreviation is shown by capital letters but you can use either upper or lower-case letters in your code. For example, the syntax

  `-CHecktasks`

means that you can type the option as `-checktasks`, `-CHECKTASKS`, `-ch`, `-CH`, `-cH`, and so on.

- Vertical bars indicate alternatives. You can choose to use any one of the items separated by the bars. For example,

```
attribute ::=
    abstol
   |access
   |ddt_nature
   |idt_nature
   |units
   |huge
   |blowup
   |identifier
```

- Square brackets enclose optional items. For example,

```
input declaration ::=
    input [ range ] list_of_port_identifiers ;
```

- Braces enclose an item that you can specify zero or more times. For example,

```
list_of_ports ::=
    ( port { , port } )
```

- Code examples appear in constant-width font.

```
/* This is an example of the font used for code.*/
```

- Within the text, variables are in italic font, like this: *allowed_errors*.

- Keywords, filenames, names of natures, and names of disciplines appear in constant-width font, like this:

```
keyword
file_name
name_of_nature
name_of_discipline
```

- If a statement is too long to fit on one line, the remainder of the statement appears indented on the next line, like this:

```
qgf = width*length*cfbb*(vgfs - wkf - qb/(2*cbb) -
    (vgbs - vfbb + qb/(2*cob))) + qgf_par ;
```

Data Type Prefixes for SKILL Arguments

The Cadence SKILL language supports several data types to identify the type of value you can assign to an argument. You can determine the data type by the single letter (followed by an underscore) that appears as a prefix on an argument name as follows:

Prefix	Internal Name	Data Type
`a`	array	array
`b`	ddUserType	DDPI Object
`C`	opfcontext	OPF context
`d`	dbobject	Open Access database (OA)
`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
`l`	list	linked list
`m`	nmpIlUserType	nmpII 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
`t`	string	character string (text)

Prefix	Internal Name	Data Type
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

For example, `t` is the data type in *`t_viewNames`*.

For example, `t` is the data type in *`t_viewNames`*.

For information on the SKILL language, see the *Cadence SKILL Language User Guide*.

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 Virtuoso AMS Designer Environment:

- Virtuoso Schematic Editor
- Virtuoso Analog Design Environment
- Analog Modeling with Verilog-A
- Behavioral Modeling with Verilog-AMS
- Real Modeling with Verilog-AMS

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 Getting Started with AMS Designer

The Virtuoso® AMS Designer environment and simulator work together so that you can set up your simulation, netlist, compile, elaborate, and simulate designs that contain analog, digital, and mixed-signal components. You can run simulations in batch mode or interactively using the SimVision debugger. When the simulation is complete, you can view waveforms using SimVision or the Virtuoso Visualization and Analysis program. The AMS Designer simulator lets you switch between using the Spectre, UltraSim and the APS solvers. See also Chapter 8, "Netlisting" for information about the two netlisters AMS Designer offers.

In this tutorial, you use the AMS Designer environment, the Spectre and UltraSim simulator solvers, SimVision, and (separately) the `amsdesigner` command to netlist, compile, elaborate, and simulate the `aeq_ac_sim` schematic which contains analog, digital, and mixed-signal components.

See the following topics for more information:

- Setting Up the Tutorial on page 30
- Running the Tutorial in the AMS Designer Environment on page 31
- Running the Tutorial Using the UltraSim Analog Solver on page 63
- Running the Tutorial from the Command Line on page 68

Setting Up the Tutorial

To set up files and libraries for this tutorial, do the following:

1. Copy the tutorial files to your local working directory. For example:

   ```
   mkdir myAMS
   cd myAMS
   cp -r $CDSHOME/tools/dfII/samples/AMS/vfs_amsflow/* .
   ```

   Note: `$CDSHOME` represents *`your_install_dir`*, the location of your Cadence Virtuoso® software installation. You must also have an environment variable, such as `AMSHOME`, that represents the location of your Cadence NC software installation. Your path must also contain the directories of the binary executables, such as `virtuoso` and `ncsim`.

2. Decompress and install the archive file:

   ```
   gunzip vfs_amsflow.tar.gz
   tar xf vfs_amsflow.tar
   ```

3. Change to the tutorial directory:

   ```
   cd vfs_amsflow
   ```

4. **Important**: Run the `compilecms` script in the tutorial files directory to compile the custom connect modules that this design uses so they are available for later use:

   ```
   ./compilecms
   ```

   Note: In addition to compiling connect rules, this script also sets the `TUTORHOME` environment variable to the current directory:

   ```
   setenv TUTORHOME `pwd`
   ```

Running the Tutorial in the AMS Designer Environment

The tutorial design is a 6-bit, differential, flash, analog-to-digital converter (ADC) with a programmable analog equalizer filter on the front end and a 6-bit digital-to-analog converter (DAC) on the output. The output DAC reproduces the differential ADC input, providing a way to verify the behavior of the design. The hierarchy includes a mix of Verilog-A, Verilog (digital), VHDL-AMS, and schematic blocks.

In this part of the tutorial, you use the AMS Designer environment to specify simulation details, netlist, compile, elaborate, and simulate the tutorial design.

See the following topics for details:

- Opening the Schematic and Design Configuration on page 32
- Initializing AMS on page 37
- Using the Quick Setup Form on page 38
- Using the Netlist and Run Form on page 41
- Running the Simulation on page 57
- Using the SimVision Source Browser on page 60
- Using the SimVision Waveform Window on page 62

Opening the Schematic and Design Configuration

To open the top-level configuration and schematic of the tutorial design, do the following:

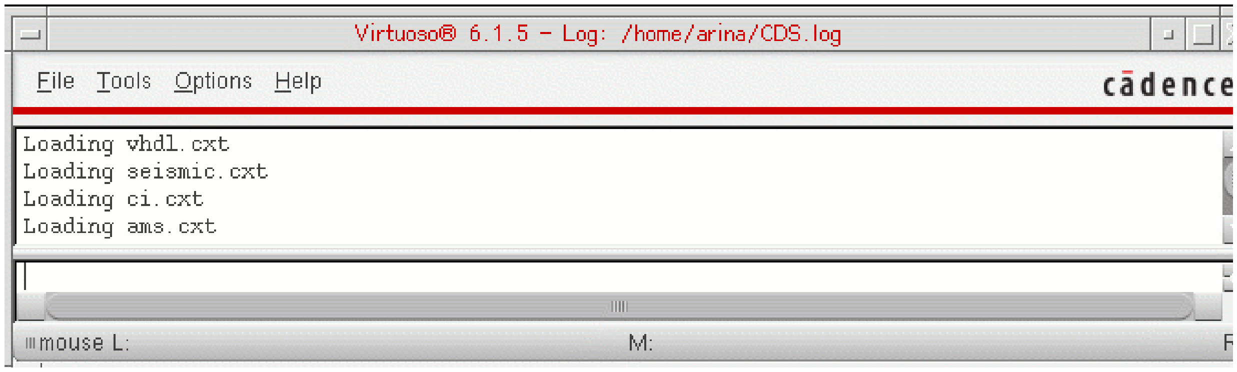
Video

Show Me

1. Type `virtuoso` at the command line.

   ```
   virtuoso &
   ```

   The CIW appears.

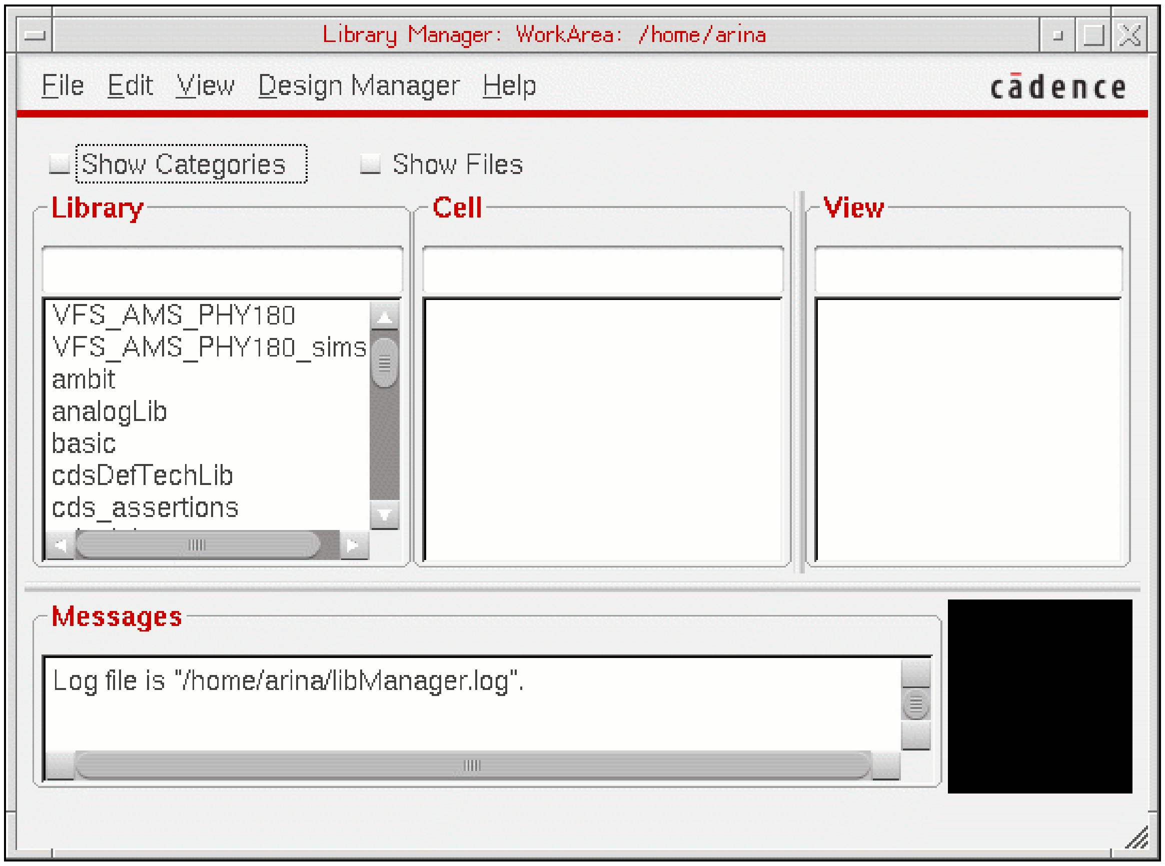


```
Loading vhdl.cxt
Loading seismic.cxt
Loading ci.cxt
Loading ams.cxt
```

   For more information on the CIW, see “Using the Command Interpreter Window” in the *Virtuoso Design Environment User Guide*.

2. In the CIW, choose *Tools – Library Manager*.

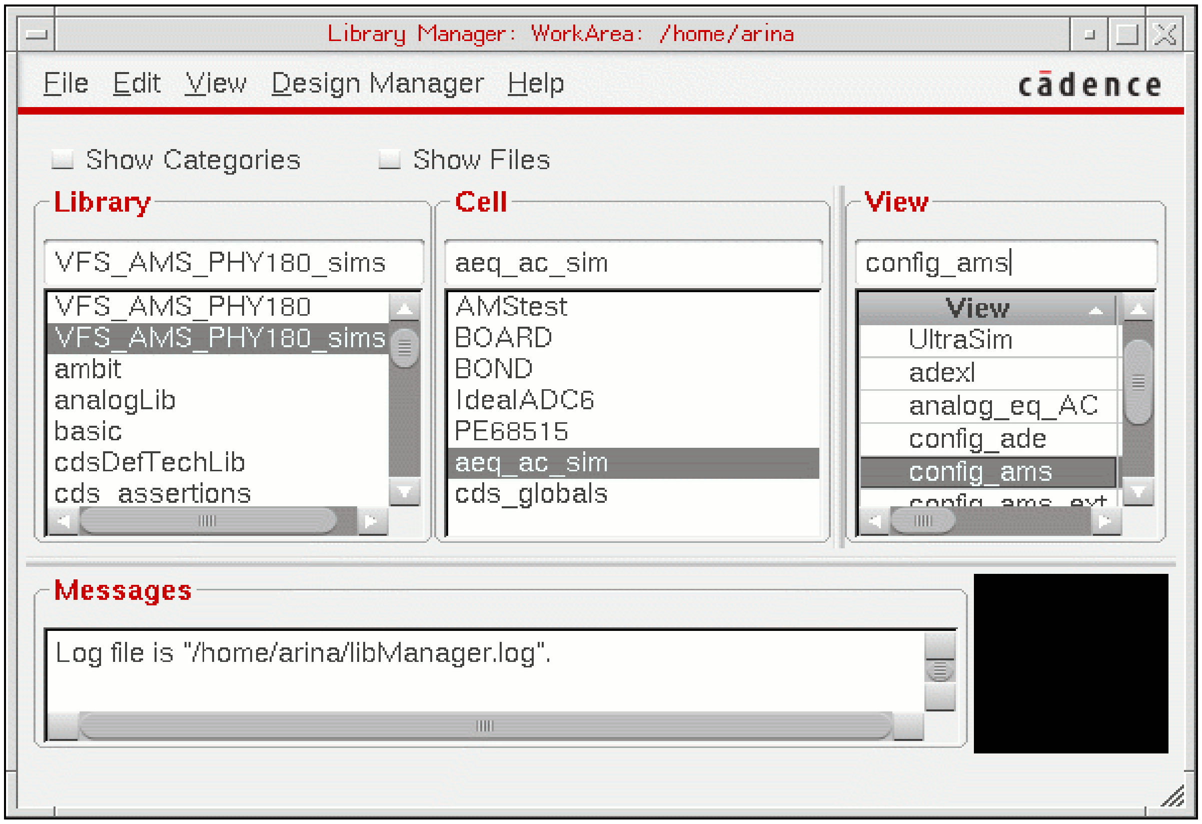
The Library Manager window appears.



Log file is "/home/arina/libManager.log".

3. In the *Library* column, click *VFS_AMS_PHY180_sims* to view the cells in that library.
4. In the *Cell* column, click *aeq_ac_sim* to show all the views in that cell.

5. In the *View* column, double-click *config_ams*.


Library Manager: WorkArea: /home/arina
File Edit View Design Manager Help
cādence
Show Categories
Show Files
Library
Cell
View
Messages

VFS_AMS_PHY180_sims

aeq_ac_sim

config_ams

VFS_AMS_PHY180
VFS_AMS_PHY180_sims
ambit
analogLib
basic
cdsDefTechLib
cds_assertions

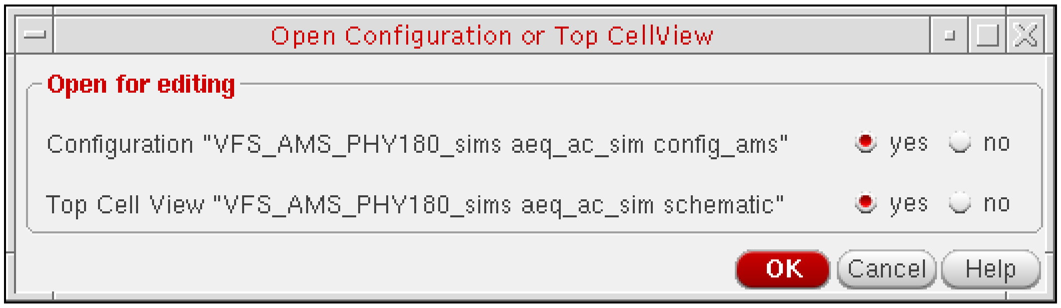
AMStest
BOARD
BOND
IdealADC6
PE68515
aeq_ac_sim
cds_globals

UltraSim
adexl
analog_eq_AC
config_ade
config_ams

```
Log file is "/home/arina/libManager.log".
```

The Open Configuration or Top CellView form appears.

6. In the *Open for editing* group box, select *yes* to open the configuration and *yes* to open the top cell view.



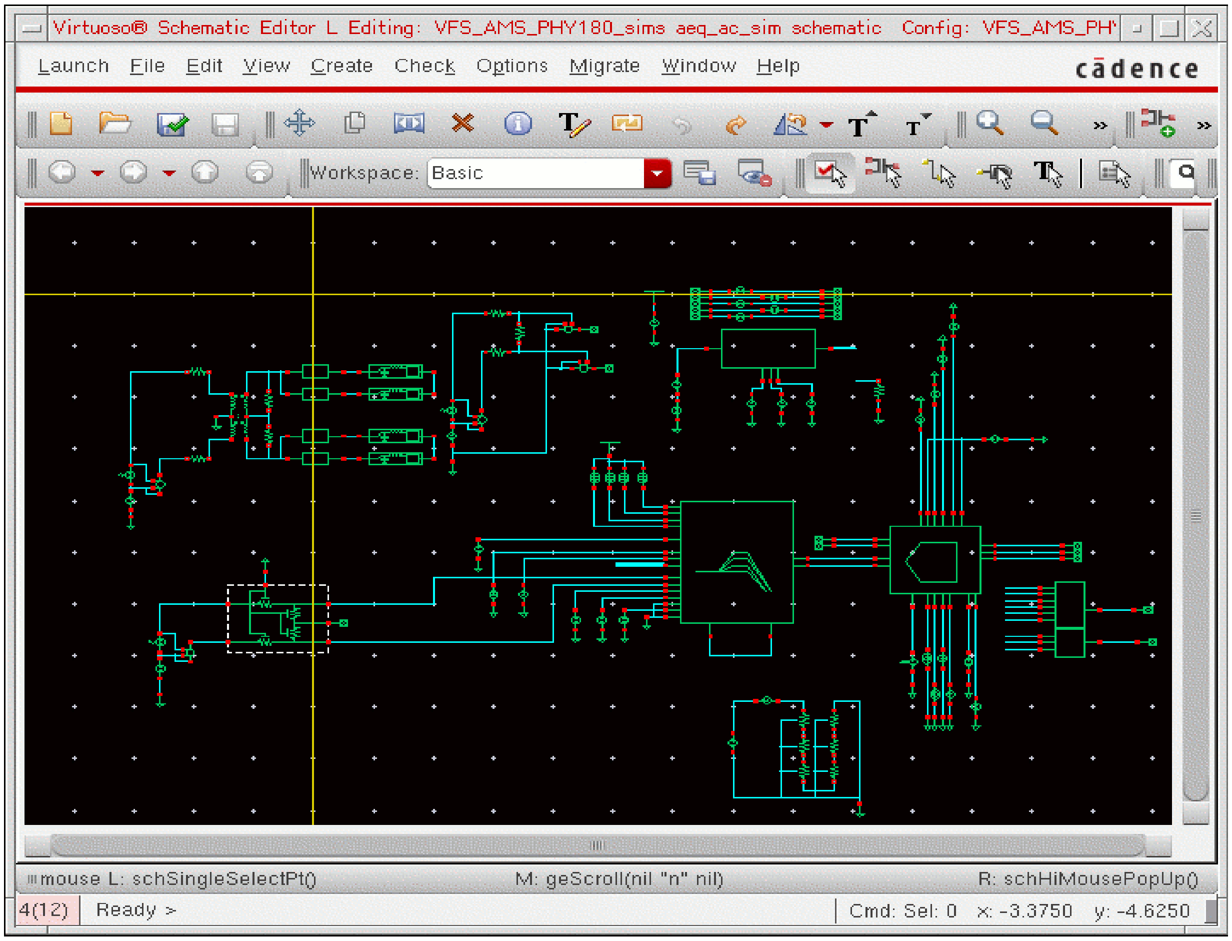
Configuration "VFS_AMS_PHY180_sims aeq_ac_sim config_ams" [✓ yes] no

Top Cell View "VFS_AMS_PHY180_sims aeq_ac_sim schematic" [✓ yes] no

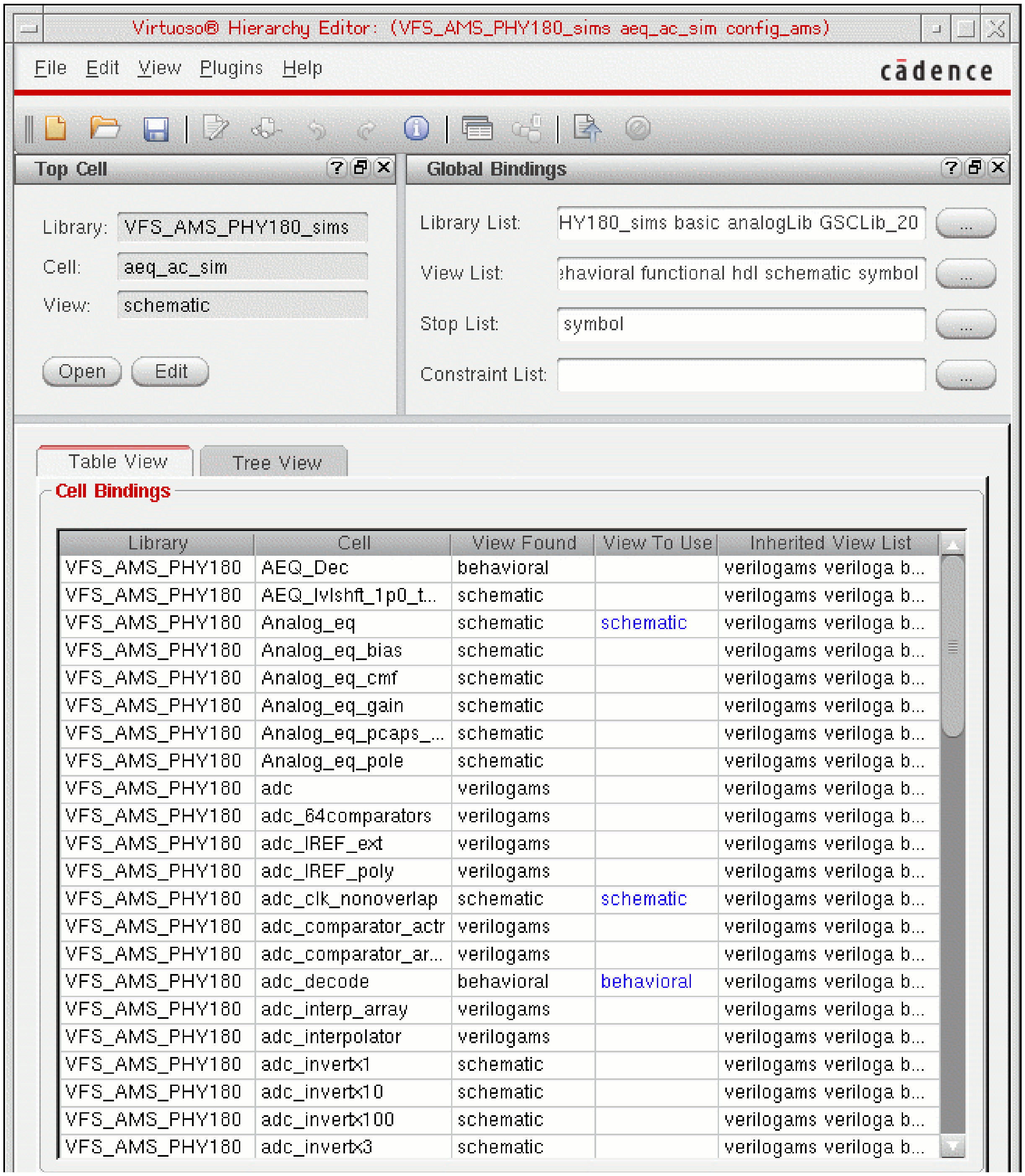
7. Click *OK*.

The *aeq_ac_sim* schematic view appears in the Virtuoso Schematic Editor and the *aeq_ac_sim* config view appears in the Virtuoso® Hierarchy Editor.

The *aeq_ac_sim* schematic looks like this:


Virtuoso® Schematic Editor L Editing: VFS_AMS_PHY180_sims aeq_ac_sim schematic Config: VFS_AMS_PH
Launch File Edit View Create Check Options Migrate Window Help
cādence
Workspace: Basic
mouse L: schSingleSelectPt()
M: geScroll(nil "n" nil)
R: schHiMousePopUp()
4(12)
Ready >
Cmd: Sel: 0
x: -3.3750
y: -4.6250

The *aeq_ac_sim* config view data looks like this:


Virtuoso® Hierarchy Editor: (VFS_AMS_PHY180_sims aeq_ac_sim config_ams)
File Edit View Plugins Help
cādence
Top Cell
Global Bindings
Open
Edit
Table View
Tree View
Cell Bindings

Library: VFS_AMS_PHY180_sims
Cell: aeq_ac_sim
View: schematic

Library List: HY180_sims basic analogLib GSCLib_20
View List: ehavioral functional hdl schematic symbol
Stop List: symbol
Constraint List:

Library	Cell	View Found	View To Use	Inherited View List
VFS_AMS_PHY180	AEQ_Dec	behavioral		verilogams veriloga b...
VFS_AMS_PHY180	AEQ_lvlshft_1p0_t...	schematic		verilogams veriloga b...
VFS_AMS_PHY180	Analog_eq	schematic	schematic	verilogams veriloga b...
VFS_AMS_PHY180	Analog_eq_bias	schematic		verilogams veriloga b...
VFS_AMS_PHY180	Analog_eq_cmf	schematic		verilogams veriloga b...
VFS_AMS_PHY180	Analog_eq_gain	schematic		verilogams veriloga b...
VFS_AMS_PHY180	Analog_eq_pcaps_...	schematic		verilogams veriloga b...
VFS_AMS_PHY180	Analog_eq_pole	schematic		verilogams veriloga b...
VFS_AMS_PHY180	adc	verilogams		verilogams veriloga b...
VFS_AMS_PHY180	adc_64comparators	verilogams		verilogams veriloga b...
VFS_AMS_PHY180	adc_IREF_ext	verilogams		verilogams veriloga b...
VFS_AMS_PHY180	adc_IREF_poly	verilogams		verilogams veriloga b...
VFS_AMS_PHY180	adc_clk_nonoverlap	schematic	schematic	verilogams veriloga b...
VFS_AMS_PHY180	adc_comparator_actr	verilogams		verilogams veriloga b...
VFS_AMS_PHY180	adc_comparator_ar...	verilogams		verilogams veriloga b...
VFS_AMS_PHY180	adc_decode	behavioral	behavioral	verilogams veriloga b...
VFS_AMS_PHY180	adc_interp_array	verilogams		verilogams veriloga b...
VFS_AMS_PHY180	adc_interpolator	verilogams		verilogams veriloga b...
VFS_AMS_PHY180	adc_invertx1	schematic		verilogams veriloga b...
VFS_AMS_PHY180	adc_invertx10	schematic		verilogams veriloga b...
VFS_AMS_PHY180	adc_invertx100	schematic		verilogams veriloga b...
VFS_AMS_PHY180	adc_invertx3	schematic		verilogams veriloga b...

Initializing AMS

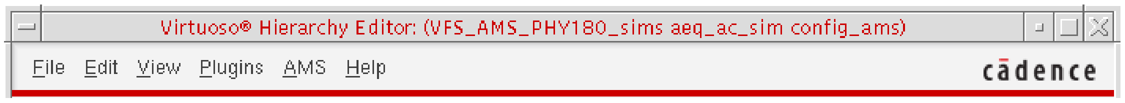
To install the *AMS* menu, specify a run directory, and specify the location of the `hdl.var` file that AMS Designer uses, do the following:

Video

Show Me

1. In the Virtuoso® Hierarchy Editor, choose *Plug-Ins – AMS*.

   AMS appears on the menu bar.


Virtuoso® Hierarchy Editor: (VFS_AMS_PHY180_sims aeq_ac_sim config_ams)
File Edit View Plugins AMS Help
cādence

   The *AMS* menu contains controls for the AMS Designer environment and simulator. When the *AMS* menu first appears, the only item you can select is *Initialize*.

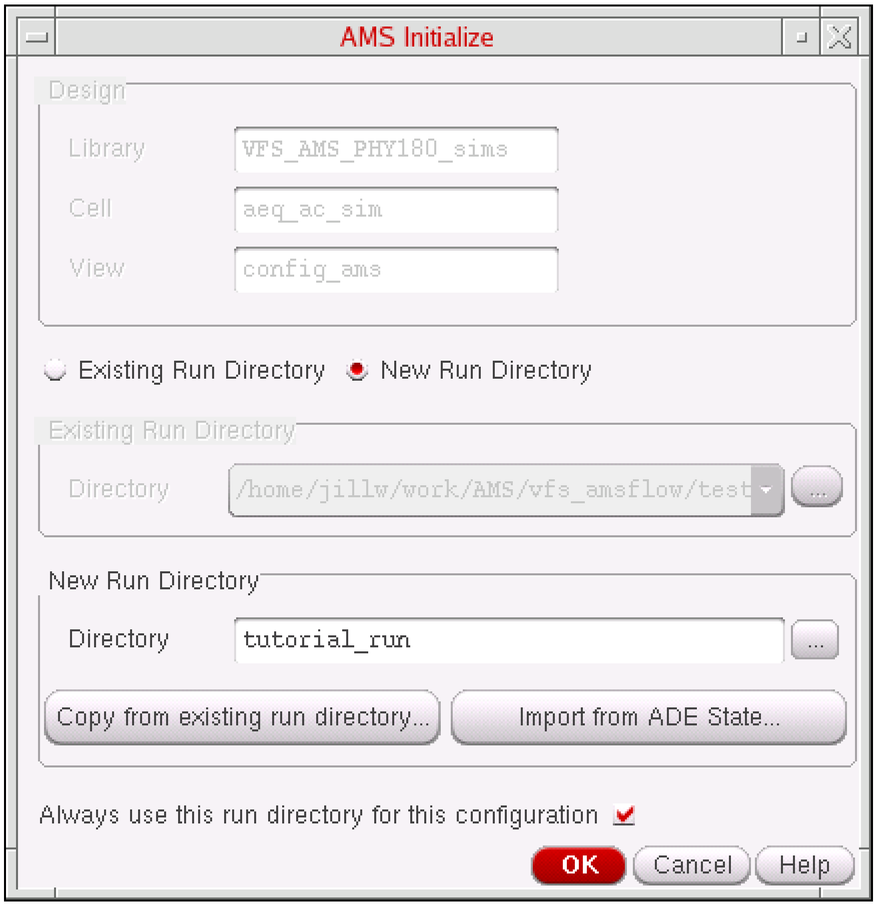
2. Choose *AMS – Initialize*.

   The AMS Initialize form appears.

3. Select *New Run Directory*.

   The *New Run Directory* group box becomes active and the *Existing Run Directory* group box becomes inactive.

4. In the *Directory* field in the *New Run Directory* group box, type `tutorial_run`.


AMS Initialize

Design
Library VFS_AMS_PHY180_sims
Cell aeq_ac_sim
View config_ams

Existing Run Directory [✓ New Run Directory]

Existing Run Directory
Directory /home/jillw/work/AMS/vfs_amsflow/test

New Run Directory
Directory tutorial_run

Copy from existing run directory... Import from ADE State...

Always use this run directory for this configuration ✓

OK Cancel Help

5. (Optional) Mark the *Always use this run directory for this configuration* check box.
6. Click *OK*.

   Note: Other items on the *AMS* menu become active.

Using the Quick Setup Form

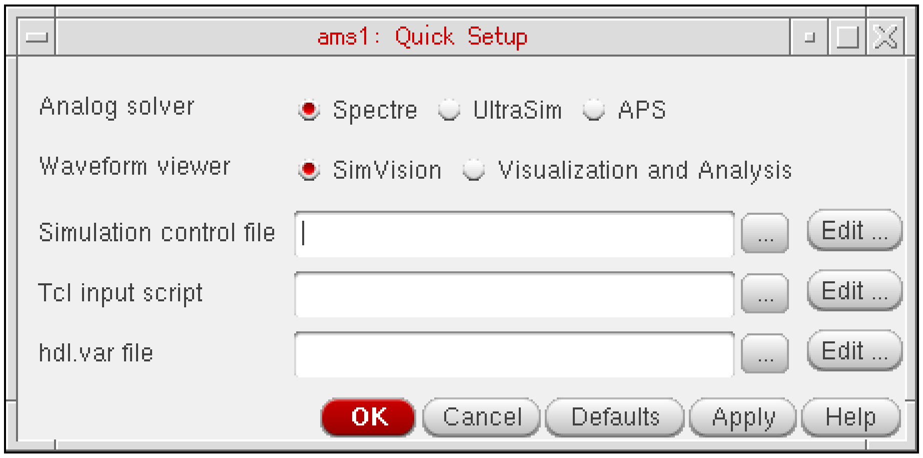
You can use the Quick Setup form for AMS if you already have files that specify your simulation setup information, such as a simulation control file, a Tcl input script, or an `hdl.var` file. In the case of this tutorial, we have a Tcl script (`demo.tcl`) and an `hdl.var` file containing setup information. To specify these files, do the following:

Video

Show Me

1. In the Virtuoso® Hierarchy Editor, choose *AMS – Quick Setup*.

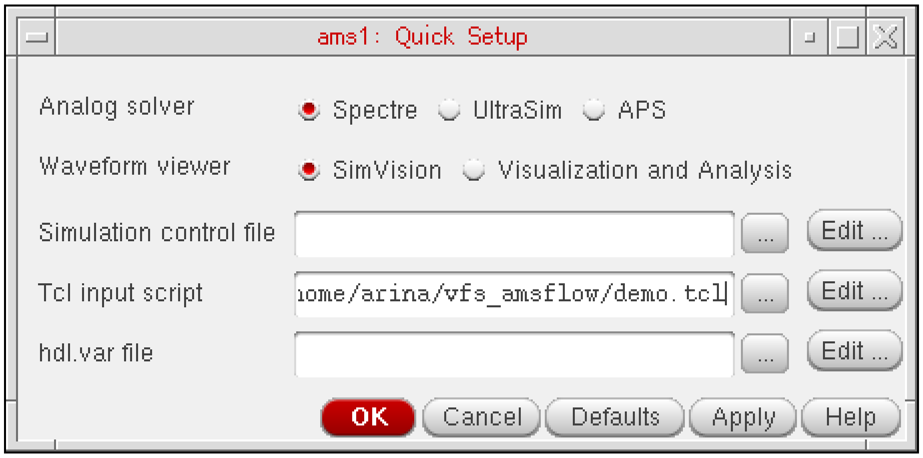
   The Quick Setup form appears.



Analog solver: [✓ Spectre] UltraSim APS
Waveform viewer: [✓ SimVision] Visualization and Analysis
Simulation control file:
Tcl input script:
hdl.var file:

2. To the right of the *Tcl input script* field, click the browse button.
3. On the Choose form that appears, double-click `demo.tcl` in the `vfs_amsflow` directory.

   The full path to `demo.tcl` appears in the *Tcl input script* field on the Quick Setup form.



Analog solver: [✓ Spectre] UltraSim APS
Waveform viewer: [✓ SimVision] Visualization and Analysis
Simulation control file:
Tcl input script: ıome/arina/vfs_amsflow/demo.tcl
hdl.var file:

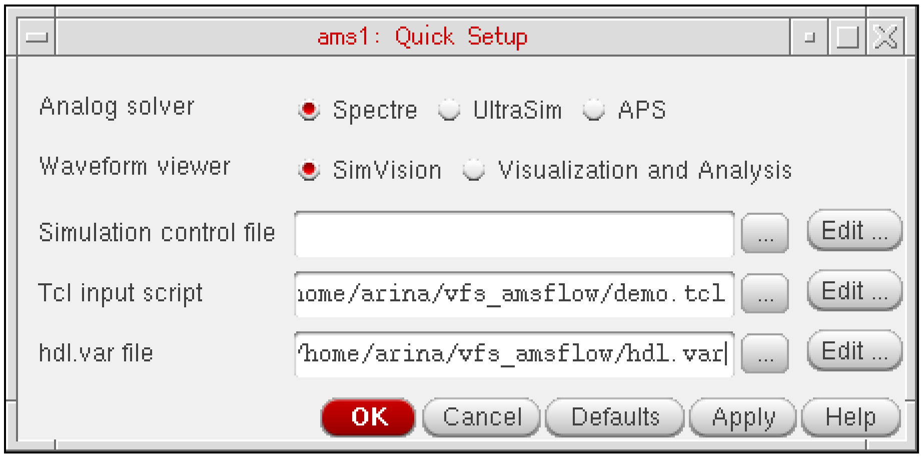
   This Tcl script contains the following commands:

```
set display_unit NS
set time_unit module
alias tp run -timepoint
alias . run
alias quit exit
stop -time 50ns -absolute
```

4. To the right of the *hdl.var file* field on the Quick Setup form, click the browse button.

5. On the Choose form that appears, double-click `hdl.var` in the `vfs_amsflow` directory.

This `hdl.var` file appears in the *hdl.var file* field on the Quick Setup form.



Analog solver	[✓ Spectre] UltraSim APS
Waveform viewer	[✓ SimVision] Visualization and Analysis
Simulation control file	
Tcl input script	ıome/arina/vfs_amsflow/demo.tcl
hdl.var file	'home/arina/vfs_amsflow/hdl.var

This `hdl.var` file contains the following definitions:

```
define ncuse5x
define ncvlogopts -linedebug -mess -define SPEEDUP -nowarn MACNDF -nowarn
RECOMP
define ncelabopts -nowarn CUVWSP -nowarn CUNGL1 -nowarn CSINFI -nowarn CUVUKP
-nowarn CUSRCH -nowarn SYWARN
define ncvhdlopts -linedebug -mess -v93
```

6. Click *OK*.

Using the Netlist and Run Form

You can use the Netlist and Run form to specify the following details prior to running the AMS Designer simulator with the Spectre solver and using the SimVision waveform viewer:

- Transient stop time
- Model file for simuation
- Verilog-AMS include file search path
- Outputs you want to save and plot
- Connect rules
- Netlist and run options

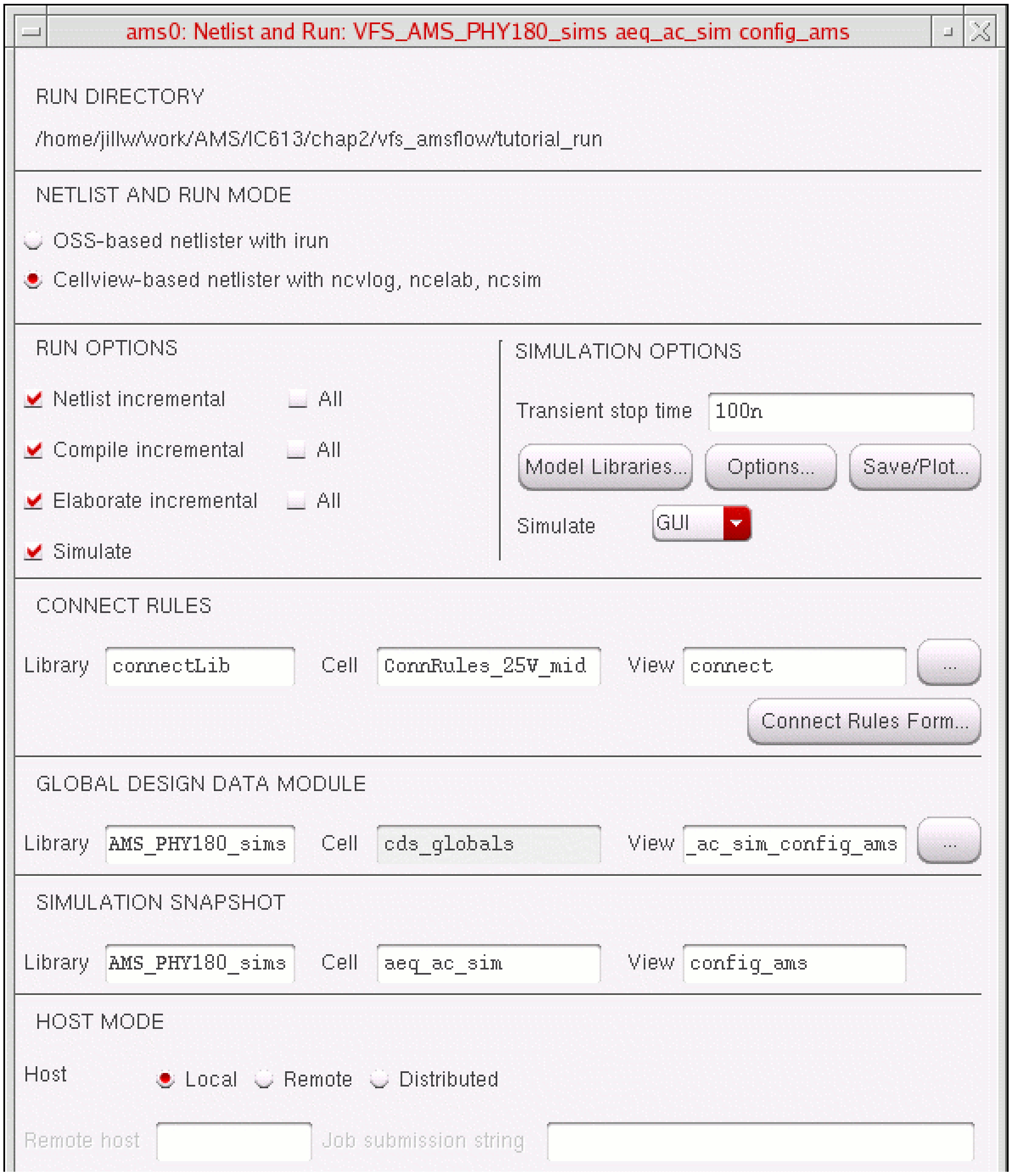
Note: You can also specify these most of these details using the *AMS – Detailed Setup* pull-right menu choices as follows:

Transient stop time	Choose *AMS – Detailed Setup – Analyses*
Model file for simulation	Choose *AMS – Detailed Setup – Model Libraries*
Verilog-AMS include file	Choose *AMS – Detailed Setup – AMS Options*
Outputs to save and plot	Choose *AMS – Detailed Setup – Save/Plot Outputs*
Connect rules	Choose *AMS – Detailed Setup – Connect Rules*

To open the Netlist and Run form, do the following:

➤ In the Virtuoso® Hierarchy Editor, choose *AMS – Netlist and Run*.

The Netlist and Run form appears.



ams0: Netlist and Run: VFS_AMS_PHY180_sims aeq_ac_sim config_ams

RUN DIRECTORY

/home/jillw/work/AMS/IC613/chap2/vfs_amsflow/tutorial_run

NETLIST AND RUN MODE

- OSS-based netlister with irun
- [✓ Cellview-based netlister with ncvlog, ncelab, ncsim]

RUN OPTIONS

- [✓ Netlist incremental] All
- [✓ Compile incremental] All
- [✓ Elaborate incremental] All
- [✓ Simulate]

SIMULATION OPTIONS

Transient stop time: 100n

Model Libraries... | Options... | Save/Plot...

Simulate: GUI

CONNECT RULES

Library: connectLib Cell: ConnRules_25V_mid View: connect

Connect Rules Form...

GLOBAL DESIGN DATA MODULE

Library: AMS_PHY180_sims Cell: cds_globals View: _ac_sim_config_ams

SIMULATION SNAPSHOT

Library: AMS_PHY180_sims Cell: aeq_ac_sim View: config_ams

HOST MODE

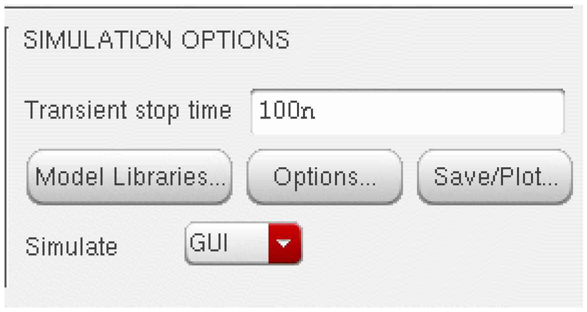
Host: [✓ Local] Remote Distributed

Remote host Job submission string

Specifying the Transient Stop Time

To specify the transient stop time for the analog solver, do the following:

➤ On the Netlist and Run form, in the *SIMULATION OPTIONS* group box, in the *Transient stop time* field, type `100n`.


SIMULATION OPTIONS

Transient stop time: 100n

Simulate: GUI

Note: You can specify additional analysis settings by choosing *AMS – Detailed Setup*. For details, see Chapter 4, "Using the Detailed Setup Menu."

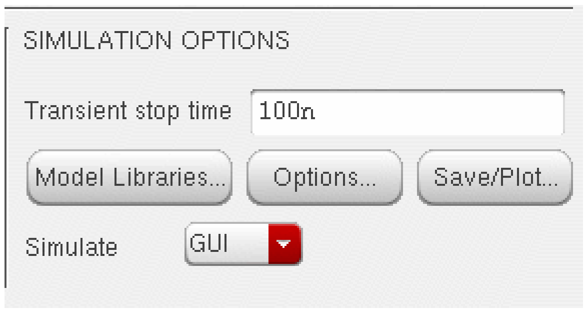
Specifying the Simulation Model File

To specify the simulation model file, do the following:

Video

Show Me

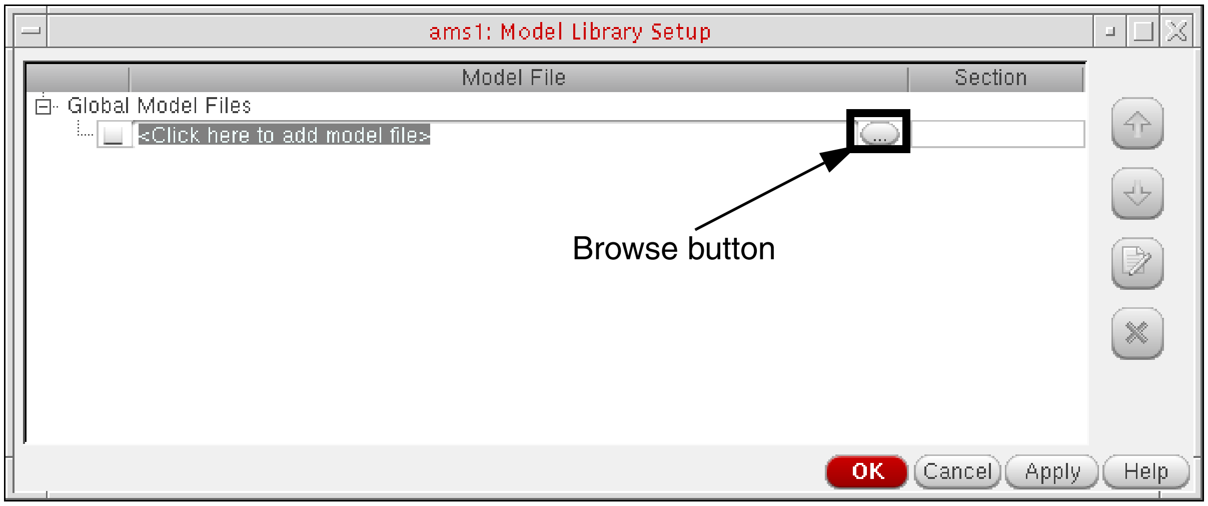
1. On the Netlist and Run form, in the *SIMULATION OPTIONS* group box, click *Model Libraries*.


SIMULATION OPTIONS

Transient stop time: 100n

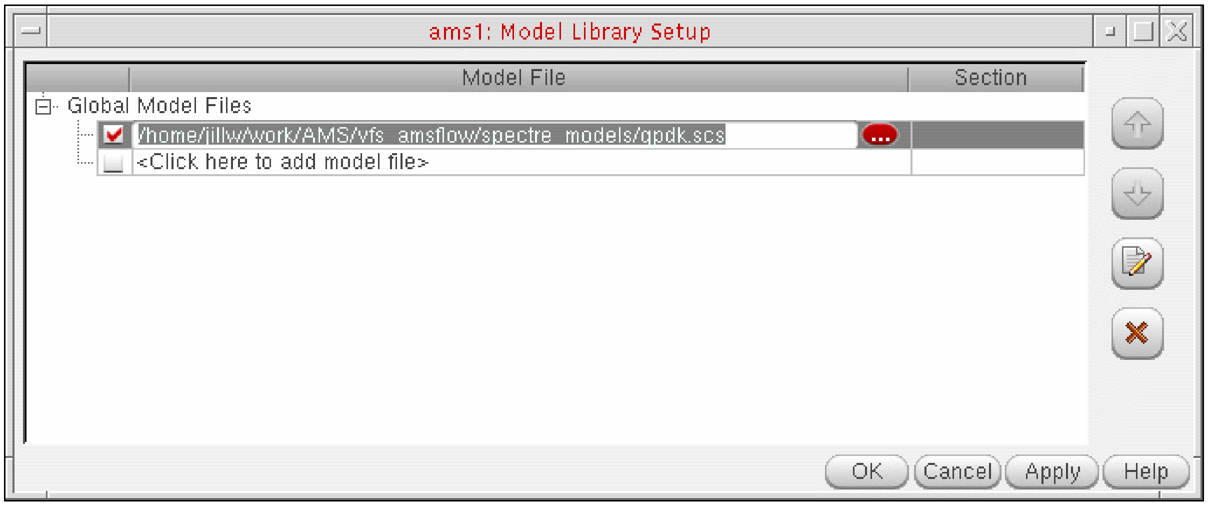
Simulate: GUI

The Model Library Setup form appears.



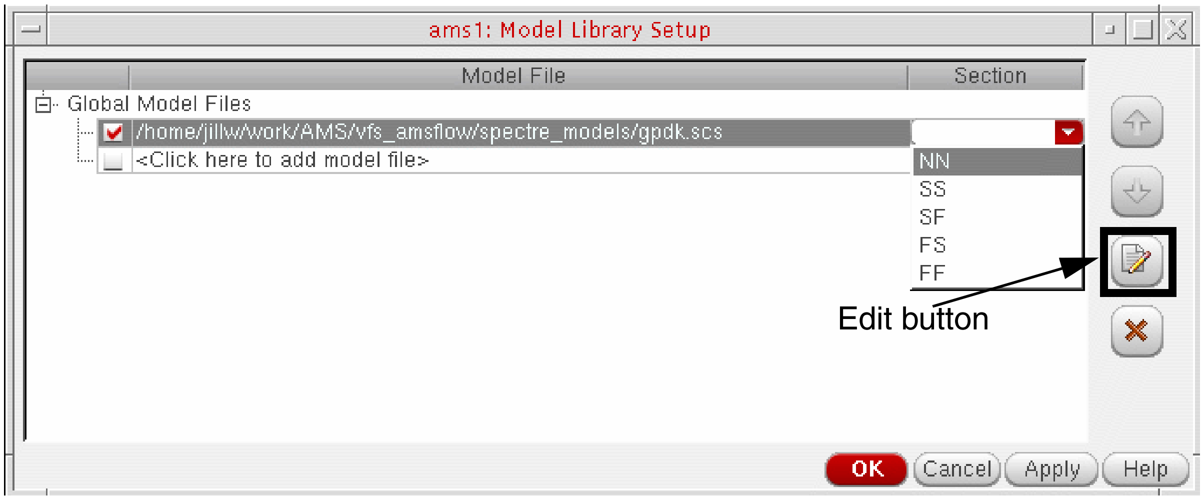
2. Click the browse button.
3. On the Choose form that appears, navigate to and double-click the `vfs_amsflow/spectre_models/gpdk.scs` file.

   This model file appears in the *Global Model Files* tree on the Model Library Setup form.


ams1: Model Library Setup
Model File
Section
Global Model Files
/home/jillw/work/AMS/vfs_amsflow/spectre_models/gpdk.scs
<Click here to add model file>
OK
Cancel
Apply
Help

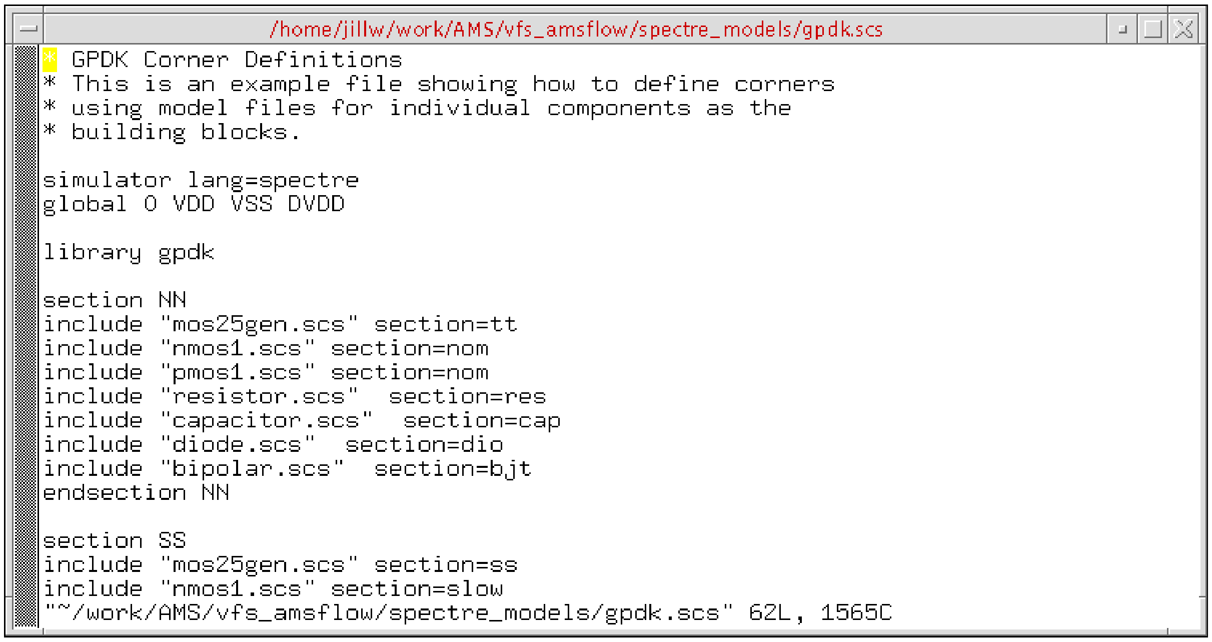
4. In the drop-down combo box in the *Section* column, select `NN`.

NN specifies the model designed to represent nominal operating conditions.


ams1: Model Library Setup
Model File
Section
Global Model Files
/home/jillw/work/AMS/vfs_amsflow/spectre_models/gpdk.scs
<Click here to add model file>
NN
SS
SF
FS
FF
Edit button
OK
Cancel
Apply
Help

5. (Optional) Click the edit button.

The model file appears in an editing window so you can see which files will contribute to the characterization of the various operating conditions.



```
* GPDK Corner Definitions
* This is an example file showing how to define corners
* using model files for individual components as the
* building blocks.

simulator lang=spectre
global 0 VDD VSS DVDD

library gpdk

section NN
include "mos25gen.scs" section=tt
include "nmos1.scs" section=nom
include "pmos1.scs" section=nom
include "resistor.scs"  section=res
include "capacitor.scs"  section=cap
include "diode.scs"  section=dio
include "bipolar.scs"  section=bjt
endsection NN

section SS
include "mos25gen.scs" section=ss
include "nmos1.scs" section=slow
"~/work/AMS/vfs_amsflow/spectre_models/gpdk.scs" 62L, 1565C
```

When you are done looking at the model file, close the window without saving.

6. On the Model Library Setup form, click *OK*.

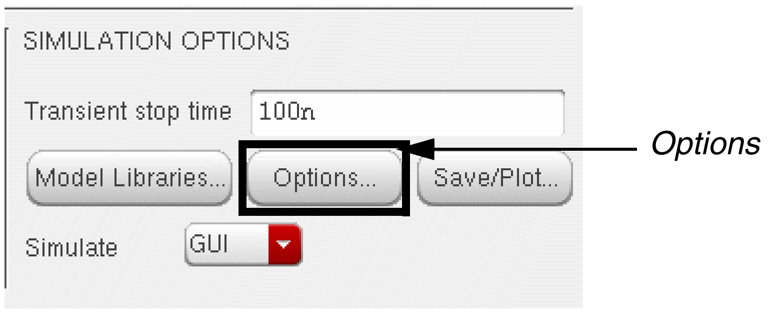
Specifying the Search Path for Included Verilog-AMS Files

To specify the path to search for Verilog-AMS include files declared in `` `include `` statements, do the following:

Video

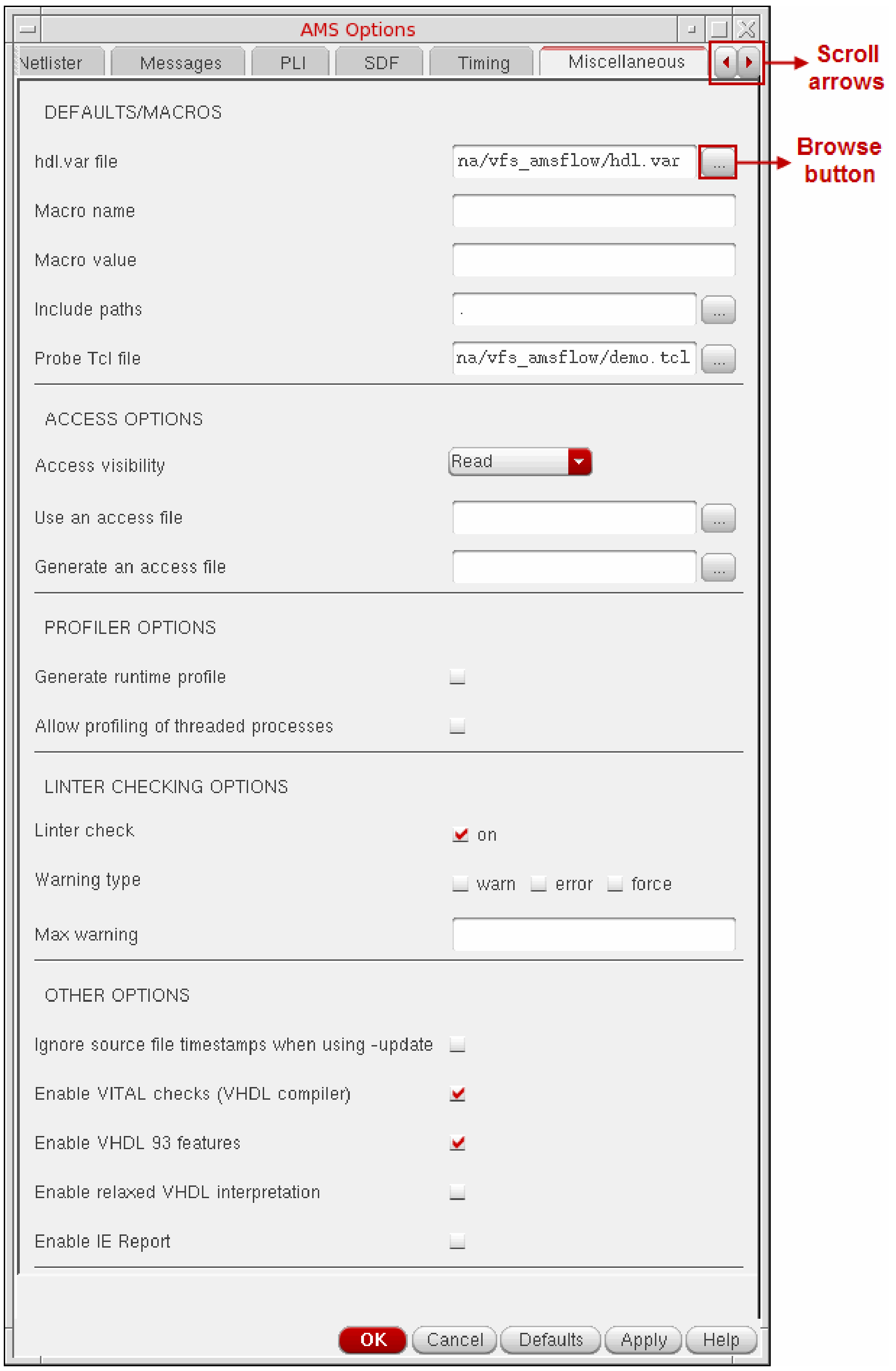
Show Me

1. On the Netlist and Run form, in the *SIMULATION OPTIONS* group box, click *Options*.



The AMS Options form appears.

2. Select the *Misc* tab.



DEFAULTS/MACROS

hdl.var file	na/vfs_amsflow/hdl.var
Macro name	
Macro value	
Include paths	.
Probe Tcl file	na/vfs_amsflow/demo.tcl

ACCESS OPTIONS

Access visibility	Read
Use an access file	
Generate an access file	

PROFILER OPTIONS

Generate runtime profile	☐
Allow profiling of threaded processes	☐

LINTER CHECKING OPTIONS

Linter check	✓ on
Warning type	☐ warn ☐ error ☐ force
Max warning	

OTHER OPTIONS

Ignore source file timestamps when using -update	☐
Enable VITAL checks (VHDL compiler)	✓
Enable VHDL 93 features	✓
Enable relaxed VHDL interpretation	☐
Enable IE Report	☐

Note: You might need to resize the form window or use the arrows at the top right of the form to scroll to the *Misc* tab.

3. To the right of the *Include path* field, click the browse button.

   The Choose a Directory form appears.

4. Select the `myfunctions` subdirectory and click *Open*.

   This subdirectory contains the `myfunctions.vams` file.

5. Click *OK*.

   The program uses this path to search for include files declared in `` `include`` statements, such as `` `include myfunctions.vams``.

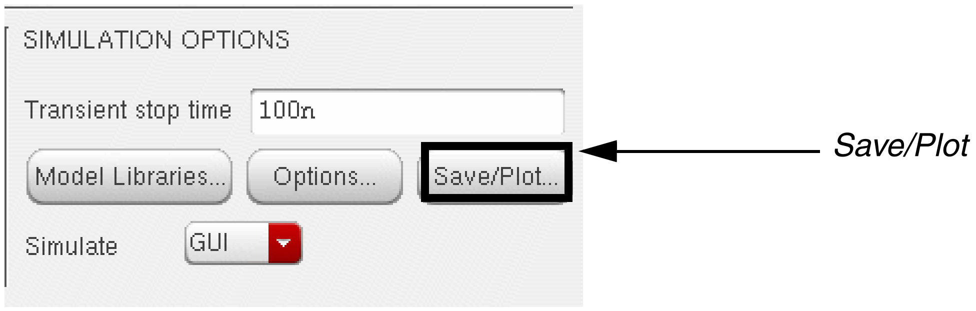
Specifying Values to Save and Plot

To specify the information you want to save and plot, do the following:

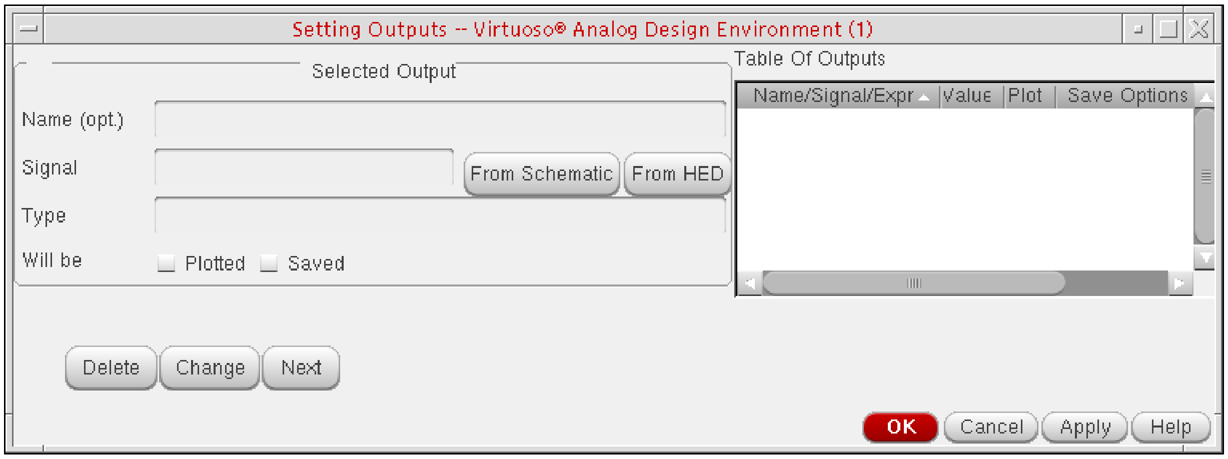
Video

Show Me

1. On the Netlist and Run form, in the *SIMULATION OPTIONS* group box, click *Save/Plot*.



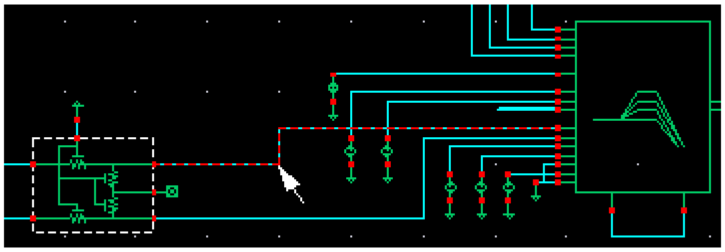
The Setting Outputs form appears.



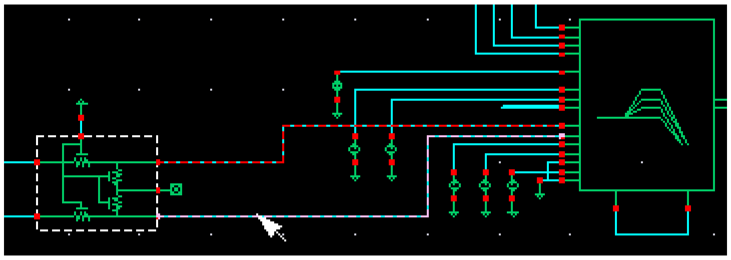
2. To select nets from the schematic, click *From Schematic*.

   The Virtuoso Schematic Editor window moves to the foreground. The status message at the bottom of the window indicates that you can select objects to be saved and plotted.

3. In the Schematic Editor window, select the `InP` net:

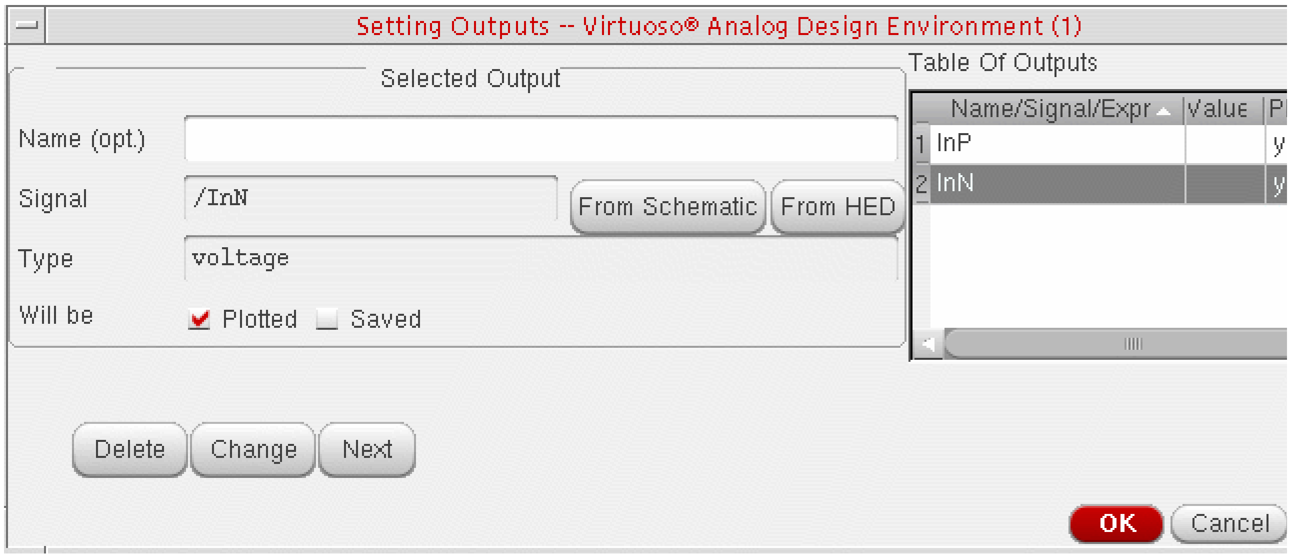


   and the `InN` net:



4. Press `Escape` to end selection mode.

`InP` and `InN` appear in the *Table Of Outputs* on the Setting Outputs form.



Signal: /InN
Type: voltage
Will be: ✓ Plotted ☐ Saved

Name/Signal/Expr	Value	Pl
1 InP		y
2 InN		y

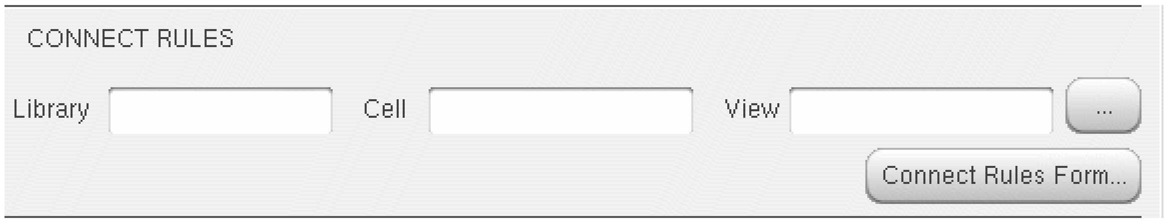
5. Click *OK*.

The signals you selected will appear in SimVision as the simulation runs.

Specifying User-Defined Connect Rules

To specify the user-defined connect rules that you compiled earlier by running the `compilecms` script, do the following on the Netlist and Run form:

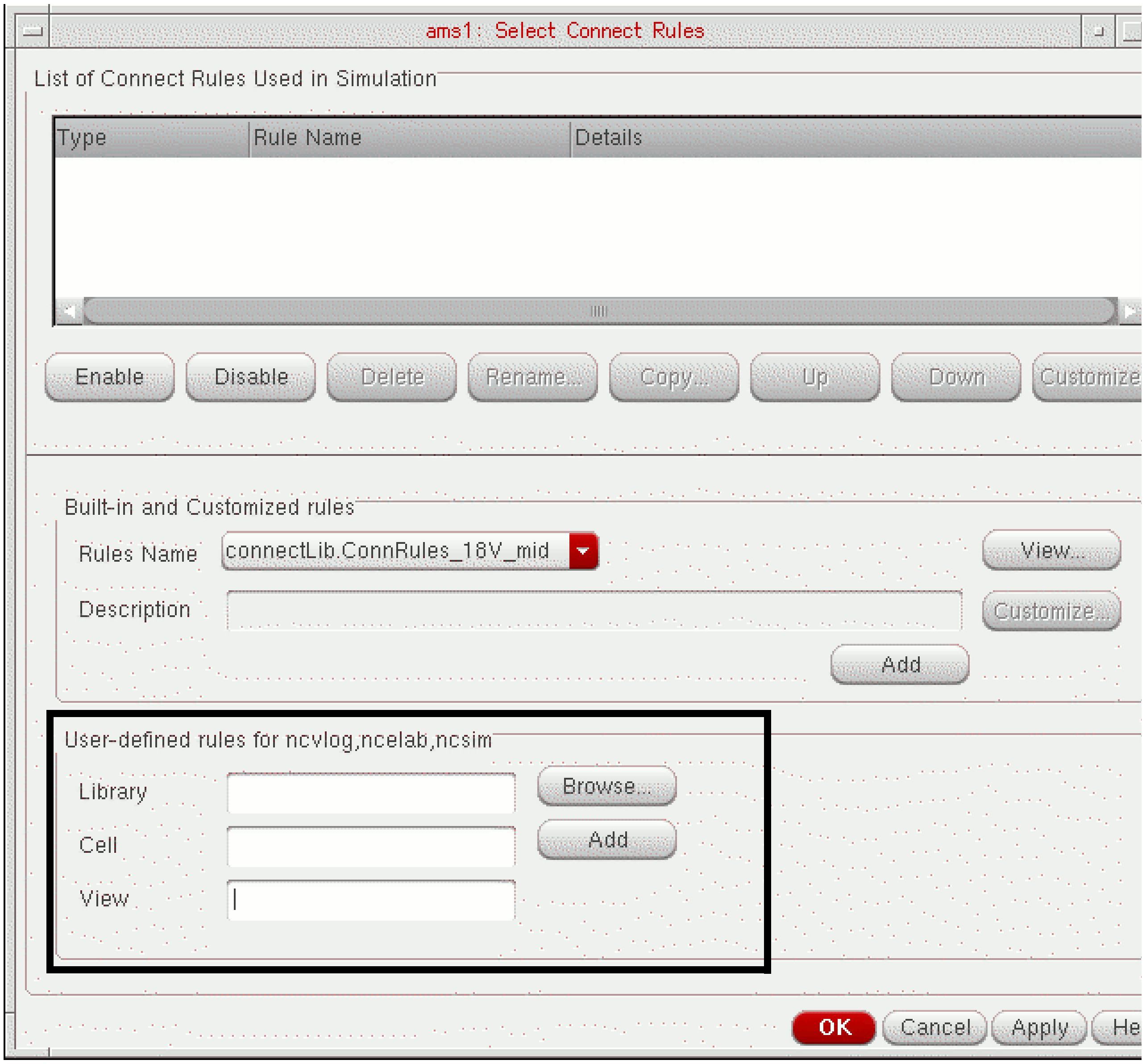
1. On the Netlist and Run form, in the *CONNECT RULES* group box, click *Connect Rules Form*.



The Select Connect Rules form appears.

2. Beneath *List of Connect Rules Used in Simulation*, select *User-defined(ncvlog,ncelab,ncsim)*.

The *User-defined rules for ncvlog,ncelab,ncsim* group box becomes active.

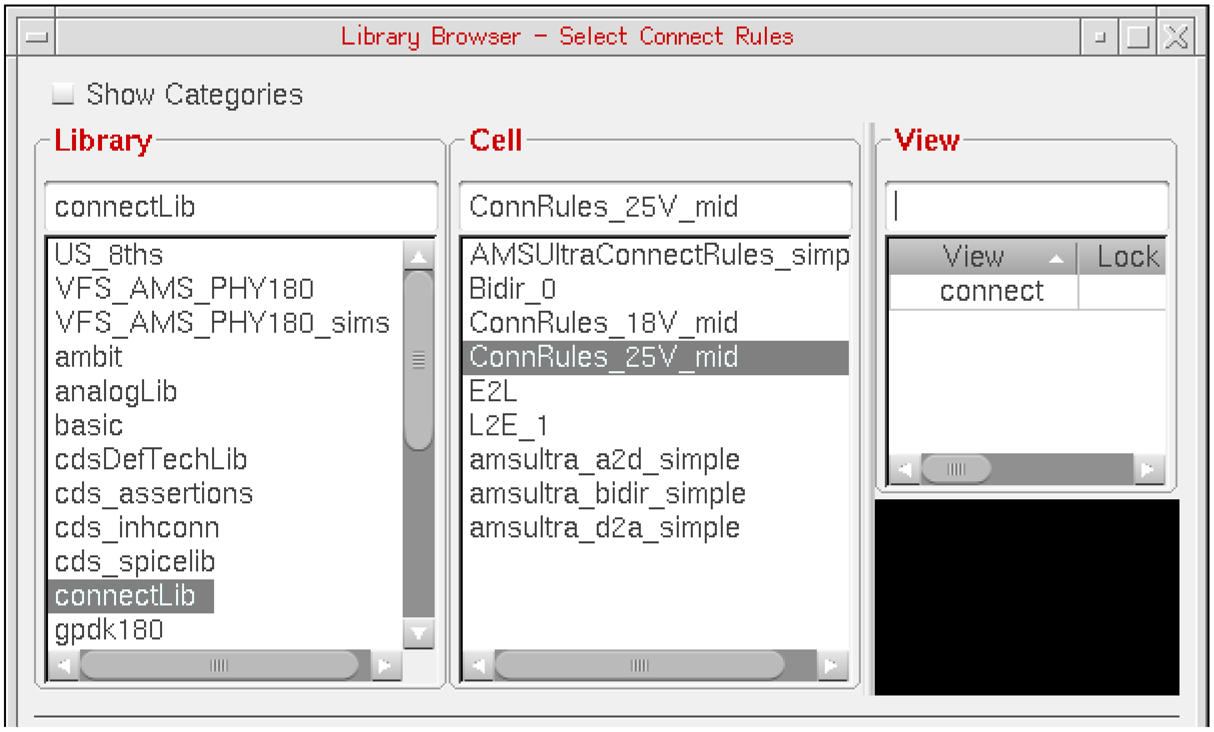


Rules Name: connectLib.ConnRules_18V_mid

3. Click *Browse*.

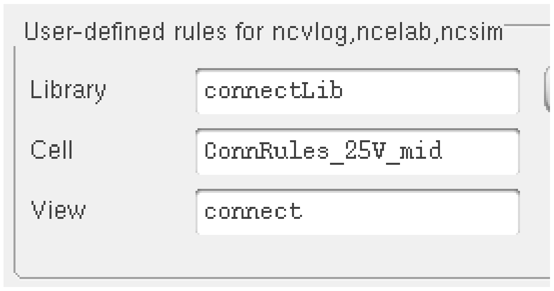
   The Library Browser – Select Connect Rules window appears.

4. In the *Library*, *Cell*, and *View* columns, select *connectLib*, *ConnRules_25V_mid*, *connect*.



Library	Cell	View
connectLib	ConnRules_25V_mid	
US_8ths	AMSUltraConnectRules_simp	connect
VFS_AMS_PHY180	Bidir_0	
VFS_AMS_PHY180_sims	ConnRules_18V_mid	
ambit	ConnRules_25V_mid	
analogLib	E2L	
basic	L2E_1	
cdsDefTechLib	amsultra_a2d_simple	
cds_assertions	amsultra_bidir_simple	
cds_inhconn	amsultra_d2a_simple	
cds_spicelib		
connectLib		
gpdk180		

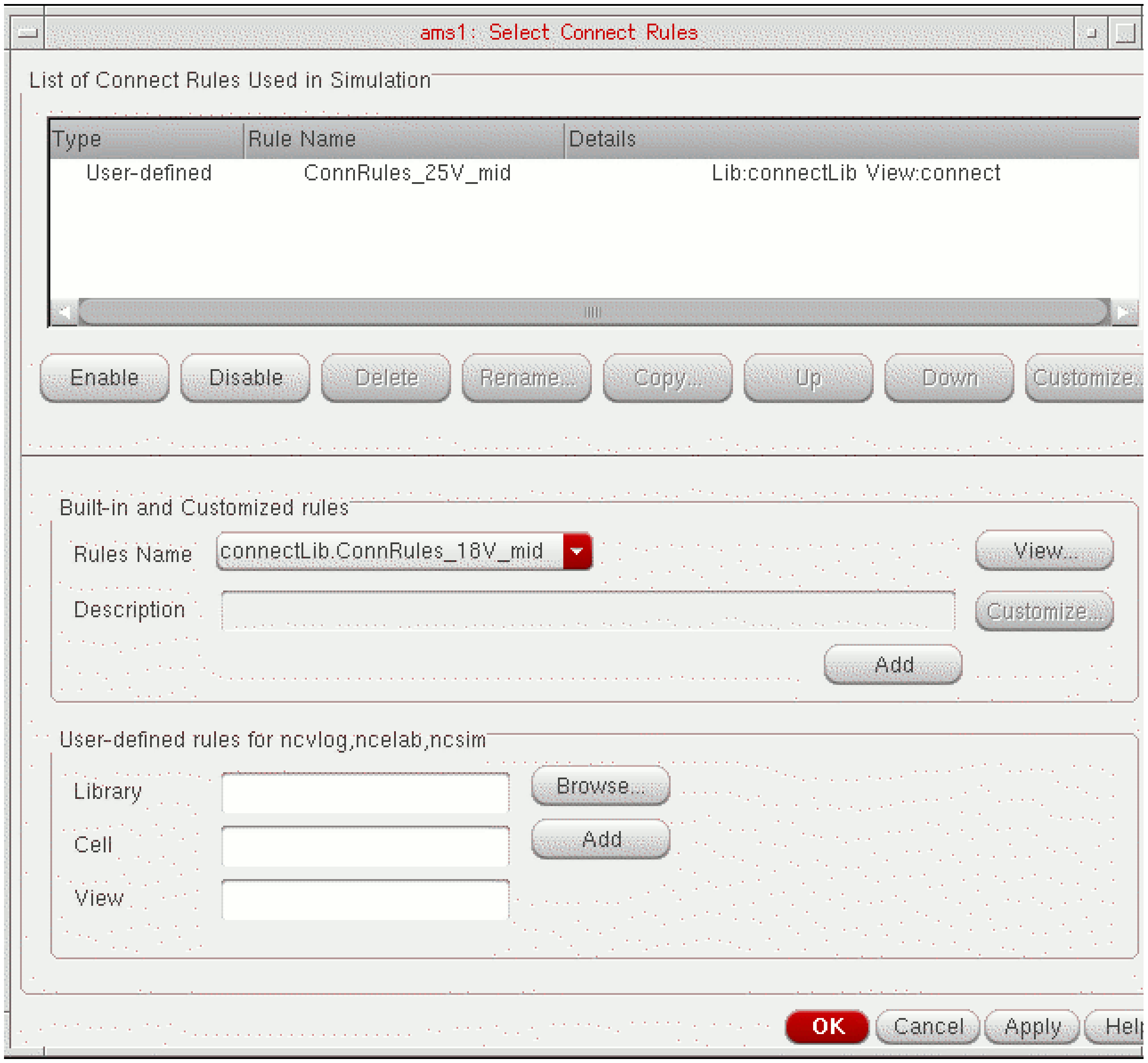
ConnRules_25V_mid appears in the *User-defined rules for ncvlog,ncelab,ncsim* group box on the Select Connect Rules form.



Library	connectLib
Cell	ConnRules_25V_mid
View	connect

5. Click *Add*.

ConnRules_25V_mid appears in the *List of Connect Rules Used in Simulation*.


ams1: Select Connect Rules

List of Connect Rules Used in Simulation

Type	Rule Name	Details
User-defined	ConnRules_25V_mid	Lib:connectLib View:connect

Built-in and Customized rules

Rules Name: connectLib.ConnRules_18V_mid

Description:

User-defined rules for ncvlog,ncelab,ncsim

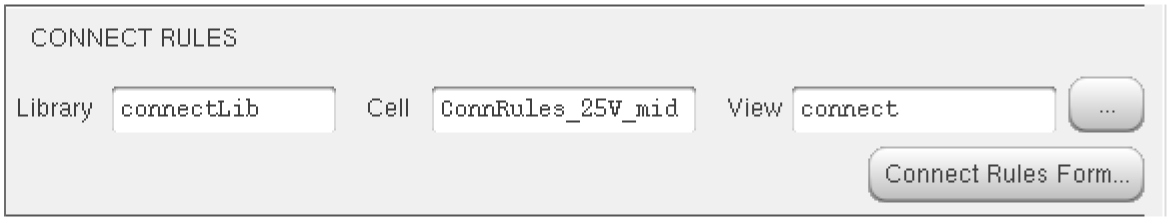
Library:

Cell:

View:

6. Click *OK*.

In the *CONNECT RULES* group box on the Netlist and Run form, *connectLib* appears in the *Library* field, *ConnRules_25V_mid* appears in the *Cell* field, and *connect* appears in the *View* field.

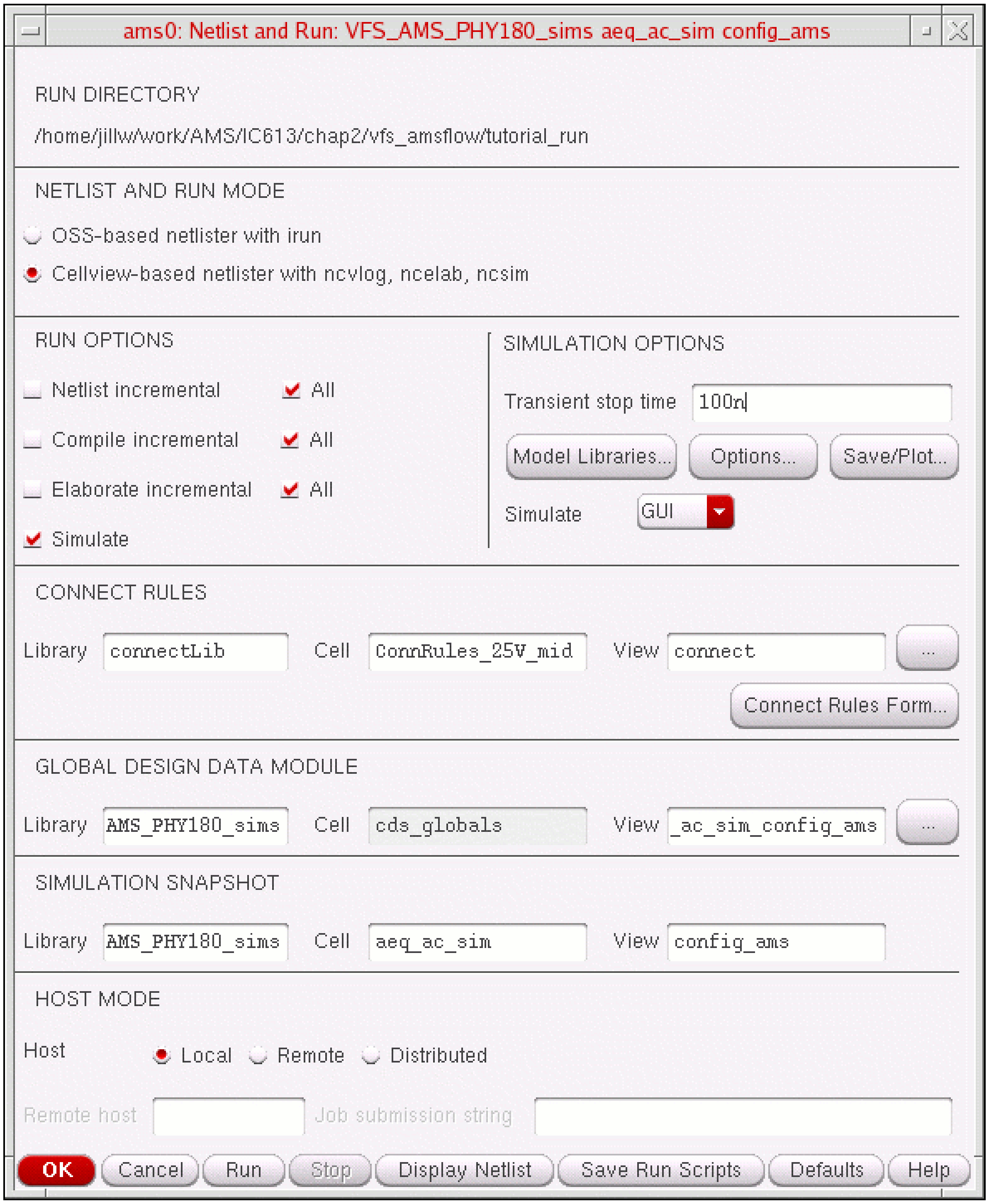

CONNECT RULES

Library	connectLib	Cell	ConnRules_25V_mid	View	connect

Specifying Run Options

To specify run options, do the following:

1. On the Netlist and Run form, in the *RUN OPTIONS* group box, select *All* for *Netlist*, *Compile*, and *Elaborate*:


ams0: Netlist and Run: VFS_AMS_PHY180_sims aeq_ac_sim config_ams

RUN DIRECTORY

/home/jillw/work/AMS/IC613/chap2/vfs_amsflow/tutorial_run

NETLIST AND RUN MODE

OSS-based netlister with irun [✓ Cellview-based netlister with ncvlog, ncelab, ncsim]

RUN OPTIONS

Netlist incremental	✓ All
Compile incremental	✓ All
Elaborate incremental	✓ All
✓ Simulate	

SIMULATION OPTIONS

Transient stop time: 100n

Model Libraries... Options... Save/Plot...

Simulate: GUI

CONNECT RULES

Library: connectLib Cell: ConnRules_25V_mid View: connect ...

Connect Rules Form...

GLOBAL DESIGN DATA MODULE

Library: AMS_PHY180_sims Cell: cds_globals View: _ac_sim_config_ams ...

SIMULATION SNAPSHOT

Library: AMS_PHY180_sims Cell: aeq_ac_sim View: config_ams

HOST MODE

Host: [✓ Local] Remote Distributed

Remote host Job submission string

❑ *Netlist All*

AMS Designer netlists all cellviews to make sure that everything is up to date. Later in the tutorial, you will switch to *Netlist incremental* so that AMS Designer netlists only changed schematics.

- ❑ *Compile All*

  AMS Designer compiles everything during this first run so that all the compilations are up to date. Later in the tutorial, you will switch to *Compile incremental* so that AMS Designer compiles only changed netlists.

- ❑ *Elaborate All*

  AMS Designer runs the elaborator. (*Elaborate incremental* causes AMS Designer to run the elaborator only when you change something in your design that requires re-elaboration.)

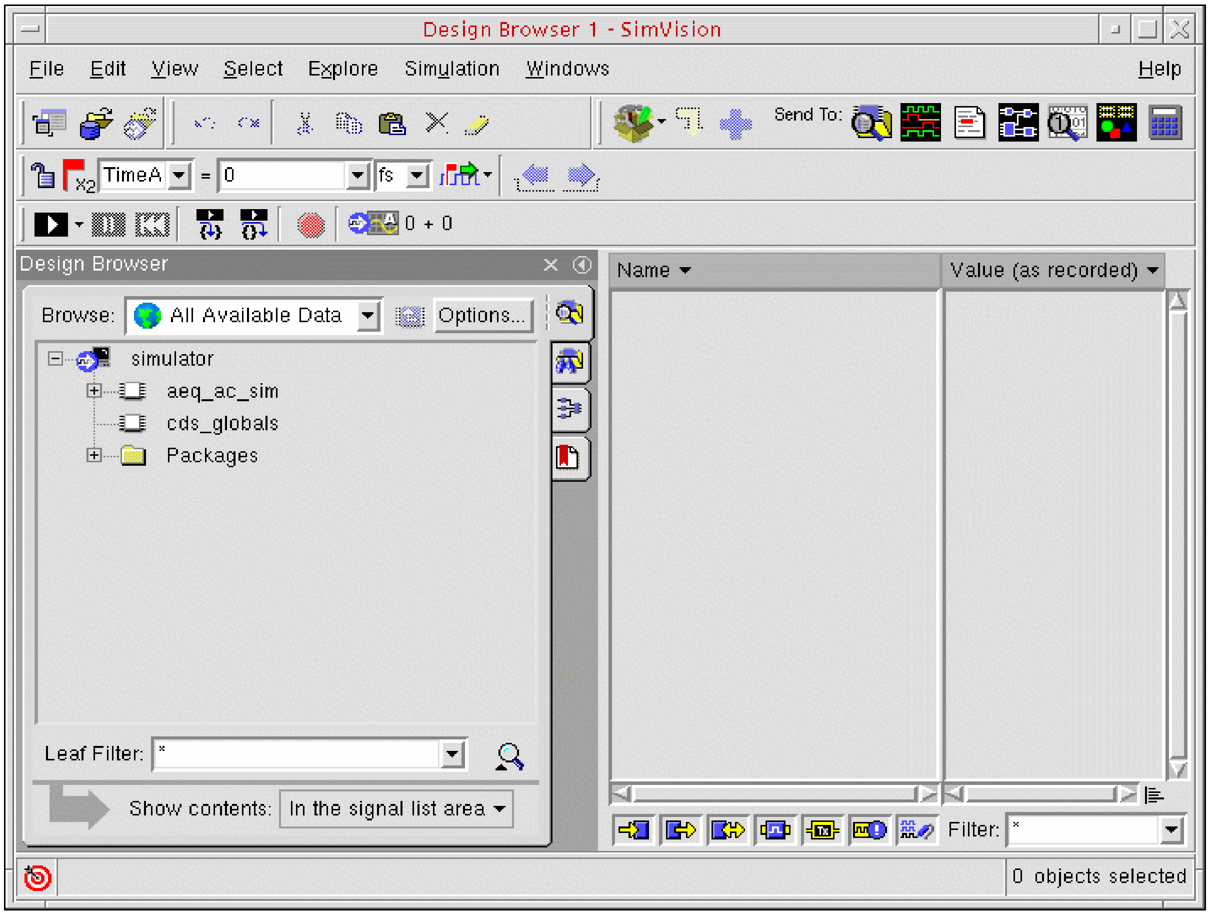
Running the Simulation

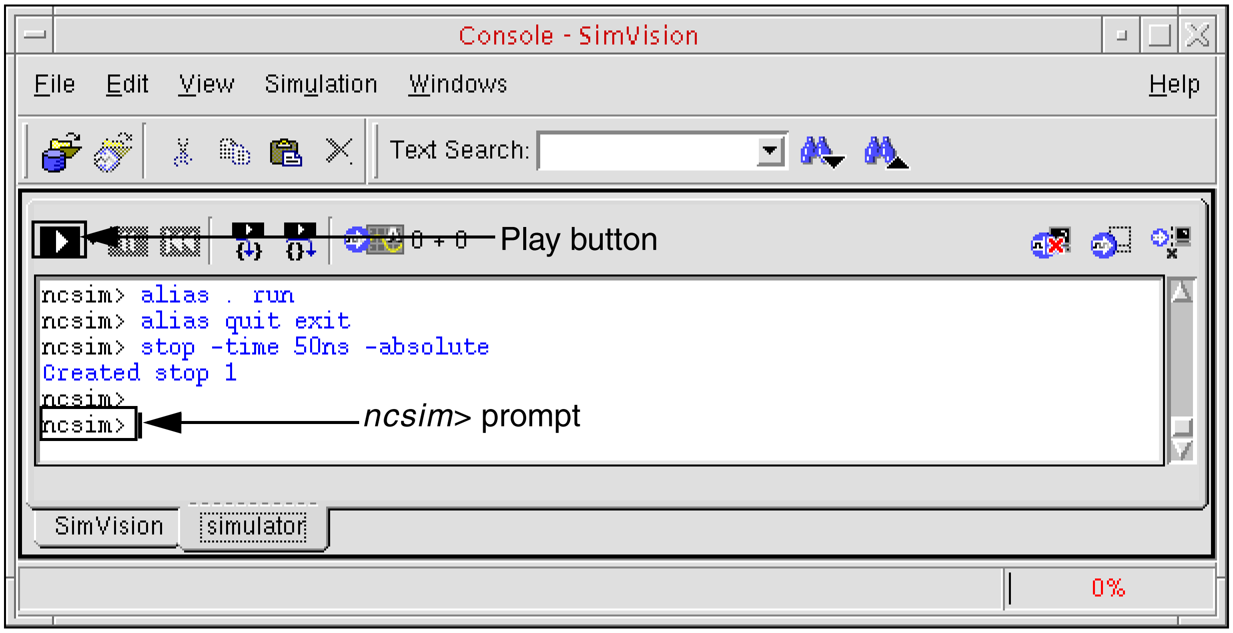
To netlist and run the simulation, do the following:

1. On the Netlist and Run form, click *Run*.

   Because you have both Netlist and Simulate turned on, the program netlists according to the options you specified and runs the simulation. Success and completion messages appear in the CIW.

The SimVision Design Browser, Waveform, and Console windows appear.





```
ncsim> alias . run
ncsim> alias quit exit
ncsim> stop -time 50ns -absolute
Created stop 1
ncsim>
ncsim>
```

2. To run to the first break point, either type `run` at the *ncsim>* prompt or click the play button.

3. Click the play button one more time (a total of twice) to simulate past the `50 ns` breakpoint set in the `demo.tcl` file.

   Waveforms appear and continue to march in the SimVision Waveform window for the remainder of the simulation. When the simulation finishes, the program returns you to the *ncsim>* prompt. Informational messages and statistics such as the following appear just before the *ncsim>* prompt on the *simulator* tab in the SimVision Console window:

```
Simulation complete via transient analysis stoptime at time 99.999999 NS
Memory Usage - 14.3M program + 300.2M data = 314.6M total
CPU Usage - 0.9s system + 51.2s user = 52.1s total (41.2% cpu)
```

   You can note these statistics, such as the CPU usage information, and compare the user time you see here to the user time you get when you use the UltraSim solver.

Using the SimVision Source Browser

For this tutorial example, we will perform the following tasks in the SimVision Source Browser:

- View the Verilog-AMS source for the `aeq_ac_sim` design.
- View signal values in the Source Browser.
- Traverse the design hierarchy in the Source Browser.

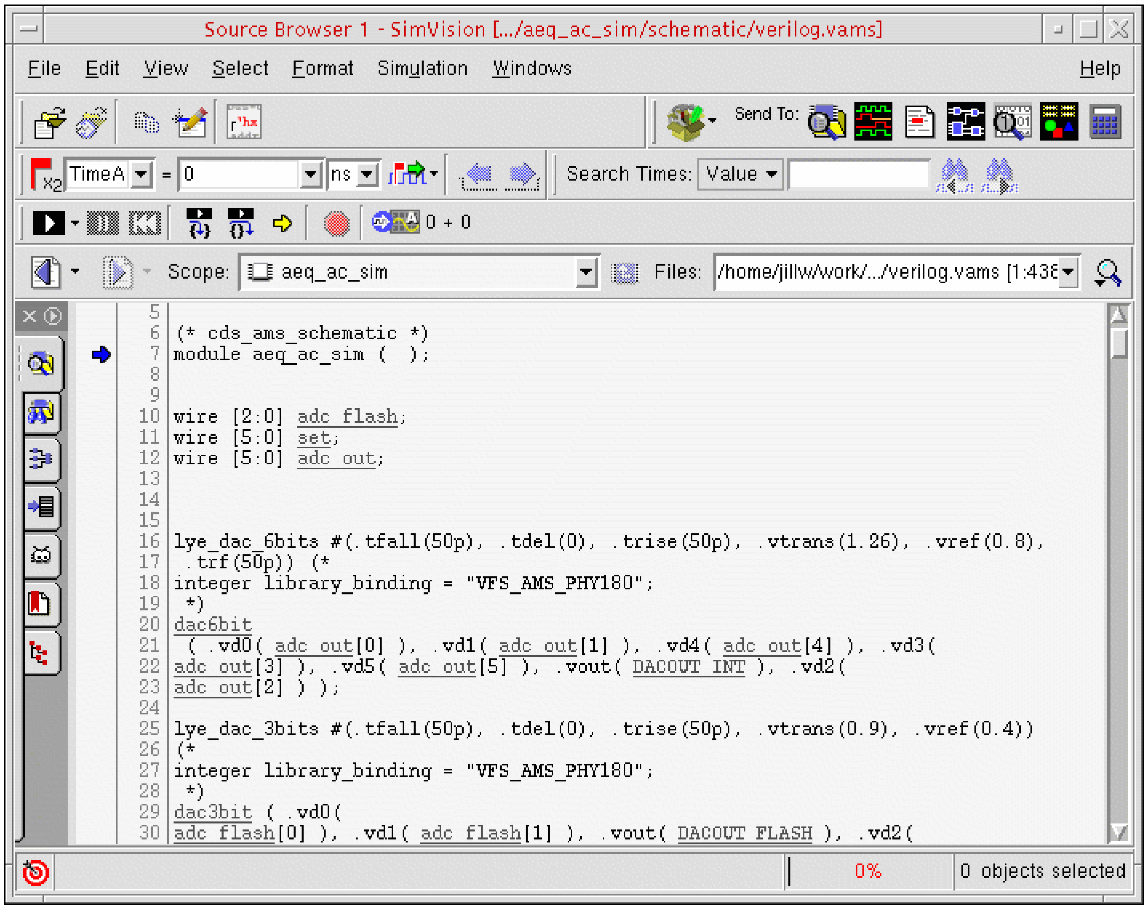
Note: For detailed information, see the *SimVision User Guide*.

Viewing the Verilog-AMS Source

To view the Verilog-AMS source for the `aeq_ac_sim` design, do the following:

➤ In the Design Browser window, choose *Windows – New – Source Browser*.

The Verilog-AMS netlist for the `aeq_ac_sim` design appears in the Source Browser.


Source Browser 1 - SimVision [.../aeq_ac_sim/schematic/verilog.vams]
File Edit View Select Format Simulation Windows Help
Send To:
TimeA = 0 ns
Search Times: Value
0 + 0
Scope: aeq_ac_sim
Files: /home/jillw/work/.../verilog.vams [1:438
0%
0 objects selected

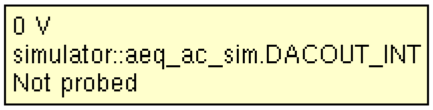
```

(* cds_ams_schematic *)
module aeq_ac_sim (  );


wire [2:0] adc_flash;
wire [5:0] set;
wire [5:0] adc_out;



lye_dac_6bits #(.tfall(50p), .tdel(0), .trise(50p), .vtrans(1.26), .vref(0.8),
 .trf(50p)) (*
integer library_binding = "VFS_AMS_PHY180";
 *)
dac6bit
 ( .vd0( adc_out[0] ), .vd1( adc_out[1] ), .vd4( adc_out[4] ), .vd3(
adc_out[3] ), .vd5( adc_out[5] ), .vout( DACOUT_INT ), .vd2(
adc_out[2] ) );

lye_dac_3bits #(.tfall(50p), .tdel(0), .trise(50p), .vtrans(0.9), .vref(0.4))
(*
integer library_binding = "VFS_AMS_PHY180";
 *)
dac3bit ( .vd0(
adc_flash[0] ), .vd1( adc_flash[1] ), .vout( DACOUT_FLASH ), .vd2(
```

Viewing Signal Values in the Source Browser

You can hover over objects such as signal names in the Source Browser to see their values. For example,

➤ Hold your cursor over `DACOUT_INT` at about line 22.

Its value (initially zero) appears as follows:

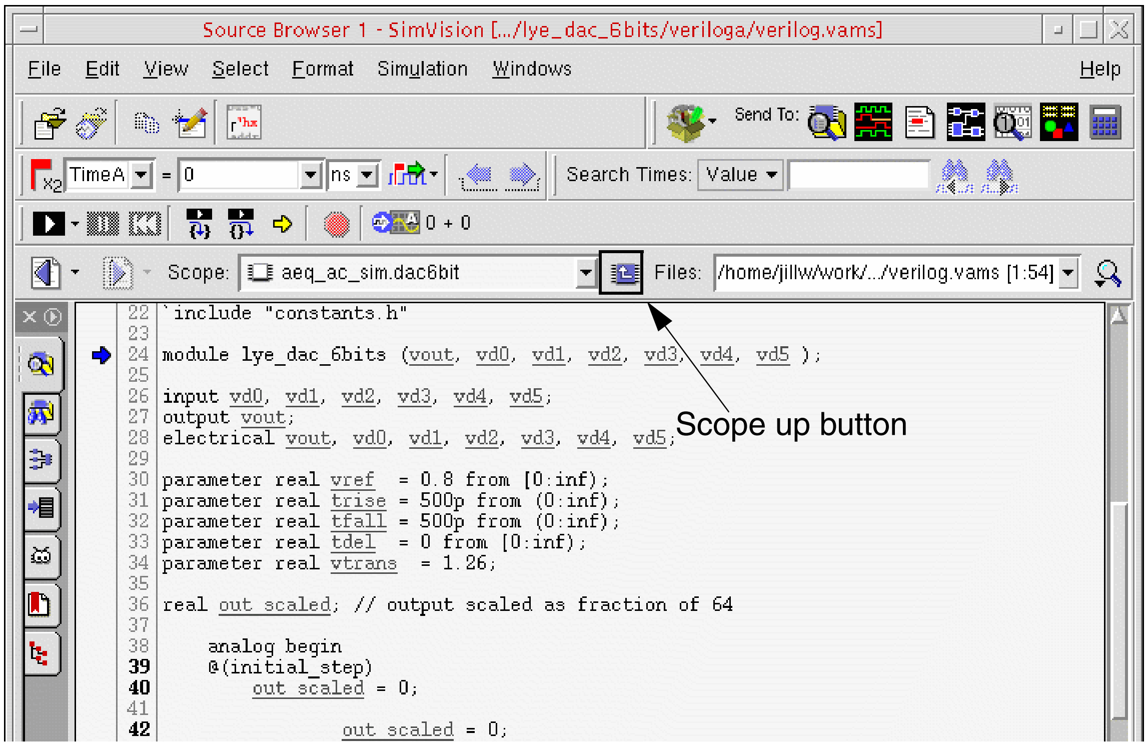


```
0 V
simulator::aeq_ac_sim.DACOUT_INT
Not probed
```

Traversing the Design Hierarchy in the Source Browser

You can traverse the design hierarchy by double-clicking on an instance. For example,

➤ Double-click `dac6bit` at about line 20 to descend into the Verilog-A view of the 6-bit digital to analog converter.



```
`include "constants.h"

module lye_dac_6bits (vout, vd0, vd1, vd2, vd3, vd4, vd5 );

input vd0, vd1, vd2, vd3, vd4, vd5;
output vout;
electrical vout, vd0, vd1, vd2, vd3, vd4, vd5;

parameter real vref  = 0.8 from [0:inf);
parameter real trise = 500p from (0:inf);
parameter real tfall = 500p from (0:inf);
parameter real tdel  = 0 from [0:inf);
parameter real vtrans  = 1.26;

real out_scaled; // output scaled as fraction of 64

    analog begin
    @(initial_step)
        out_scaled = 0;

                out_scaled = 0;
```

You can return to `aeq_ac_sim` by clicking the scope up button.

Using the SimVision Waveform Window

1. In the SimVision Waveform window, choose the following menu selections to fit the waveforms into the available area in the window:
   - ❑ *View – Zoom – Full X*
   - ❑ *View – Zoom – Full Y*
2. When you are done using SimVision, choose *File – Exit SimVision*.

Running the Tutorial Using the UltraSim Analog Solver

You can switch to using the UltraSim solver (from using the Spectre solver). For some designs, the UltraSim solver is faster, while maintaining near SPICE accuracy, and uses less memory than the Spectre solver.

Note: The Virtuoso® UltraSim™ simulator is a multi-purpose single-engine, hierarchical simulator designed for the verification of analog, mixed signal, memory, and digital circuits. You can use UltraSim for functional verification of billion-transistor memory circuits, as well as for high-precision simulation of complex analog circuits.

Before you begin, you must exit SimVision from the previous session:

➤ In any SimVision window, choose *File – Exit SimVision*.

To run the tutorial using the UltraSim solver, you will perform the following tasks:

- Switch to Using the UltraSim Solver on page 64
- Specify Incremental Netlisting, Compilation, and Elaboration on page 64
- Change the Simulation Snapshot Name on page 66
- Run the Simulation Using the UltraSim Solver and View Results on page 66

Switch to Using the UltraSim Solver

Video

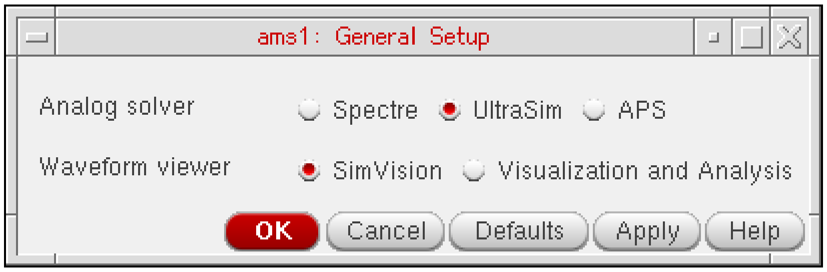
Show Me

To switch to using the UltraSim solver, do the following:

1. In the Virtuoso® Hierarchy Editor, choose *AMS – Detail Setup – General Setup*.

   The General Setup form appears.

2. For *Analog Solver*, select *UltraSim*.



Analog solver ○ Spectre [✓ UltraSim] ○ APS

Waveform viewer [✓ SimVision] ○ Visualization and Analysis

3. Click *OK*.

   You are now set up to use the AMS Designer simulator with the UltraSim solver.

Tip

You can also switch solvers using the Quick Setup form. See also Using the Quick Setup Form on page 38.

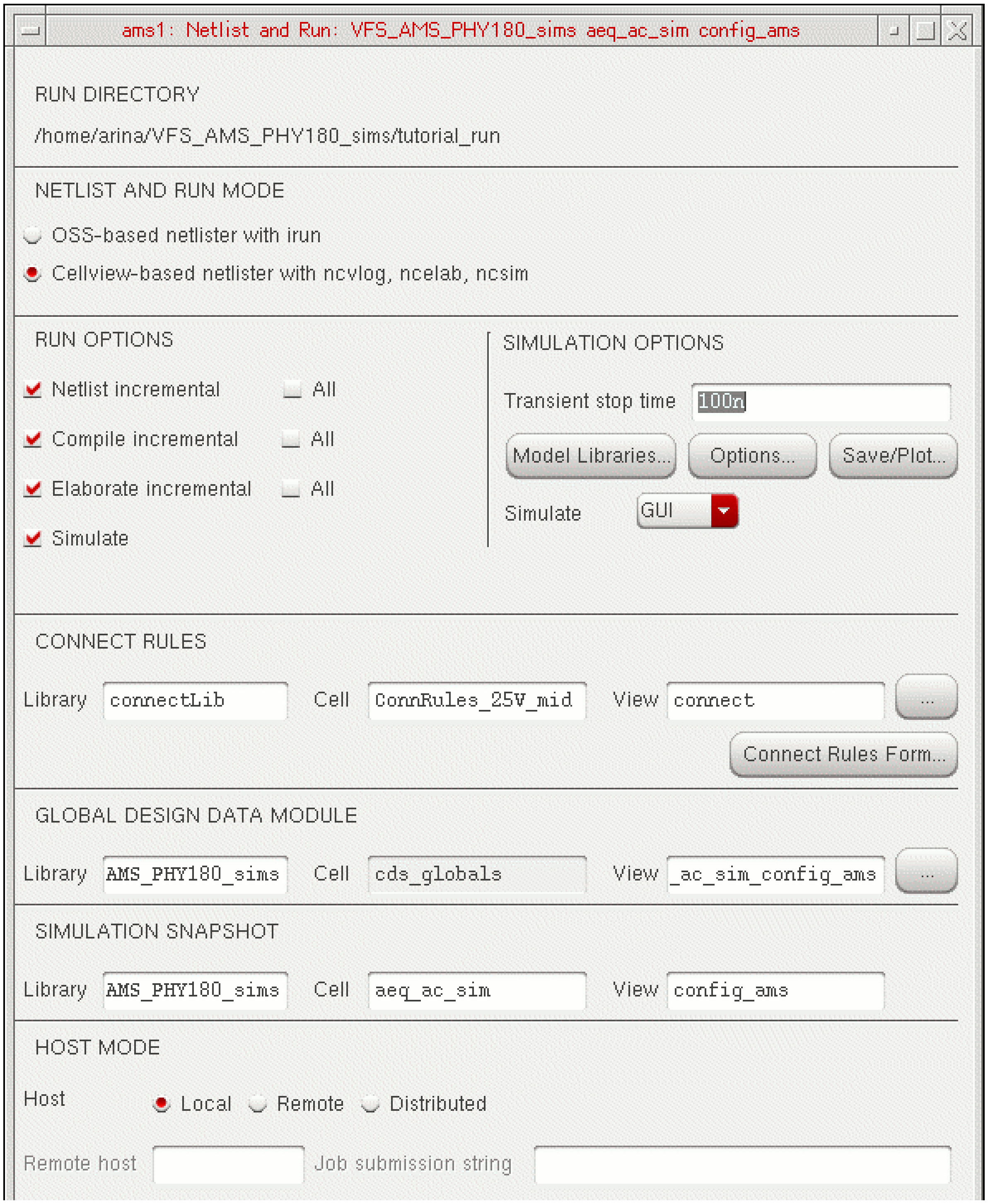
Specify Incremental Netlisting, Compilation, and Elaboration

To specify incremental netlisting, compiling, and elaborating, do the following:

1. In the Virtuoso® Hierarchy Editor, choose *AMS – Netlist and Run*.

   The Netlist and Run form appears.

2. In the *RUN OPTIONS* group box, turn on *Netlist incremental*, *Compile incremental*, and *Elaborate incremental*.


ams1: Netlist and Run: VFS_AMS_PHY180_sims aeq_ac_sim config_ams

RUN DIRECTORY

/home/arina/VFS_AMS_PHY180_sims/tutorial_run

NETLIST AND RUN MODE

○ OSS-based netlister with irun

● Cellview-based netlister with ncvlog, ncelab, ncsim

RUN OPTIONS

- ✓ Netlist incremental ☐ All
- ✓ Compile incremental ☐ All
- ✓ Elaborate incremental ☐ All
- ✓ Simulate

SIMULATION OPTIONS

Transient stop time 100n

Simulate GUI

CONNECT RULES

Library connectLib Cell ConnRules_25V_mid View connect

GLOBAL DESIGN DATA MODULE

Library AMS_PHY180_sims Cell cds_globals View _ac_sim_config_ams

SIMULATION SNAPSHOT

Library AMS_PHY180_sims Cell aeq_ac_sim View config_ams

HOST MODE

Host ● Local ○ Remote ○ Distributed

Remote host Job submission string

AMS Designer netlists and compiles only those blocks that changed, and elaborates only if necesary. You are now ready to netlist and run the simulation.

Important

We recommend that you change the simulation snapshot name before running the simulation using a different solver. You must select *All* for the *Elaborate* run option.

Change the Simulation Snapshot Name

It is a good idea to change the simulation snapshot name before running the simulation using the UltraSim solver so that the elaborator does not overwrite the simulation snapshot that resulted from the simulation you ran using the Spectre solver. With two differently-named snapshots, you can resimulate either snapshot as necessary.

To change the simulation snapshot name, do the following:

1. In the *RUN OPTIONS* section of the Netlist and Run form, select *All* for the *Elaborate*.

   You cannot use *Elaborate incremental* when changing a simulation snapshot name.

2. In the *SIMULATION SNAPSHOT* section of the Netlist and Run form, add a `_US` suffix to the name that appears in the *View* field to indicate that this snapshot will be for the UltraSim solver.

   Note: You do not have to use a `_US` suffix. You can type whatever new name you prefer.

   The AMS Designer simulator will use this name when it creates the simulation snapshot.

Run the Simulation Using the UltraSim Solver and View Results

To netlist the design and run the simulation using the AMS Designer simulator with the UltraSim solver in one step, do the following:

1. On the Netlist and Run form, click *Run*.

   The program netlists and simulates according to the options you specified in the *RUN OPTIONS* group box. Status and completion messages appear in the command interpreter window (CIW). The `simulation.log` file appears in a text window. The SimVision Console, Design Browser, and Waveform windows appear.

2. In any SimVision window, click the play button twice to run the simulation to the end.

The first click takes you to the first breakpoint set in the `demo.tcl` file (at 50 ns). The second click simulates to the end.

Waveforms appear in the Waveform window and continue marching during the remainder of the simulation.

CPU usage information appears in the Console window. You can compare the user time you see here to the user time you got when you used the Spectre solver. For this configuration, the UltraSim solver typically finishes in about one-third the time it takes using the Spectre solver.

```
Simulation complete via transient analysis stoptime at time 99.999999 NS
Memory Usage - 14.3M program + 341.6M data = 355.9M total
CPU Usage - 2.3s system + 16.1s user = 18.5s total (17.6% cpu)
```

Note: See the "Simulation Options" chapter of the *Virtuoso UltraSim Simulator User Guide* for information about settings that can affect simulation accuracy and performance.

3. In the SimVision Waveform window, choose the following menu selections to fit the waveforms into the available area in the window:

❑ *View – Zoom – Full X*

❑ *View – Zoom – Full Y*

4. When you are done using SimVision, choose *File – Exit SimVision*.

5. In the CIW, choose *File – Exit* to exit the workbench and close the application windows.

Running the Tutorial from the Command Line

You can use the AMS Designer simulator without using the AMS Designer environment.

The `run_amsdesigner` script contains the commands necessary to netlist, compile, elaborate, and simulate the tutorial design.

1. Verify that you are in the directory where you installed the tutorial (`vfs_amsflow`).
2. (Optional) To view the contents of the script, type the following at the command prompt:

   ```
   more run_amsdesigner
   ```

   The `run_amsdesigner` script contains the following lines:

   ```
   #!/bin/csh -f

   setenv TUTORHOME `pwd`

   amsdesigner -lib VFS_AMS_PHY180_sims -cell aeq_ac_sim -view config_ams -cdslib
   $TUTORHOME/cds.lib -netlist all -compile all -elaborate -simulate -modelpath
   '$TUTORHOME/spectre_models/gpdk.scs(NN)' -cdsglobals overwriteEdits
   ```

3. To run the script, type the following at the command prompt:

   ```
   ./run_amsdesigner
   ```

   The `amsdesigner` command creates netlists for the schematics, compiles the netlists, elaborates the design, and starts SimVision so you can simulate the design.
4. (Optional) Run the simulation as you did earlier in the tutorial.
5. To exit SimVision, choose *File – Exit SimVision*.

2 Setting Up the AMS Designer Environment

The following configuration and control files help you manage your data and the operation of the AMS Designer environment:

Configuration File	Description
`cds.lib`	Defines your design libraries and associates logical library names with physical library locations. For more information, see “The cds.lib File” in the *Virtuoso AMS Designer Simulator User Guide* and in *NC-Verilog Simulator Help*. See also “About the library path editor and the library definitions files” in the *Cadence Library Path Editor User Guide*.
`.cdsenv`	Specifies environment settings for Cadence® applications. For more information, see “Specifying Environment Settings” in the *Virtuoso Design Environment User Guide*.
state files	Specifies simulation and environment setup information. AMS Designer automatically writes state files to the `.amsd_state` directory in the run directory (according to the lib/cell/view). For information on state files, see the *Virtuoso Analog Designer Environment L User Guide*. See also“Loading State Files” on page 170 and “Saving State Files” on page 170. **Note:** If you are migrating from an earlier version of AMS Designer and have an `ams.env` file, the first time you run this version of AMS Designer on a previous design, the software reads your `ams.env` file and saves this information to new state files.

Control File	Description
`hdl.var`	Defines variables and settings for the compiler, elaborator, and simulator. For more information, see “The hdl.var File” in the *Virtuoso AMS Designer Simulator User Guide* and in *NC-Verilog Simulator Help*. *Tip* You can specify your `hdl.var` file on the Quick Setup form or in the AMS Options form.
Tcl input script	Specifies Cadence-supported Tcl commands to interact with the simulator. For a description of the Tcl commands, see Appendix B, “Tcl-Based Debugging,” in the *Virtuoso AMS Designer Simulator User Guide*. *Tip* You can specify a Tcl input script on the Quick Setup form or in the AMS Options form.
Simulation control file	Specifies commands that tell the analog solver how to simulate the design. For more information, see “Specifying Controls for the Analog Solver” in the *Virtuoso AMS Designer Simulator User Guide*. *Tip* You can specify a simulation control file on the Quick Setup form.

See the following topics for more information:

- Understanding TMP Libraries on page 71
- Specifying Preferences for Netlisting and Compiling on page 73
- Importing Customized Built-In Connect Rules from ADE on page 74
- Opening a config View in the Hierarchy Editor on page 74
- Adding AMS to the Menu Bar in the Hierarchy Editor on page 76

- Initializing the AMS Designer Environment on page 77
- Using Quick Setup on page 83
- Using the AMS Options Form on page 125

Understanding TMP Libraries

You can have the following types of libraries in the Virtuoso® AMS Designer environment:

Library	Description
Master library	The library you define in your `cds.lib` library definitions file: `DEFINE` *`masterLibraryName directoryPath`*
TMP library	A shadow copy of the master library; you can use explicit or implicit TMP libraries

As the cellview-based netlister runs, it writes data into the library. If your master library is read-only, you can create a temporary library (or TMP library) to which the netlister can write its data. The structure of a TMP library is exactly the same as that of the master library (same number of cells, same cell names, same views, same view names, and so on).

While the TMP library structure exactly matches that of the master library, the TMP library does not contain all the files that the master library contains. Instead, the TMP library contains only those files that the netlister writes (such as the `verilog.vams` file). You can think of it as the TMP and master libraries having the same directory tree structure, but not the same files in the tree.

TMP libraries can be either explicit or implicit. See the following topics for details:

- Understanding Explicit TMP Libraries on page 72
- Understanding Implicit TMP Libraries on page 72

For more information about TMP libraries, see “Temporary Directory for a Library” in the “Cadence Library Structure” chapter of the *Cadence Application Infrastructure User Guide*.

Understanding Explicit TMP Libraries

You assign each explicit TMP library using an `ASSIGN` statement of the following form in your `cds.lib` library definitions file:

```
ASSIGN masterLibraryName TMP TMPdirPath
```

You must pair each `ASSIGN` statement with the `DEFINE` statement for the corresponding master library:

```
DEFINE masterLibraryName directoryPath
ASSIGN masterLibraryName TMP TMPdirPath
```

You can read more about how to set up an explicit TMP library in "Compiling into Temporary Libraries" on page 261.

To assign a specific directory as the root directory for all implicit TMP libraries, type an `ASSIGN` statement of the following form in your `cds.lib` file:

```
ASSIGN AllLibs TmpRootDir directory
```

For example, if you have the following line in your `cds.lib` file

```
ASSIGN AllLibs TmpRootDir ./myTmpLibs
```

AMS Designer creates the TMP library directory structure in the `./myTmpLibs` directory. So, if you have master libraries `Lib1` and `Lib2`, for example, the software creates

```
./myTmpLibs/Lib1
./myTmpLibs/Lib2
```

For more information about this form of the `ASSIGN` statement, see "ASSIGN" under "cds.lib Statements" in "The cds.lib File" in the "Setting Up Your Environment" chapter of the *Virtuoso® AMS Designer Simulator User Guide*.

Understanding Implicit TMP Libraries

You use the `CDS_IMPLICIT_TMPDIR` option to specify an implicit TMP directory to search for design data and to hold new design data. When you do, the software automatically creates the library directory structure in the specified TMP directory using the master library names you have defined in your `cds.lib` file.

When using the AMS Designer environment, the default behavior is to prefer master and explicit TMP libraries over implicit TMP libraries when netlisting and compiling. If you choose to set the preference for implicit TMP libraries instead, the program creates an `inhl` directory in the netlist directory in your run directory and write temporary/derived data to that directory:

```
runDirectory/netlist/inhl
```

For more information about implicit TMP libraries, see also “Using Implicit TMP Libraries” in the *Virtuoso® AMS Designer Simulator User Guide*.

Specifying Preferences for Netlisting and Compiling

You can specify whether you want the program to netlist and compile cells to master libraries and explicit TMP libraries (the default behavior) or to implicit TMP libraries.

By default, AMS Designer (when run from the Virtuoso® Hierarchy Editor) prefers master and explicit TMP libraries when netlisting and compiling. You can specify this behavior explicitly in your `.cdsenv` file as follows:

```
ams.netlisterOpts    preferMEOverImplicit    boolean    t
```

Because this behavior is the default, you do not have to specify this setting.

Note: If you are familiar with running AMS Designer from the Virtuoso Analog Design Environment (ADE), the default behavior in that environment is for the software to prefer implicit TMP libraries. The `.cdsenv` file setting you must use to get this same behavior running AMS Designer from the Virtuoso Hierarchy Editor is as follows:

```
ams.netlisterOpts    preferMEOverImplicit    boolean    nil
```

See also “Importing Customized Built-In Connect Rules from ADE” on page 74.

If you have not changed the `preferMEOverImplicit` setting from its default value (`t`), the program compiles built-in and user-defined connect rules according to the `compileCRsForPlugin` setting in your `.cdsenv` file as follows:

- By default, the setting is

  ```
  ams.envOpts    compileCRsForPlugin    string    "doNotCompile"
  ```

  The program does not compile built-in and user-defined connect rules. In this case, you must precompile your built-in and user-defined connect rules.

- If you use this setting:

  ```
  ams.envOpts    compileCRsForPlugin    string    "explicitsOrMaster"
  ```

  the program compiles built-in and user-defined connect rules into the master `connectLib` or the explicit TMP `connectLib` only.

If `preferMEOverImplicit` is `nil`, the program ignores the `compileCRsForPlugin` setting.

Importing Customized Built-In Connect Rules from ADE

If you have created a customized built-in connect rule in the Virtuoso® Analog Design Environment (ADE) and saved that state, you can import and use your customized built-in connect rule into the Virtuoso AMS Designer environment by doing one of the following:

- Use the `preferMEOverImplicit` setting to specify a preference for implicit TMP libraries. For example, in your `.cdsenv` file:

  ```
  ams.netlisterOpts    preferMEOverImplicit    boolean    nil
  ```

  Note: If `preferMEOverImplicit` is `nil`, the program ignores the `compileCRsForPlugin` setting.

  You can also specifying this setting in the command interpreter window (CIW).

  The software will netlist and compile your customized built-in connect rules in the AMS Designer environment.

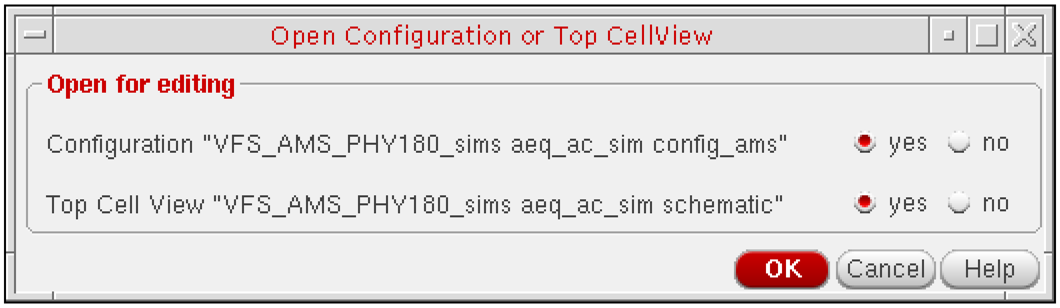
- Alternatively, you can copy customized built-in connect rules from the ADE run directory and precompile them. You can select and use precompiled customized built-in connect rules in the AMS Designer environment.

Opening a config View in the Hierarchy Editor

When you have a design configuration, you can control AMS Designer from the *AMS* menu in the Virtuoso® Hierarchy Editor. To open your config view in the hierarchy editor, do one of the following in the command interpreter window (CIW):

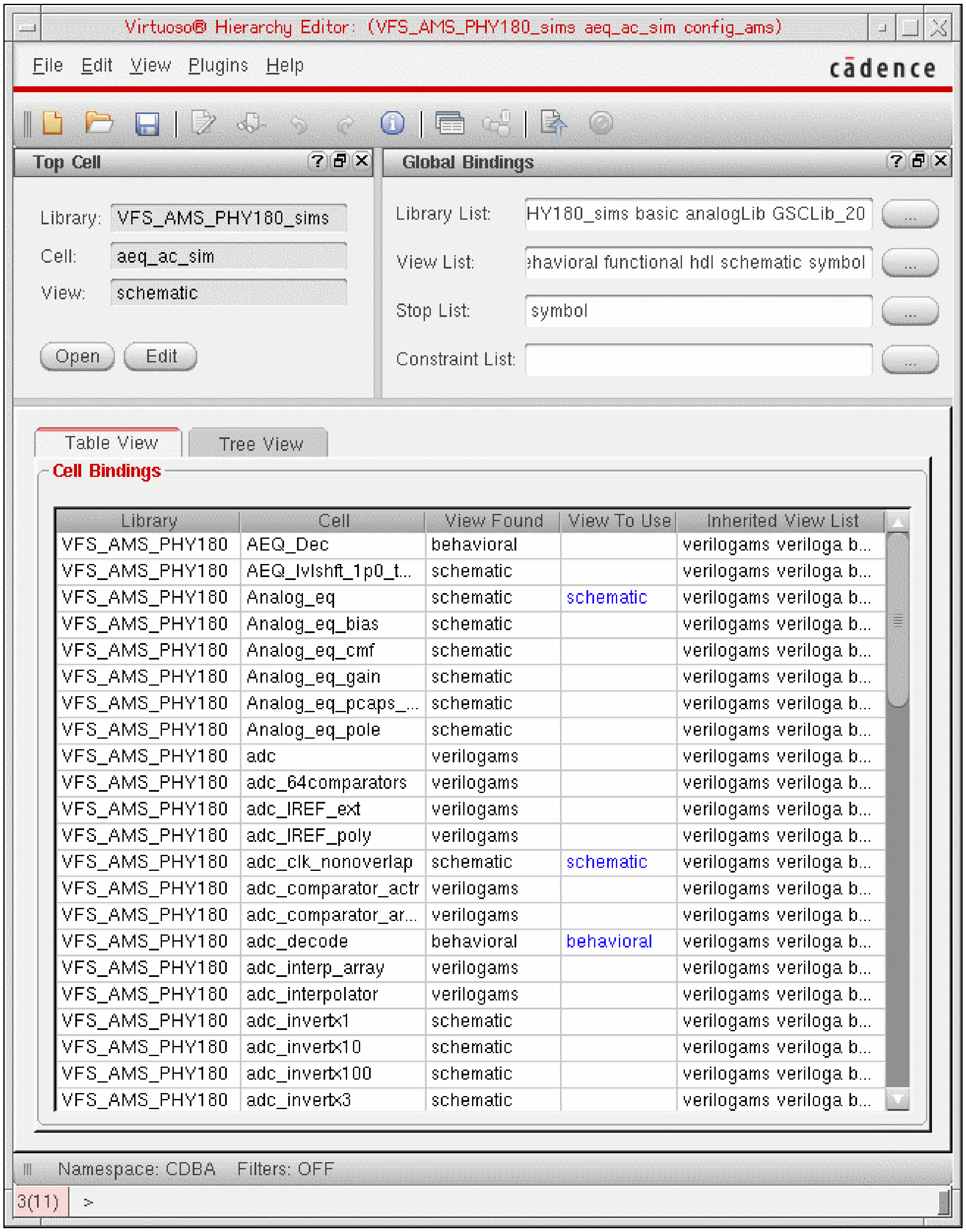
- Choose *File – Open* and open a config view from the Open File form that appears.
- Choose *Tools – Library Manager* and open a config view from the Library Manager window.

In either case, be sure to select *yes* to open the *Configuration* on the Open Configuration form that appears.



Configuration "VFS_AMS_PHY180_sims aeq_ac_sim config_ams"	[✓ yes]	no
Top Cell View "VFS_AMS_PHY180_sims aeq_ac_sim schematic"	[✓ yes]	no

The Virtuoso® Hierarchy Editor window appears.


Virtuoso® Hierarchy Editor: (VFS_AMS_PHY180_sims aeq_ac_sim config_ams)
File Edit View Plugins Help
cādence
Top Cell
Global Bindings
Open
Edit
Table View
Tree View
Cell Bindings

Library: VFS_AMS_PHY180_sims
Cell: aeq_ac_sim
View: schematic

Library List: HY180_sims basic analogLib GSCLib_20
View List: ehavioral functional hdl schematic symbol
Stop List: symbol
Constraint List:

Library	Cell	View Found	View To Use	Inherited View List
VFS_AMS_PHY180	AEQ_Dec	behavioral		verilogams veriloga b...
VFS_AMS_PHY180	AEQ_lvlshft_1p0_t...	schematic		verilogams veriloga b...
VFS_AMS_PHY180	Analog_eq	schematic	schematic	verilogams veriloga b...
VFS_AMS_PHY180	Analog_eq_bias	schematic		verilogams veriloga b...
VFS_AMS_PHY180	Analog_eq_cmf	schematic		verilogams veriloga b...
VFS_AMS_PHY180	Analog_eq_gain	schematic		verilogams veriloga b...
VFS_AMS_PHY180	Analog_eq_pcaps_...	schematic		verilogams veriloga b...
VFS_AMS_PHY180	Analog_eq_pole	schematic		verilogams veriloga b...
VFS_AMS_PHY180	adc	verilogams		verilogams veriloga b...
VFS_AMS_PHY180	adc_64comparators	verilogams		verilogams veriloga b...
VFS_AMS_PHY180	adc_IREF_ext	verilogams		verilogams veriloga b...
VFS_AMS_PHY180	adc_IREF_poly	verilogams		verilogams veriloga b...
VFS_AMS_PHY180	adc_clk_nonoverlap	schematic	schematic	verilogams veriloga b...
VFS_AMS_PHY180	adc_comparator_actr	verilogams		verilogams veriloga b...
VFS_AMS_PHY180	adc_comparator_ar...	verilogams		verilogams veriloga b...
VFS_AMS_PHY180	adc_decode	behavioral	behavioral	verilogams veriloga b...
VFS_AMS_PHY180	adc_interp_array	verilogams		verilogams veriloga b...
VFS_AMS_PHY180	adc_interpolator	verilogams		verilogams veriloga b...
VFS_AMS_PHY180	adc_invertx1	schematic		verilogams veriloga b...
VFS_AMS_PHY180	adc_invertx10	schematic		verilogams veriloga b...
VFS_AMS_PHY180	adc_invertx100	schematic		verilogams veriloga b...
VFS_AMS_PHY180	adc_invertx3	schematic		verilogams veriloga b...

Namespace: CDBA Filters: OFF
3(11) >

Adding AMS to the Menu Bar in the Hierarchy Editor

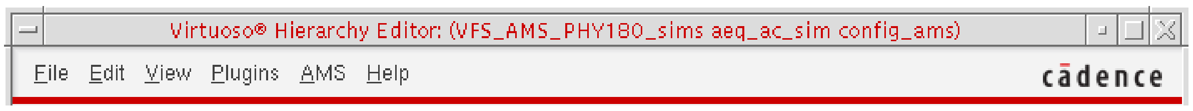
Important

You must have an open design configuration before continuing.

To add *AMS* to the menu bar in the Virtuoso® Hierarchy Editor, do the following:

➤ Choose *Plug-Ins – AMS.*

AMS appears on the menu bar.


Virtuoso® Hierarchy Editor: (VFS_AMS_PHY180_sims aeq_ac_sim config_ams)
File Edit View Plugins AMS Help
cādence

The *AMS* menu contains controls for the AMS Designer environment and simulator. When the *AMS* menu first appears, the only item you can select is *Initialize*.

Initializing the AMS Designer Environment

The first step in configuring your AMS Designer environment is specifying a run directory. Run directories allow you to tailor the AMS Designer environment for different simulation runs of a design and to track the data associated with each run. Elements of AMS Designer that can vary as you change run directories include the following:

- ❑ AMS Designer behaviors having to do with connect rules and SimVision
- ❑ The analog simulation control file for simulation
- ❑ The destination directory for error log files
- ❑ The directory that holds waveform data
- ❑ The `cds_globals` module associated with the design
- ❑ The simulation snapshot

Important

You do not need to follow these steps if you have previously specified either a new or an existing run directory for a particular configuration by marking the *Always use this run directory for this configuration* check box on the AMS Initialize form.

Always use this run directory for this configuration ✓

To initialize your AMS Designer environment, do the following:

1. Choose *AMS – Initialize*.

   The AMS Initialize form appears.

   Note: Other items on the *AMS* menu become active.

2. Specify a run directory by doing one of the following:

   - ❑ Specify an existing run directory
   - ❑ Specify a new run directory
   - ❑ Specify a new run directory and copy data over from an existing run directory
   - ❑ Specify a new run directory and import data from an existing ADE state

Specifying an Existing Run Directory

To specify an existing run directory on the AMS Initialize form, do the following:

1. Select *Existing Run Directory*.

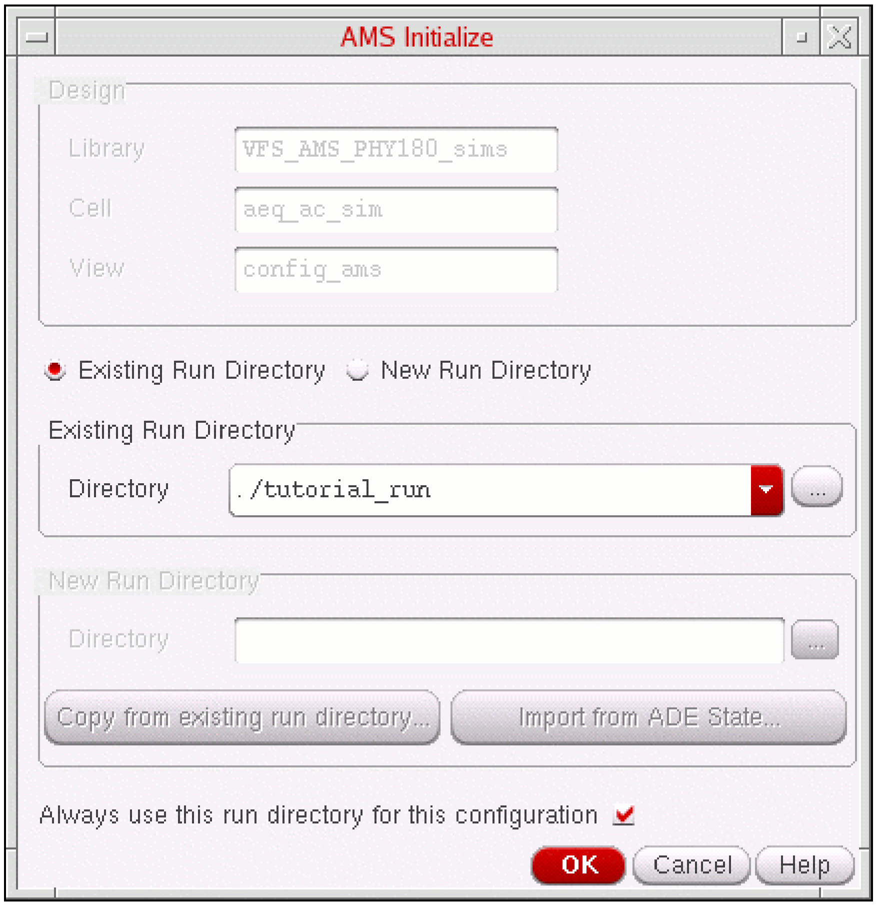
   The *Existing Run Directory* group box becomes active and the *New Run Directory* group box becomes inactive.

2. Specify an existing run directory using one of the following methods:

   ❑ In the *Directory* field, type the path to and name of the existing run directory.

   ❑ Use the drop-down combo box in the *Directory* field to select a previously specified run directory.

   ❑ Click the browse button to the right of the *Directory* field and use the Choose form that appears to navigate to and select an existing run directory.


AMS Initialize

Design

Library	VFS_AMS_PHY180_sims
Cell	aeq_ac_sim
View	config_ams

[✓ Existing Run Directory] New Run Directory

Existing Run Directory

Directory ./tutorial_run

New Run Directory

Directory

Always use this run directory for this configuration ✓

3. (Optional) Mark the *Always use this run directory for this configuration* check box so that you can skip this procedure for this configuration in the future.

4. Click *OK*.

Specifying a New Run Directory

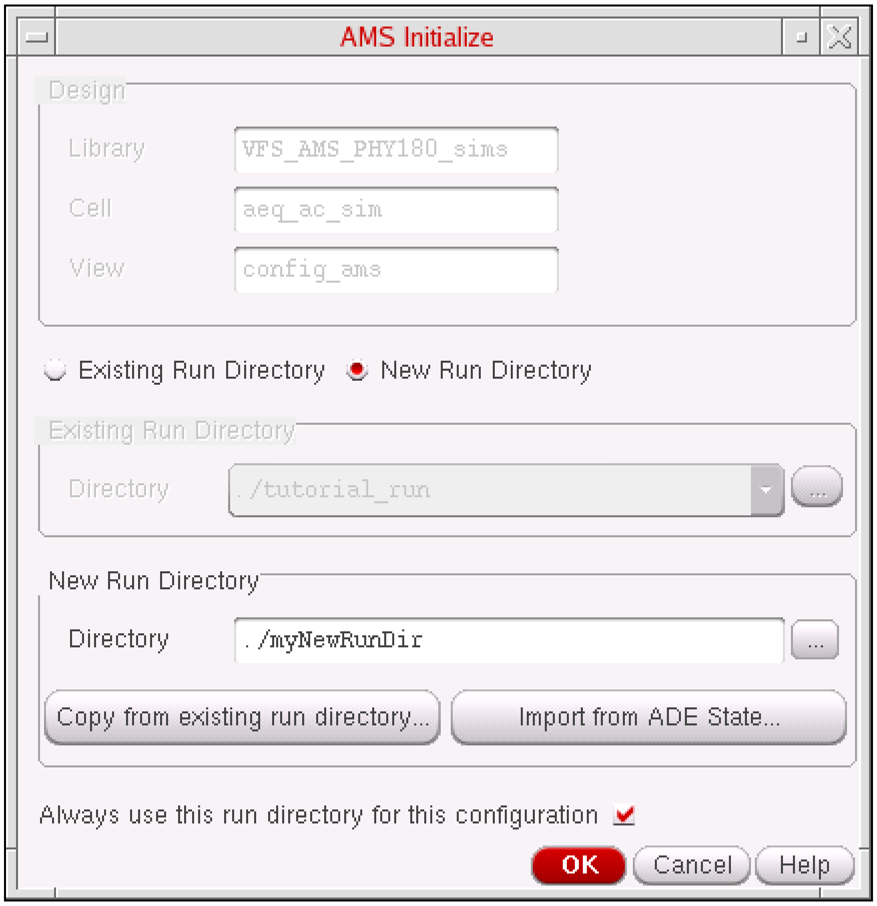
1. Select *New Run Directory*.

   The *New Run Directory* group box becomes active and the *Existing Run Directory* group box becomes inactive.

2. Specify a new run directory using one of the following methods:

   ❑ In the *Directory* field, type the path to and name of a new run directory.

   ❑ Click the browse button to the right of the *Directory* field and use the Choose form that appears to navigate to and select a new run directory.



AMS Initialize

Design

Library: VFS_AMS_PHY180_sims

Cell: aeq_ac_sim

View: config_ams

Existing Run Directory [✓ New Run Directory]

Existing Run Directory

Directory: ./tutorial_run

New Run Directory

Directory: ./myNewRunDir

Always use this run directory for this configuration ✓

3. (Optional) You can populate your new run directory with data from an existing run directory or by importing an ADE state. See

   ❑ "Copying from an Existing Run Directory" on page 81

   ❑ "Importing from an ADE State" on page 81

4. (Optional) Mark the *Always use this run directory for this configuration* check box so that you can skip this procedure for this configuration in the future.

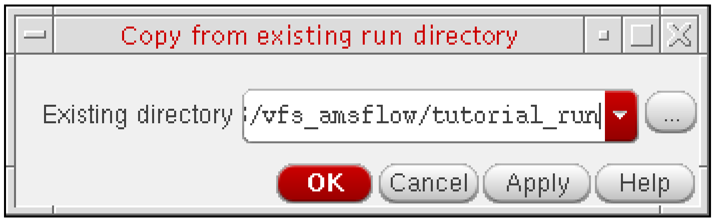
5. Click *OK*.

Copying from an Existing Run Directory

To copy data from an existing run directory to a new run directory, do the following:

1. After you have specified the location of your new run directory, click *Copy from existing run directory*.

   A form appears.



Existing directory: /vfs_amsflow/tutorial_run

2. To specify the location of the existing run directory, do one of the following:

   ❑ In the *Existing directory* field, type the path to and name of the existing run directory.

   ❑ Use the drop-down combo box in the *Existing directory* field to select a previously specified run directory.

   ❑ Click the browse button to the right of the *Existing directory* field and use the Choose form that appears to navigate to and select an existing run directory.

3. On the Copy form, click *OK*.

4. On the AMS Initialize form, click *OK*.

Importing from an ADE State

To import data from an ADE state to a new run directory, do the following:

1. After you have specified the location of your new run directory, click *Import from ADE State*.

   The Loading State form appears. You can import state information from an analog design environment (ADE) state to a newly created run directory. If a selection is not active on this form, the information is not present in the selected state. See the *Virtuoso Analog Design Environment L User Guide* for information about this form.

   Note: If you are familiar with previous versions of AMS Designer, see also important information in "AMS Designer Simulations" on page 698.

2. On the Loading State form, click *OK*.
3. On the AMS Initialize form, click *OK*.

After you create and initialize the run directory, you can always choose to load a different state file. You can load a different state file at any time.

Using Quick Setup

If you have files that specify your simulation settings, you can use the Quick Setup form to establish your AMS setup. If you do not have these files, you can use the *AMS – Detailed Setup* menu to specify options and simulation settings.

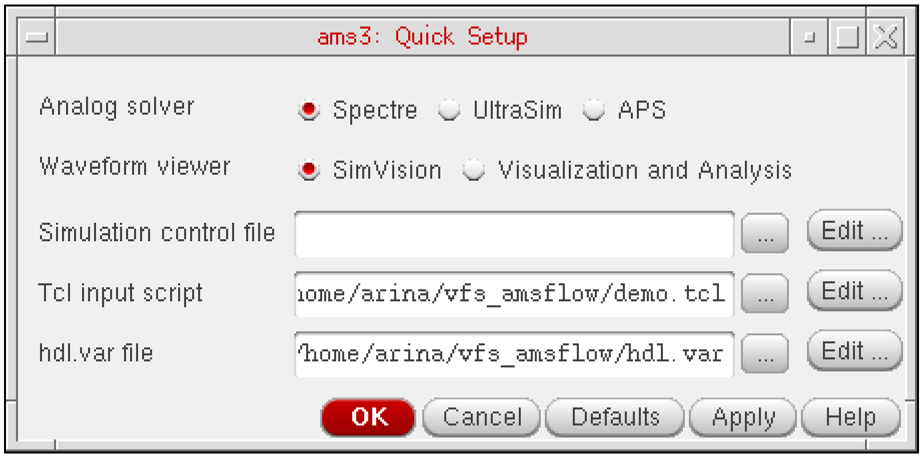
You can specify the following settings using the Quick Setup form:

- Analog solver
- Waveform viewer
- Simulation control file
- Tcl input script
- hdl.var file

To use the Quick Setup form, do the following:

1. Choose *AMS – Quick Setup*.

   The Quick Setup form appears.



ams3: Quick Setup

Analog solver: [✓ Spectre] UltraSim APS

Waveform viewer: [✓ SimVision] Visualization and Analysis

Simulation control file:

Tcl input script: ıome/arina/vfs_amsflow/demo.tcl

hdl.var file: /home/arina/vfs_amsflow/hdl.var

2. (Optional) Select one of the *Analog solver* radio buttons.

   See also "Specifying Analog Solver and Waveform Viewer" on page 100.

3. (Optional) Select one of the *Waveform viewer* radio buttons.

   See also "Specifying Analog Solver and Waveform Viewer" on page 100.

4. (Optional) Use the browse button to navigate to and select a *Simulation control file*.

When you click *Open* on the Choose form, the path and file name appear in the field.

5. (Optional) Use the browse button to navigate to and select a *Tcl input script*.

   When you click *Open* on the Choose form, the path and file name appear in the field.

   See also “Specifying a Tcl Input Script” on page 130.

6. (Optional) Use the browse button to navigate to and select an *hdl.var file*.

   When you click *Open* on the Choose form, the path and file name appear in the field.

   See also “Specifying an hdl.var File” on page 134.

7. Click *OK*.

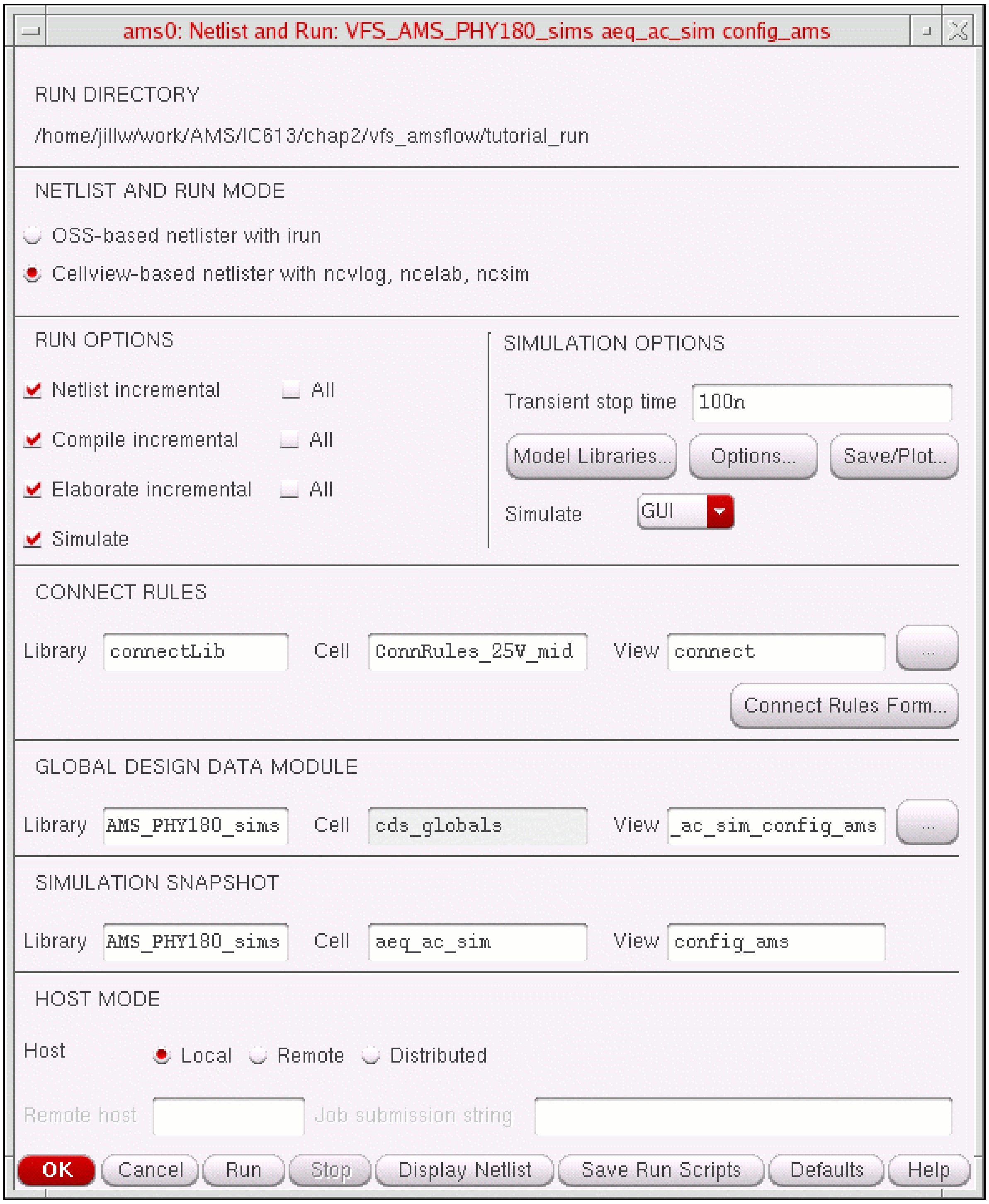
   You can specify additional setup options (for anything you did not specify in one of the files above) or, if everything is already specified, you can netlist and simulate directly.

3

Using the Netlist and Run Form

To open the Netlist and Run form, do the following:

➤ In the Virtuoso® Hierarchy Editor window, choose *AMS – Netlist and Run*.



ams0: Netlist and Run: VFS_AMS_PHY180_sims aeq_ac_sim config_ams

RUN DIRECTORY

/home/jillw/work/AMS/IC613/chap2/vfs_amsflow/tutorial_run

NETLIST AND RUN MODE

○ OSS-based netlister with irun
● Cellview-based netlister with ncvlog, ncelab, ncsim

RUN OPTIONS

✓ Netlist incremental ☐ All
✓ Compile incremental ☐ All
✓ Elaborate incremental ☐ All
✓ Simulate

SIMULATION OPTIONS

Transient stop time 100n

Simulate GUI

CONNECT RULES

Library connectLib Cell ConnRules_25V_mid View connect

GLOBAL DESIGN DATA MODULE

Library AMS_PHY180_sims Cell cds_globals View _ac_sim_config_ams

SIMULATION SNAPSHOT

Library AMS_PHY180_sims Cell aeq_ac_sim View config_ams

HOST MODE

Host ● Local ○ Remote ○ Distributed

Remote host Job submission string

The initialized run directory appears in the top section of the form.

You can use the Netlist and Run form to perform the following tasks:

- Specifying the Netlister and the Run Mode on page 88
- Specifying Run Options on page 89
- Specifying the Transient Stop Time on page 89
- Specifying Model Libraries for Simulation on page 90
- Specifying Simulation Options on page 90
- Specifying Outputs to Save and to Plot on page 90
- Specifying the Simulation Mode on page 91
- Specifying Connect Rules on page 92
- Specifying the Global Design Data Module on page 94
- Specifying the Simulation Snapshot Name and Location on page 94
- Specifying Local, Remote, or Distributed Simulation on page 95
- Using the Buttons at the Bottom of the Form on page 97

Specifying the Netlister and the Run Mode

Important

For OSS-based netlisting, you must run AMS from the Virtuoso® Analog Design Environment (ADE) by selecting *ams* as the simulator, then choosing *Simulation – Netlist and Run Options*.

To choose the netlist and run mode, do the following on the Netlist and Run form:

➤ In the *NETLIST AND RUN MODE* section, select one of the following:

- ❑ *OSS-based netlister with irun*

  See "Using the OSS Netlister" on page 179 for information about the OSS netlister. See "Using irun for AMS Simulation" in the *Virtuoso AMS Designer Simulator User Guide* for information about `irun` simulation.

- ❑ *Cellview-based netlister with ncvlog, ncelab, ncsim*

  See "Using the Cellview-Based Netlister" on page 180 for information about the cellview-based netlister. See the following topics in *Cadence NC-Verilog Simulator Help* for information about `ncvlog`, `ncelab`, and `ncsim`:

  - ❍ "Compiling Verilog Source Files with ncvlog"
  - ❍ "Elaborating the Design with ncelab"
  - ❍ "Simulating Your Design with ncsim"

Specifying Run Options

You can specify whether you want the program to perform full or incremental netlisting, compiling, and elaborating. You can also specify whether you want the program to simulate after performing these other tasks. AMS Designer performs the run options you select when you click *Run* (at the bottom of the Netlist and Run form).

To specify run options, do the following in the *RUN OPTIONS* section on the Netlist and Run form:

1. Select one of the following netlisting options:
   - ❑ *Netlist incremental* to netlist only new or changed cellviews
   - ❑ *All* to netlist the entire design
2. Select one of the following compilation options:
   - ❑ *Compile incremental* to compile only newly netlisted or changed cellviews
   - ❑ *All* to compile the entire design
3. Select one of the following elaboration options:
   - ❑ *Elaborate incremental* to run the elaborator only when you change something in your design that requires re-elaboration
   - ❑ *All* to elaborate the entire design without regard to whether you changed anything in your design that requires re-elaboration
4. If you want AMS Designer to simulate the design after performing the tasks you selected above, turn on *Simulate*.

AMS Designer performs these tasks when you click *Run*.

Specifying the Transient Stop Time

To specify a stop time for transient analysis, do the following in the *SIMULATION OPTIONS* section on the Netlist and Run form:

➤ In the *Transient stop time* field, type the transient stop time.

The simulator uses this stop time when you click *Run*.

Note: You can also specify a transient stop time in a simulation control file or by choosing *AMS – Detailed Setup – Analyses* and selecting *tran*.

Specifying Model Libraries for Simulation

To specify model libraries for simulation, do the following in the *SIMULATION OPTIONS* section on the Netlist and Run form:

1. Click *Model Libraries*.

   The Model Library Setup form appears.

   Note: You can also open this form by choosing *AMS – Detailed Setup – Model Libraries*.

2. Fill out this form and click *OK*.

Specifying Simulation Options

To specify simulation options, do the following in the *SIMULATION OPTIONS* section on the Netlist and Run form:

1. Click *Options*.

   The AMS Options form appears.

   Note: You can also open this form by choosing *AMS – Detailed Setup – AMS Options.*

2. Fill out this form and click *OK*.

Specifying Outputs to Save and to Plot

To specify outputs you want to save and plot, do the following in the *SIMULATION OPTIONS* section on the Netlist and Run form:

1. Click *Save/Plot*.

   The Setting Outputs form appears.

   Note: You can also open this form by choosing *AMS – Detailed Setup – Save/Plot Outputs*.

2. Fill out this form and click *OK*.

Specifying the Simulation Mode

To specify the simulation mode, do the following in the *SIMULATION OPTIONS* section on the Netlist and Run form:

➤ Using the drop-down combo box in the *Simulate* field, select one of the following simulation modes:

- ❑ *GUI* for interactive simulation using SimVision

  You can interact with the simulator using buttons, menus, and Tcl commands. Waveforms for any signals you selected for saving and plotting march as the simulation progresses.

- ❑ *Tcl* to use the Tcl interface to the simulator

  The Tcl interface window appears. The program saves but does not plot waveforms for any signals you selected for saving and plotting.

  - ❍ If you have not specified a Tcl input script, a window opens and waits for you to type a Tcl command.
  - ❍ If you have specified a Tcl input script, the script runs as soon as the window opens. If the script contains an `exit` or `finish` command, the window closes after the script runs. If the script does not cause the simulator to exit, the window remains open, waiting for you to type a Tcl command.

  To close the Tcl interface, type `exit` or `finish`.

  For a description of the Tcl commands you can use, see Appendix B, "Tcl-Based Debugging," in the *Virtuoso AMS Designer Simulator User Guide*.

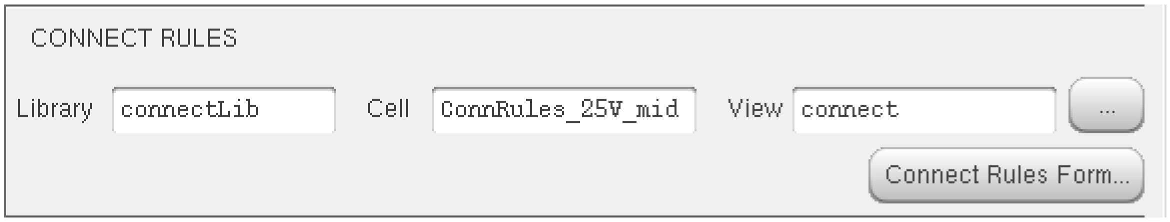
- ❑ *Batch* for background simulation

  You cannot interact with the simulator. Any signals you selected for saving and plotting appear in the waveform window at the end of the simulation.

  Note: *Batch* mode typically simulates more quickly than the other modes.

Specifying Connect Rules

If you have specified one or more connect rules for your design, the topmost set of rules appears in the *Library*, *Cell*, and *View* fields in the *CONNECT RULES* section on the Netlist and Run form.


CONNECT RULES

Library	connectLib	Cell	ConnRules_25V_mid	View	connect	...

In the *CONNECT RULES* section on the Netlist and Run form, you can:

- Specify or change a single connect rule
- Specify more than one connect rule or customize connect rules

Adding or Changing a Single Connect Rule

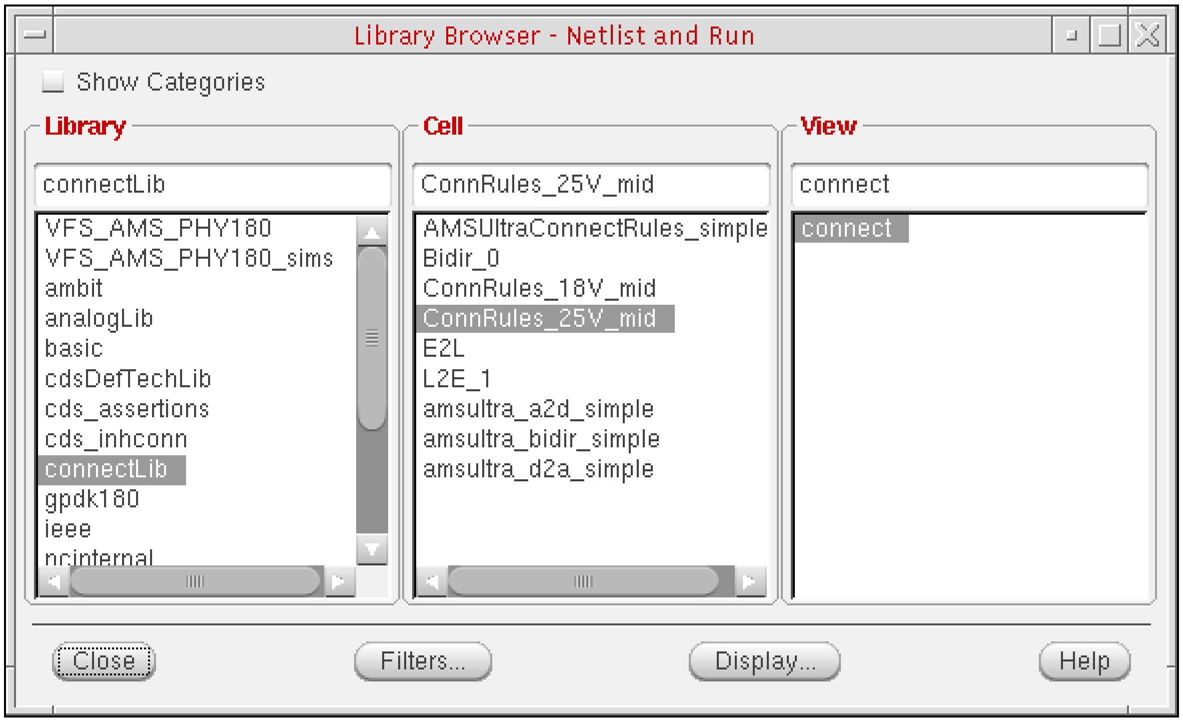
To add or change a single connect rule, do the following on the Netlist and Run form:

Important

Whatever connect rules you specify using this method not only appear in the *Library*, *Cell*, and *View* fields in the *CONNECT RULES* section on the Netlist and Run form, but also replaces the topmost set of connect rules in the *List of Connect Rules Used in Simulation* table on the Select Connect Rules form.

1. In the *CONNECT RULES* section, click the browse button to the right of the *Library*, *Cell*, and *View* fields.

The Library Browser – Netlist and Run form appears.



Library	Cell	View
connectLib	ConnRules_25V_mid	connect
VFS_AMS_PHY180	AMSUltraConnectRules_simple	connect
VFS_AMS_PHY180_sims	Bidir_0	
ambit	ConnRules_18V_mid	
analogLib	ConnRules_25V_mid	
basic	E2L	
cdsDefTechLib	L2E_1	
cds_assertions	amsultra_a2d_simple	
cds_inhconn	amsultra_bidir_simple	
connectLib	amsultra_d2a_simple	
gpdk180		
ieee		
ncinternal		

2. Select the connect rules you want to use.

   The selected connect rules appear in the *Library*, *Cell*, and *View* fields in the *CONNECT RULES* section on the Netlist and Run form.

3. On the Library Browser – Netlist and Run form, click *Close*.

Specifying More than One Set of Connect Rules or Customizing Rules

If you plan to use only one set of connect rules, see "Adding or Changing a Single Connect Rule" on page 92.

To specify more than one set of connect rules and to customize connect rules use the Select Connect Rules form by doing the following on the Netlist and Run form:

1. In the *CONNECT RULES* section, click *Connect Rules Form*.

   The Select Connect Rules form appears.

   Note: You can also open this form by choosing *AMS – Detailed Setup – Connect Rules*. See additional information in "Specifying Connect Rules" on page 114. See also important information in "Specifying Preferences for Netlisting and Compiling" on page 73.

2. Fill out this form and click *OK*.

   The topmost set of connect rules from the *List of Connect Rules Used in Simulation* table appears in the *Library*, *Cell*, and *View* fields in the *CONNECT RULES* section on the Netlist and Run form.

Specifying the Global Design Data Module

For information about how to use the Netlist and Run form to specify a global design data module, see "Global Design Data Module (cds_globals)" on page 200.

Specifying the Simulation Snapshot Name and Location

By default, the simulation snapshot name is the same as the `library.cell:view` of the top-level design unit you are simulating. To specify a different name and location for the simulation snapshot, in the *SIMULATION SNAPSHOT* section on the Netlist and Run form:

1. In the *View* field, type a new simulation snapshot name.
2. (Optional) In the *Cell* field, type a new cell location for the simulation snapshot.
3. (Optional) In the *Library* field, type a valid library name where you want the program to store the simulation snapshot.

For more information about simulation snapshots, see "-snapshot" in the "Elaborating the Design with ncelab" chapter of *NC-Verilog Simulator Help* for more information.

Specifying Local, Remote, or Distributed Simulation

You can run simulations on the local host, on a remote host, or distributed across a number of hosts. See the following topics for details:

- Specifying Local Simulation on page 95
- Specifying Remote Simulation on page 95
- Specifying Distributed Simulation on page 96

See also "Important Information about Remote and Distributed Simulations" on page 96.

Specifying Local Simulation

To specify local simulation, do the following in the *HOST MODE* section on the Netlist and Run form:

➤ For *Host*, select *Local*.

Simulations occur on your local host machine.

Specifying Remote Simulation

Important

Your simulation mode must be *Batch* in order to support remote simulation. See also "Important Information about Remote and Distributed Simulations" on page 96.

To specify remote simulation, do the following in the *HOST MODE* section on the Netlist and Run form:

1. For *Host*, select *Remote*.

   Note: You cannot select *Remote* if your simulation mode is *GUI* or *Tcl*. You must select *Batch* for the simulation mode.

   The *Remote host* field becomes active.

2. In the *Remote host* field, type the name of the remote host on which you want to run simulations.

   The program compiles and elaborates your design on the remote host before starting the simulation.

Specifying Distributed Simulation

Important

Your simulation mode must be *Batch* in order to support distributed simulation. See also “Important Information about Remote and Distributed Simulations” on page 96.

To specify distributed simulation, do the following in the *HOST MODE* section on the Netlist and Run form:

1. For *Host*, select *Distributed*.

   Note: You cannot select *Distributed* if your simulation mode is *GUI* or *Tcl*. You must select *Batch* for the simulation mode.

   The *Job submission string* field becomes active.

2. In the *Job submission string* field, type a job submission command string that is valid for your resource management software.

   The software prefixes the job submission string to the command line.

   The program compiles and elaborates your design on the chosen host before starting the simulation.

Tip

You can use the full command string (with the job submission string prefix) to submit batch jobs to a compute farm.

Important Information about Remote and Distributed Simulations

When you select *Remote* or *Distributed* simulation, the tool paths on the remote hosts must be the same as those on your local host. For example, if the path to the MMSIM hierarchy on your local host machine is `/usr/cad/MMSIM`, the path to the MMSIM hierarchy on all remote machines on which your simulations will run must also be `/usr/cad/MMSIM`.

If the paths are different—such as

```
/usr/cad/MMSIM
```

on the local host and

```
/usr3/mnt/cad/MMSIM
```

on a remote host—you can create a link so that the tool paths on the remote host appear to match the tool paths on your local host.

For example, you can create a link on the remote host at `/usr/cad/MMSIM` that points to `/usr3/mnt/cad/MMSIM`.

Using the Buttons at the Bottom of the Form

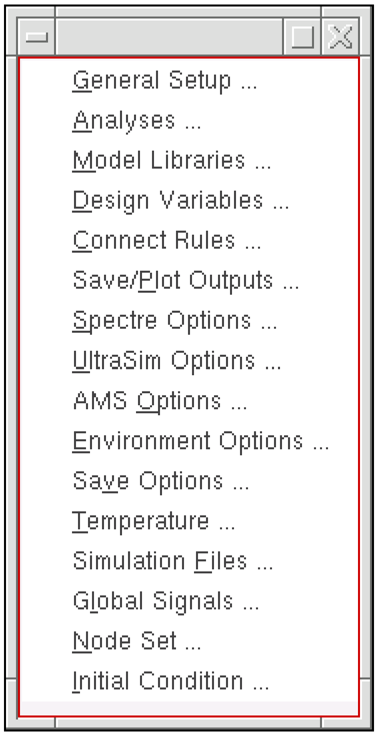
The buttons at the bottom of the Netlist and Run form perform tasks as indicated:

Button	Function
OK	Saves your settings and closes the form without performing any tasks
Cancel	Closes the form without performing any tasks or saving any settings
Run	Performs all tasks specified on the form
Stop	Terminates the simulation **Note:** This button is only active during simulation.
Display Netlist	Netlists the design, if required, and displays the netlist in a viewing window (`runDirectory/netlist/completeDesignInfo.ckt`) **Note:** See also "Displaying the Netlist" on page 170.
Save Run Scripts	Generates a valid netlist, then generates and saves run scripts (such as `runElabSim` and `runCompileElabSim`) to your `runDirectory/netlist` directory
Defaults	Restores all values on the form to their default values
Help	Opens the Cadence Help topic page for this form

4

Using the Detailed Setup Menu

You can use the *AMS – Detailed Setup* menu to set up all aspects of your simulation, from the analysis specification to outputs you want to plot. You can specify the model libraries you want to use, any connect rules you want to apply, and values for design variables. You can specify options for the AMS Designer simulator and its analog solvers (Spectre or UltraSim). You can specify global signals, nodesets, and initial conditions.



See the following topics for more information:

- Specifying Analog Solver and Waveform Viewer on page 100
- Specifying an Analysis on page 101
- Specifying Model Libraries on page 113
- Specifying Design Variables on page 113

Note: The Virtuoso® AMS Designer environment shares many of the detailed setup forms with the Virtuoso Analog Design Environment. You can read more about the options you can set on these forms in the *Virtuoso Analog Design Environment L User Guide*, the *Virtuoso Spectre Circuit Simulator User Guide*, the *Virtuoso UltraSim Simulator User Guide*, and the *Virtuoso AMS Designer Simulator User Guide*.

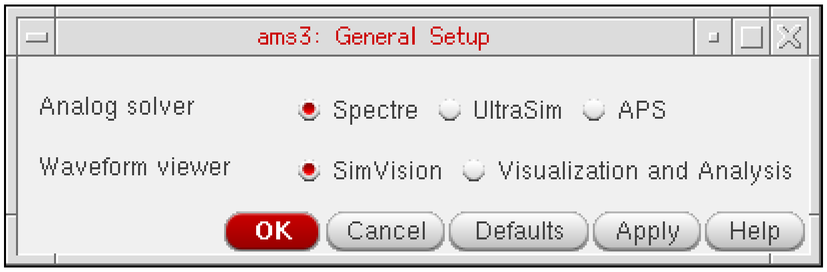
Specifying Analog Solver and Waveform Viewer

You can use the General Setup form to specify the analog solver and the waveform viewer.

To use the General Setup form, do the following:

1. Choose *AMS – Detailed Setup – General Setup*.

   The General Setup form appears.



Analog solver [✓ Spectre] UltraSim APS

Waveform viewer [✓ SimVision] Visualization and Analysis

2. Select one of the *Analog solver* radio buttons.
3. Select one of the *Waveform viewer* radio buttons.
4. Click *OK*.

These options are also available on the Quick Setup form.

Specifying an Analysis

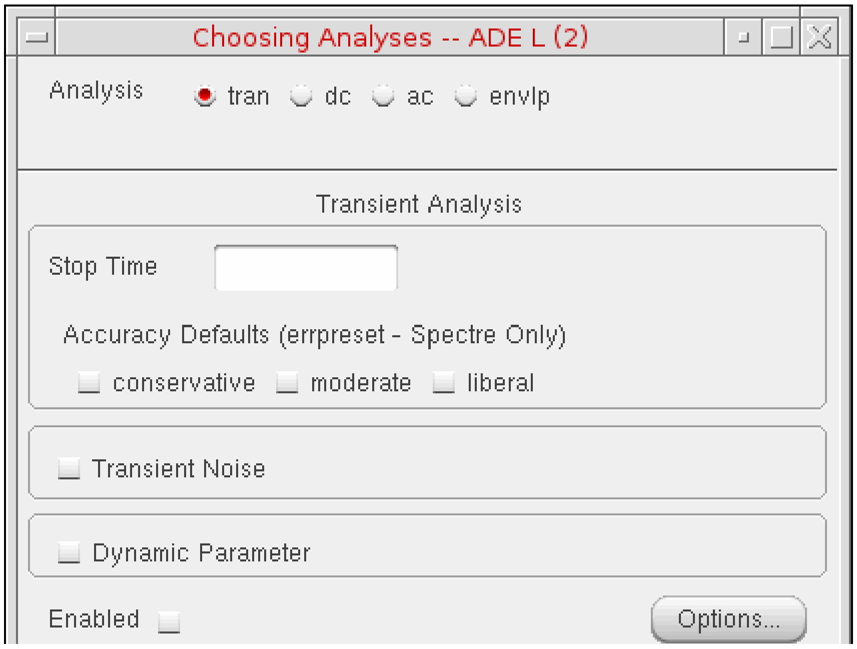
To specify an analysis, do the following:

1. Choose *AMS – Detailed Setup – Analyses*.

   The Choosing Analyses form appears.

   The appearance of this form depends on which solver you have selected on the General Setup form.

   If you have the Spectre solver selected, the Choosing Analyses form looks like this:



Choosing Analyses -- ADE L (2)

Analysis [✓ tran] dc ac envlp

Transient Analysis

Stop Time

Accuracy Defaults (errpreset - Spectre Only)

conservative moderate liberal

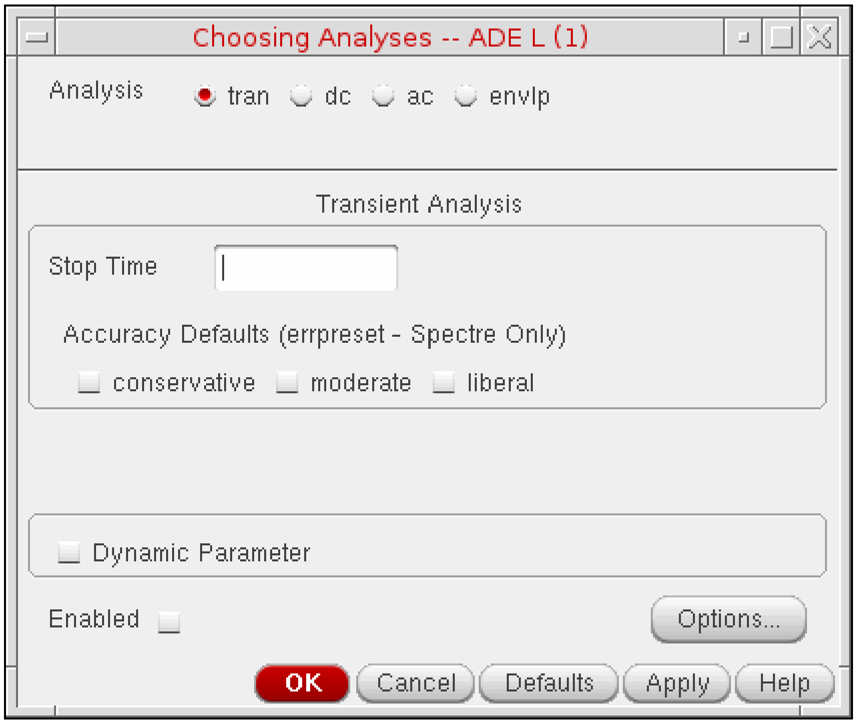
Transient Noise

Dynamic Parameter

Enabled

Options...

If you have the UltraSim solver selected, the Choosing Analyses form looks like this:



Analysis: [✓ tran] dc ac envlp

2. Specify analysis setup information and click *OK*.

For specific analysis setup instructions, see the following sections:

- ❑ Specifying a Transient Analysis on page 103
- ❑ Specifying a DC Analysis on page 106
- ❑ Specifying an AC Analysis (Spectre Solver Only) on page 107
- ❑ Specifying an Envelope Analysis on page 109

For detailed information about the various analysis settings, see the relevant User Guide:

- ❑ For information about Spectre circuit simulator settings, see the Analyses chapter of the *Virtuoso Spectre Circuit Simulator User Guide*.
- ❑ For information about UltraSim circuit simulator settings, see the *Virtuoso UltraSim Simulator User Guide*.

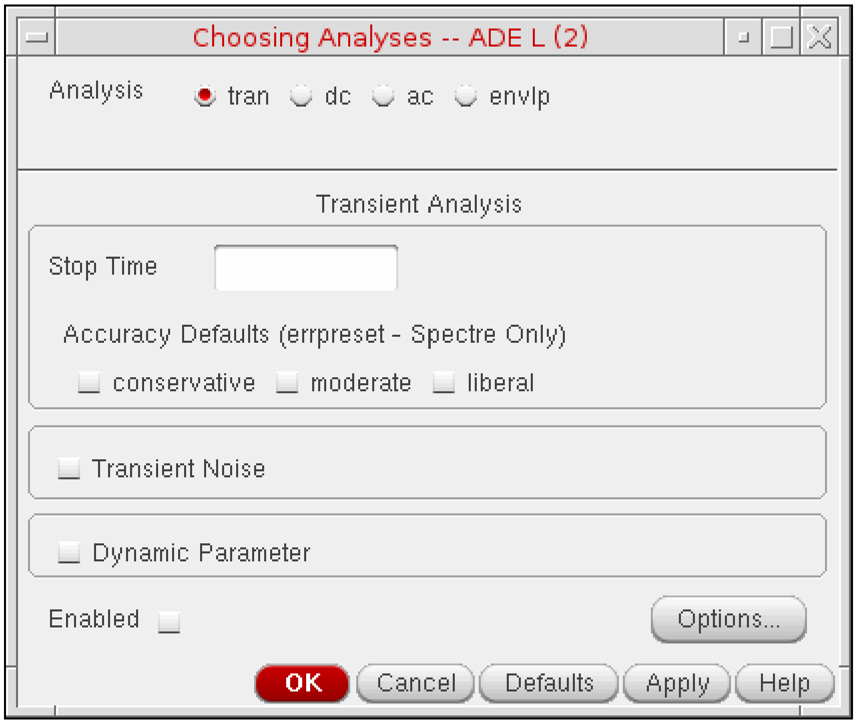
See also “Specifying Controls for the Analog Solvers” in the *Virtuoso AMS Designer Simulator User Guide*.

Specifying a Transient Analysis

To specify a transient analysis, do the following on the Choosing Analyses form:

1. For *Analysis*, select *tran*.

   The Choosing Analyses form looks like this:



Choosing Analyses -- ADE L (2)

Analysis [✓ tran] dc ac envlp

Transient Analysis

Stop Time

Accuracy Defaults (errpreset - Spectre Only)

☐ conservative ☐ moderate ☐ liberal

☐ Transient Noise

☐ Dynamic Parameter

Enabled ☐

   For details about the settings on this form for the Spectre solver, see "Transient Analysis" in the *Virtuoso Analog Design Environment L User Guide*. For the UltraSim solver, see the relevant section in the "Setting Up for an Analysis" chapter of the *Virtuoso Analog Design Environment L User Guide*.

   See also "Specifying Controls for the Analog Solvers" in the *Virtuoso AMS Designer Simulator User Guide*.

2. In the *Stop Time* field, type a transient analysis stop time.

   If you press *Tab* at this point, a check mark appears in the *Enabled* check box.

3. (Spectre solver only) For *Accuracy Defaults*, select one of the following:

   ❑ *conservative*

   ❑ *moderate*

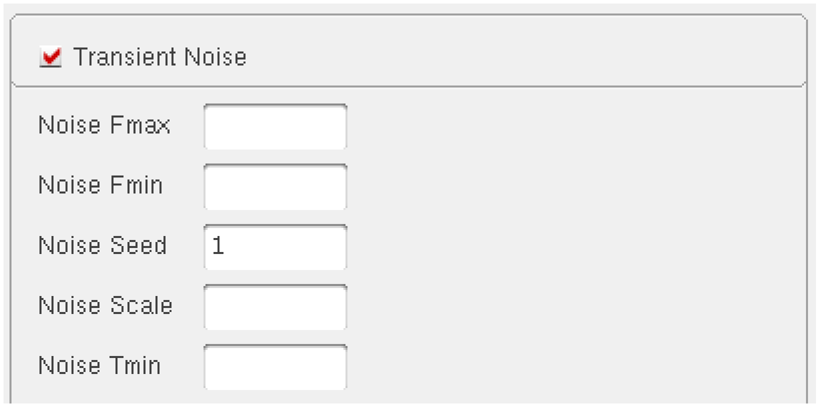
- ❑ *liberal*

  For details about these settings, see “Description of errpreset Parameter Settings” in the Analyses chapter of the *Virtuoso Spectre Circuit Simulator User Guide*.

4. (Spectre solver only) (Optional) To run a transient noise analysis, turn on *Transient Noise*.

5. Select the Dynamic Parameter check box to vary temperature, design parameters, options, or transient analysis parameters (such as reltol, residualtol, vabstol, iabstol, isnoisy) during transient simulation.

Additional fields appear on the form so that you can specify transient noise parameters.



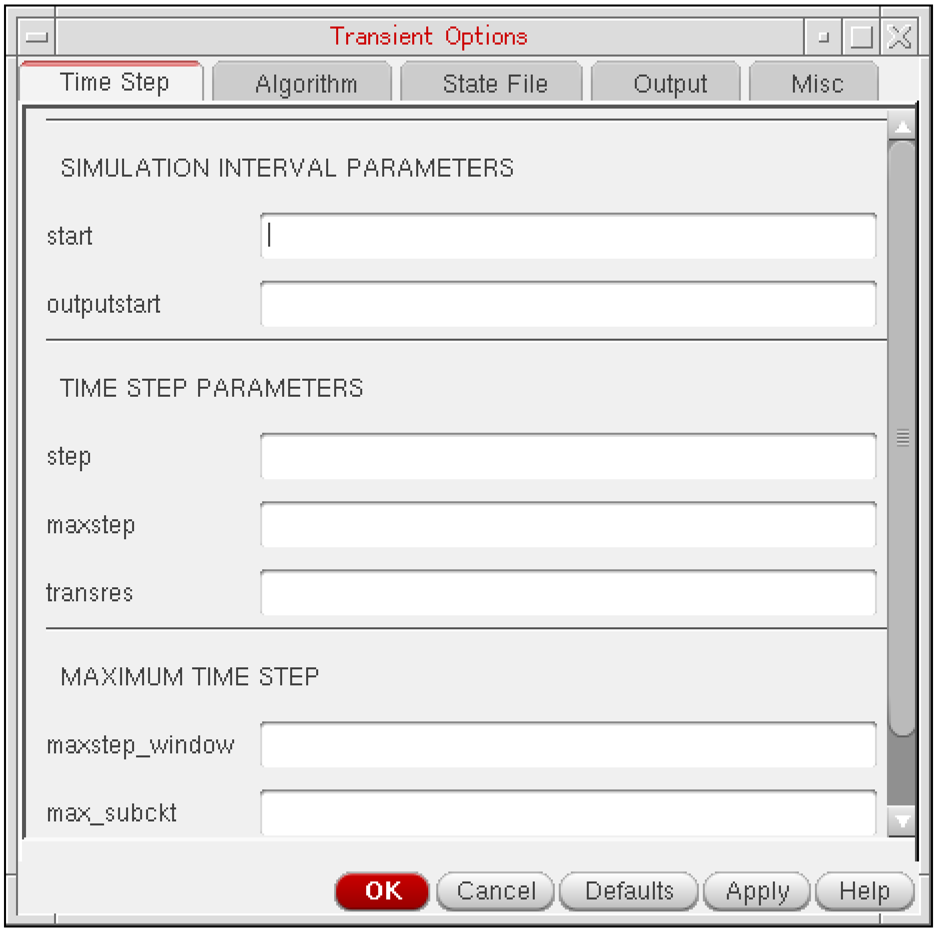
✓ Transient Noise

Noise Fmax	
Noise Fmin	
Noise Seed	1
Noise Scale	
Noise Tmin	

For details about transient noise analysis and these settings, see “Calculating Transient Noise” in the Analyses chapter of the *Virtuoso Spectre Circuit Simulator User Guide*.

6. (Optional) To specify transient options, click *Options*.

The Transient Options form appears.



SIMULATION INTERVAL PARAMETERS

start	
outputstart	

TIME STEP PARAMETERS

step	
maxstep	
transres	

MAXIMUM TIME STEP

maxstep_window	
max_subckt	

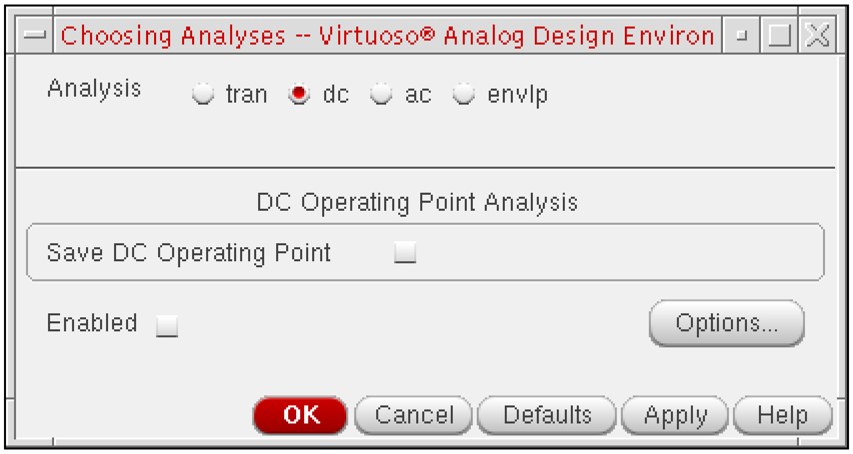
7. Click *OK*.

Specifying a DC Analysis

To specify a DC analysis, do the following on the Choosing Analyses form:

1. For *Analysis*, select *dc*.

   The Choosing Analyses form looks like this:



Choosing Analyses -- Virtuoso® Analog Design Environ

Analysis ○ tran [✓ ● dc] ○ ac ○ envlp

DC Operating Point Analysis

Save DC Operating Point ☐

Enabled ☐

   For details about the settings on this form for the Spectre solver, see "DC Analysis" in the *Virtuoso Analog Design Environment L User Guide*. For the UltraSim solver, see the relevant section in the "Setting Up for an Analysis" chapter of the *Virtuoso Analog Design Environment L User Guide*.

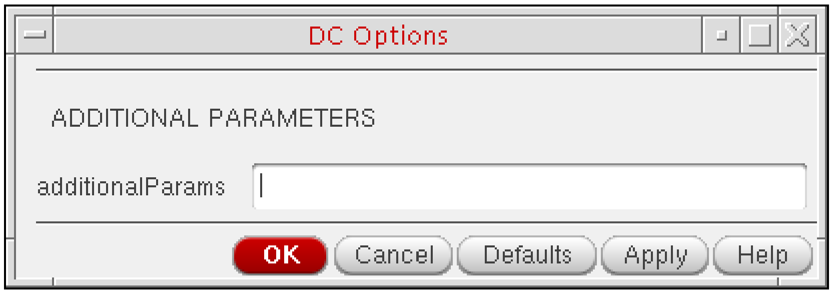
   See also "Mixed-Signal DC Initialization" in the *Virtuoso AMS Designer Simulator User Guide*.

2. To enable the DC analysis, turn on the *Enabled* check box.

3. (Optional) To enable DC operating point analysis, turn on the *Save DC Operating Point* check box.

4. (Optional) To specify DC options, click *Options*.

   The DC Options form appears.



DC Options

ADDITIONAL PARAMETERS

additionalParams

5. Click *OK*.

Specifying an AC Analysis (Spectre Solver Only)

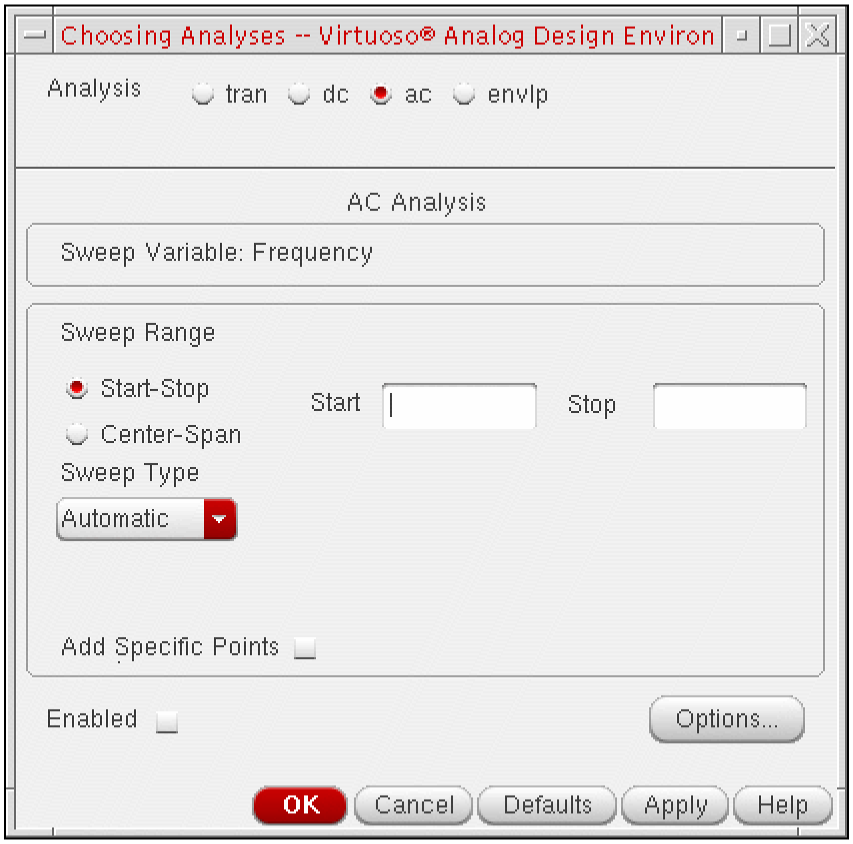
Important

You can only run AC analysis using the Spectre solver. The UltraSim solver does not support AC analysis.

To specify an AC analysis for the Spectre solver, do the following on the Choosing Analyses form:

1. For *Analysis*, select *ac*.

   The Choosing Analyses form looks like this:



Choosing Analyses -- Virtuoso® Analog Design Environ

Analysis: tran, dc, [✓ ac], envlp

AC Analysis

Sweep Variable: Frequency

Sweep Range: [✓ Start-Stop] Center-Span

Start | Stop

Sweep Type: Automatic

Add Specific Points ☐

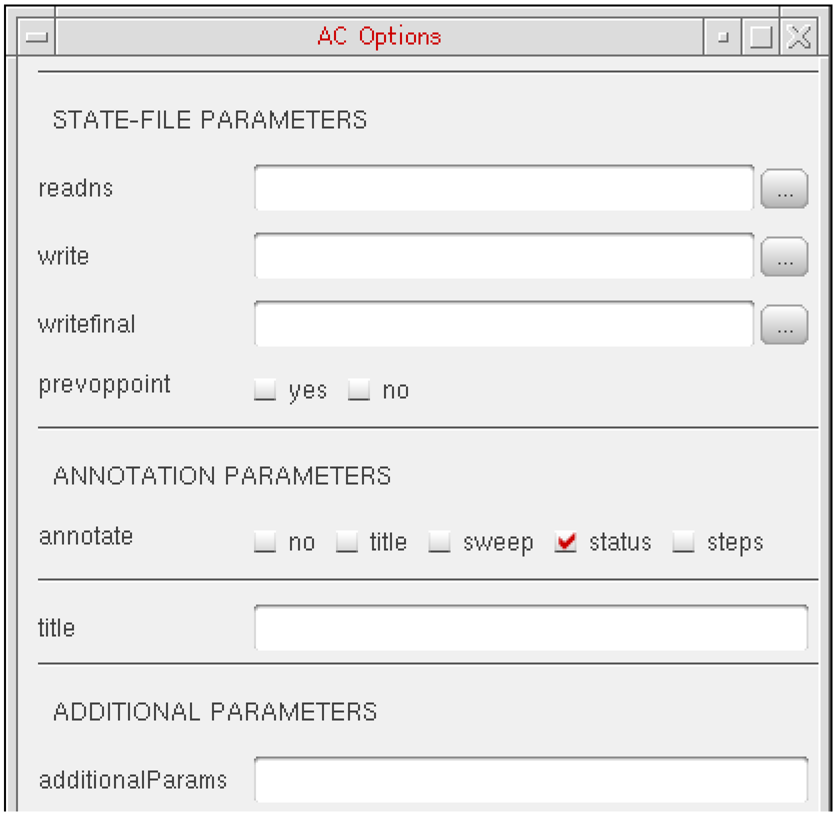
Enabled ☐

   For details about the settings on this form, see "AC Analysis" in the *Virtuoso Analog Design Environment L User Guide*.

   See also "AC Analysis (ac)" in the *Virtuoso AMS Designer Simulator User Guide*.

2. To enable the AC analysis, turn on the *Enabled* check box.
3. (Optional) To specify additional parameters for AC analysis, click *Options*.

   The AC Options form appears.



AC Options

STATE-FILE PARAMETERS

readns	
write	
writefinal	
prevoppoint	☐ yes ☐ no

ANNOTATION PARAMETERS

annotate	☐ no ☐ title ☐ sweep ☑ status ☐ steps
title	

ADDITIONAL PARAMETERS

additionalParams	

   For information about the options you can specify on this form, see “AC Analysis (ac)” in the *Virtuoso Spectre Circuit Simulator Reference*.

   a. Specify the options you want to use.
   b. Click *OK*.

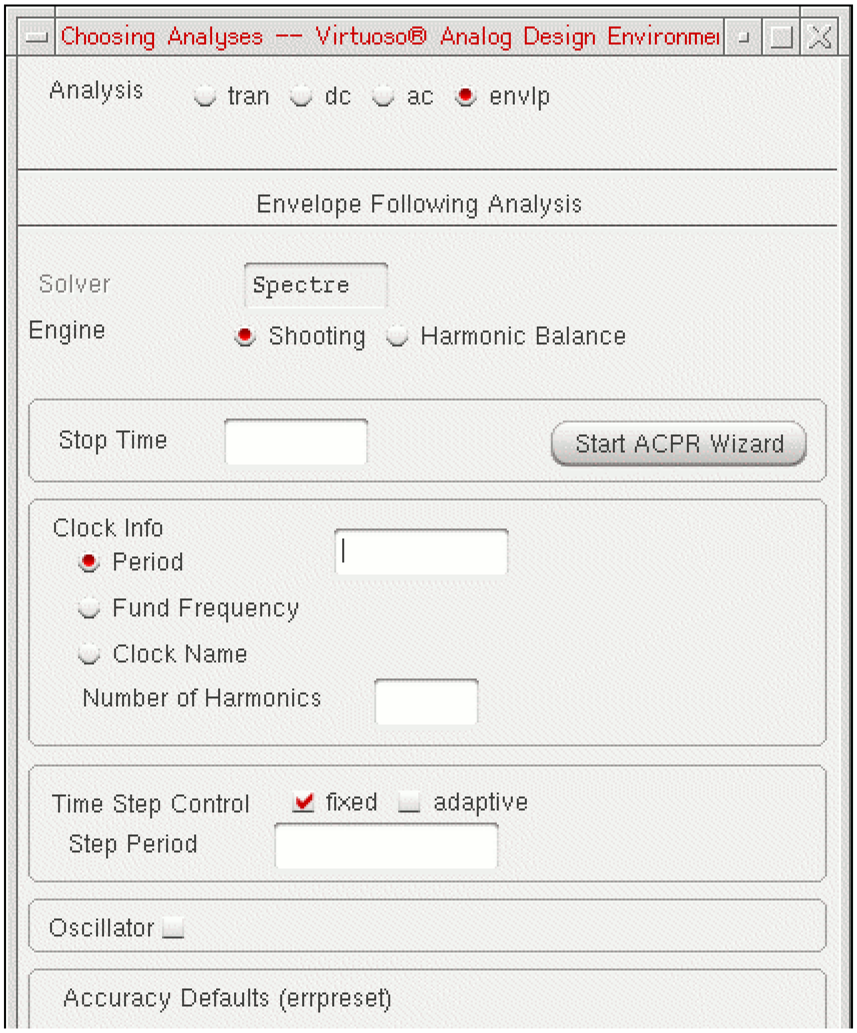
4. On the Choosing Analyses form, click *OK*.

Specifying an Envelope Analysis

To specify an envelope analysis, do the following on the Choosing Analyses form:

1. For *Analysis*, select *envlp*.
2. Under *Envelope Following Analysis*, select your solver (*Spectre* or *UltraSim*).

   The Choosing Analyses form for Spectre looks like this:



Choosing Analyses -- Virtuoso® Analog Design Environme

Analysis: ○ tran ○ dc ○ ac ● envlp

Envelope Following Analysis

Solver: Spectre

Engine: ● Shooting ○ Harmonic Balance

Stop Time

Start ACPR Wizard

Clock Info
● Period
○ Fund Frequency
○ Clock Name
Number of Harmonics

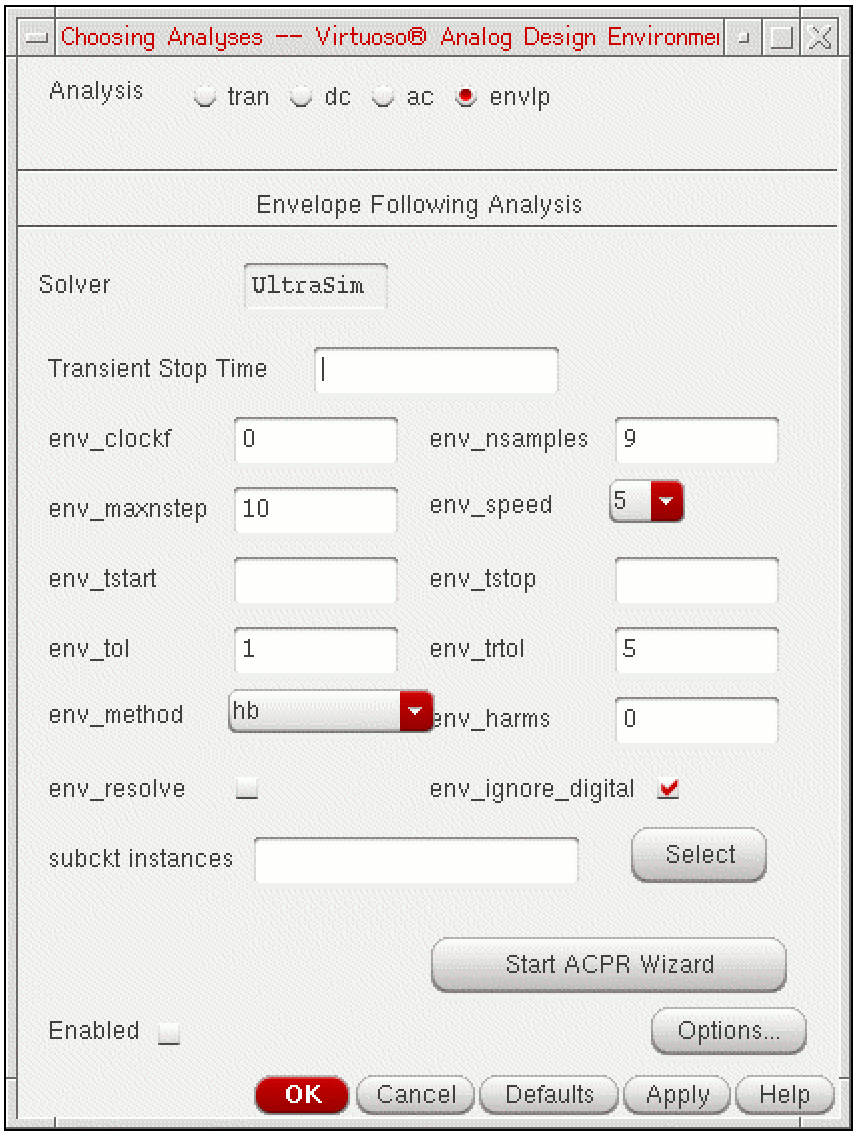
Time Step Control: ✓ fixed ☐ adaptive
Step Period

Oscillator ☐

Accuracy Defaults (errpreset)

For details about the settings on this form (for the Spectre solver), see “Envelope Following Analysis (envlp)” in the *Virtuoso Spectre Circuit Simulator Reference*.

The Choosing Analyses form for UltraSim looks like this:



Choosing Analyses -- Virtuoso® Analog Design Environme

Analysis: tran, dc, ac, [✓ envlp]

Envelope Following Analysis

Solver	UltraSim		
Transient Stop Time			
env_clockf	0	env_nsamples	9
env_maxnstep	10	env_speed	5
env_tstart		env_tstop	
env_tol	1	env_trtol	5
env_method	hb	env_harms	0
env_resolve		env_ignore_digital	✓
subckt instances			
Enabled			

For details about the settings on this form (for the UltraSim solver), see the relevant section in the "Setting Up for an Analysis" chapter of the *Virtuoso Analog Design Environment L User Guide*.

See also "Envelope Analysis (envlp)" in the *Virtuoso AMS Designer Simulator User Guide*.

3. To enable the analysis, turn on the *Enabled* check box.

4. (Optional) To specify adjacent channel power ratio (ACPR) information, click *Start ACPR*.

The ACPR Wizard form appears.

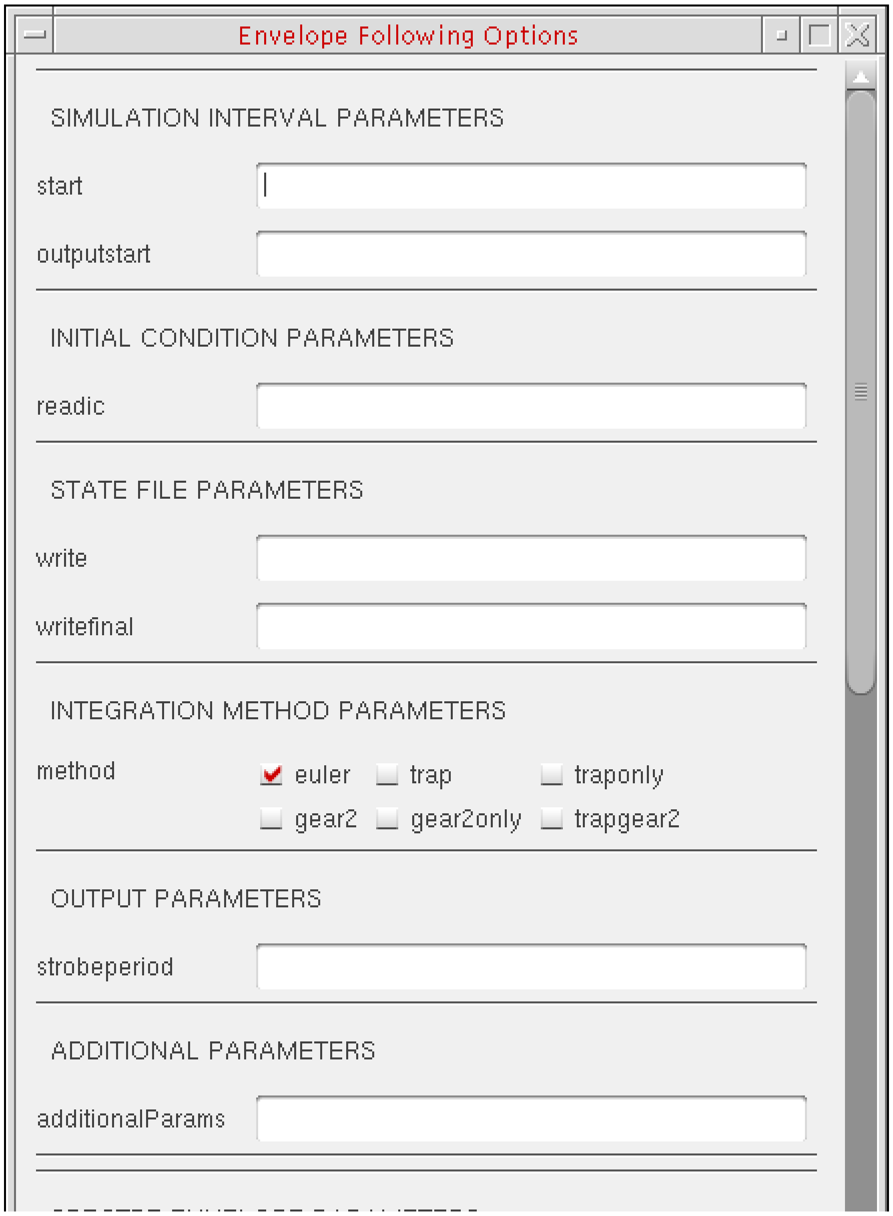
For information about this form, see the relevant section in the "Setting Up for an Analysis" chapter of the *Virtuoso Analog Design Environment L User Guide*.

a. Specify the information you want.

b. Click *OK*.

5. (Optional) To specify additional parameters for the analysis, click *Options*.

The Envelope Following Options form appears.



Envelope Following Options

SIMULATION INTERVAL PARAMETERS

start	
outputstart	

INITIAL CONDITION PARAMETERS

readic	

STATE FILE PARAMETERS

write	
writefinal	

INTEGRATION METHOD PARAMETERS

method [✓ euler] trap traponly gear2 gear2only trapgear2

OUTPUT PARAMETERS

strobeperiod	

ADDITIONAL PARAMETERS

additionalParams	

For information about the options you can specify on this form, see the relevant section in the “Setting Up for an Analysis” chapter of the *Virtuoso Analog Design Environment L User Guide*.

a. Specify the options you want to use.

b. Click *OK*.

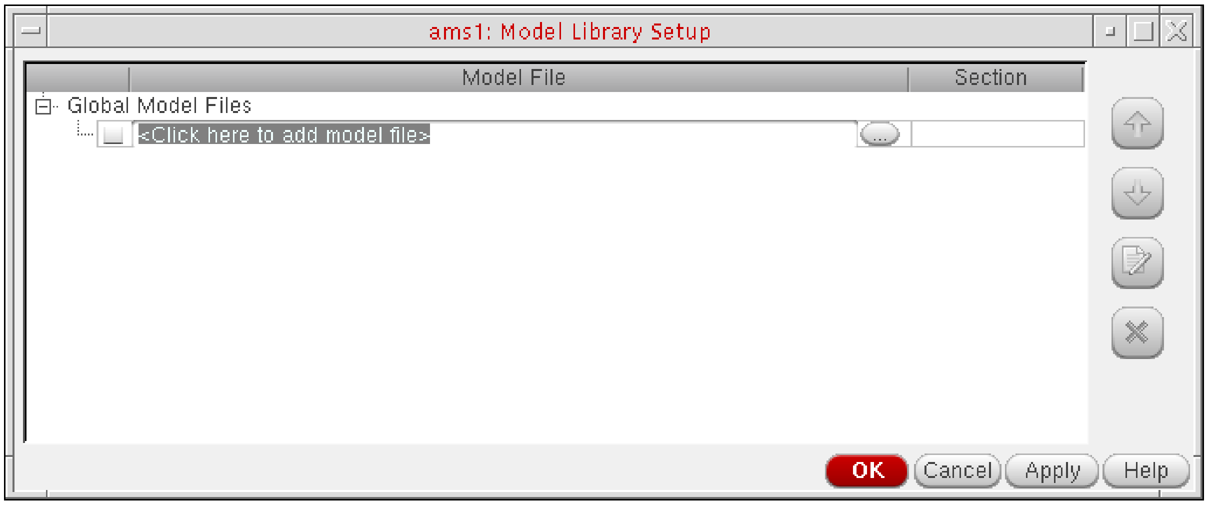
6. On the Choosing Analyses form, click *OK*.

Specifying Model Libraries

To specify the model libraries you want to use for simulation, do the following:

1. Choose *AMS – Detailed Setup – Model Libraries*.

   The Model Library Setup form appears.



   For information about this form, see "Model Library Setup" in the *Virtuoso Analog Design Environment L User Guide*.

2. Specify your model library information and click *OK*.

Specifying Design Variables

To specify design variables, do the following:

1. Choose *AMS – Detailed Setup – Design Variables*.

   The Editing Design Variables form appears.

2. Use the form to specify your design variables, then click *OK*.

Specifying Connect Rules

To specify connect rules, do the following:

1. Choose *AMS – Detailed Setup – Connect Rules*.

   The Select Connect Rules form appears.

   For information about filling out this form, see "Setting Connect Rules" in the *Virtuoso Analog Design Environment L User Guide*.

2. Fill out the form and click *OK*.

   The set of rules you specify instructs the elaborator when to insert connect modules. For more information, see "Connect Modules" in the "Mixed-Signal Aspects of Verilog-AMS" chapter of the *Cadence Verilog-AMS Language Reference*.

   See additional information in "Specifying Connect Rules" on page 92. See also important information in "Specifying Preferences for Netlisting and Compiling" on page 73.

Specifying Outputs to Save and to Plot

To specify outputs you want to save and to plot, do the following:

1. Choose *AMS – Detailed Setup – Save/Plot Outputs*.

   The Setting Outputs form appears.

   For information about filling out this form, see Setting Outputs form in the *Virtuoso Analog Design Environment L User Guide*.

2. Fill out this form and click *OK*.

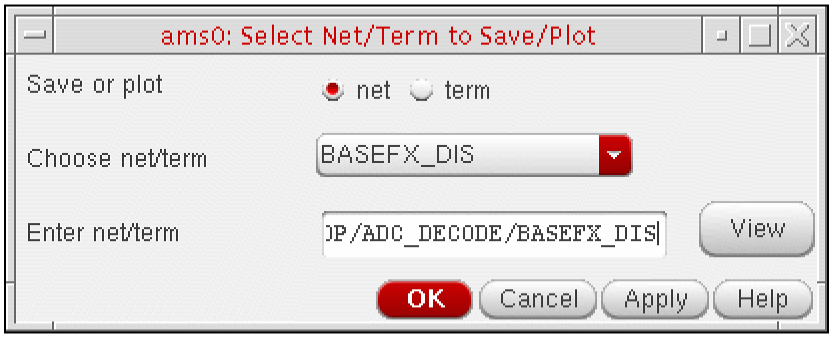
To select nets or terminals from the Virtuoso Hierarchy Editor (instead of from the schematic), do the following:

1. On the Setting Outputs form (see above), click *From HED*.

   The *Tree View* tab appears in the foreground.

2. Select an instance.

The Select Net/Term to Save/Plot form appears.



Save or plot [✓ net] term

Choose net/term BASEFX_DIS

Enter net/term)P/ADC_DECODE/BASEFX_DIS

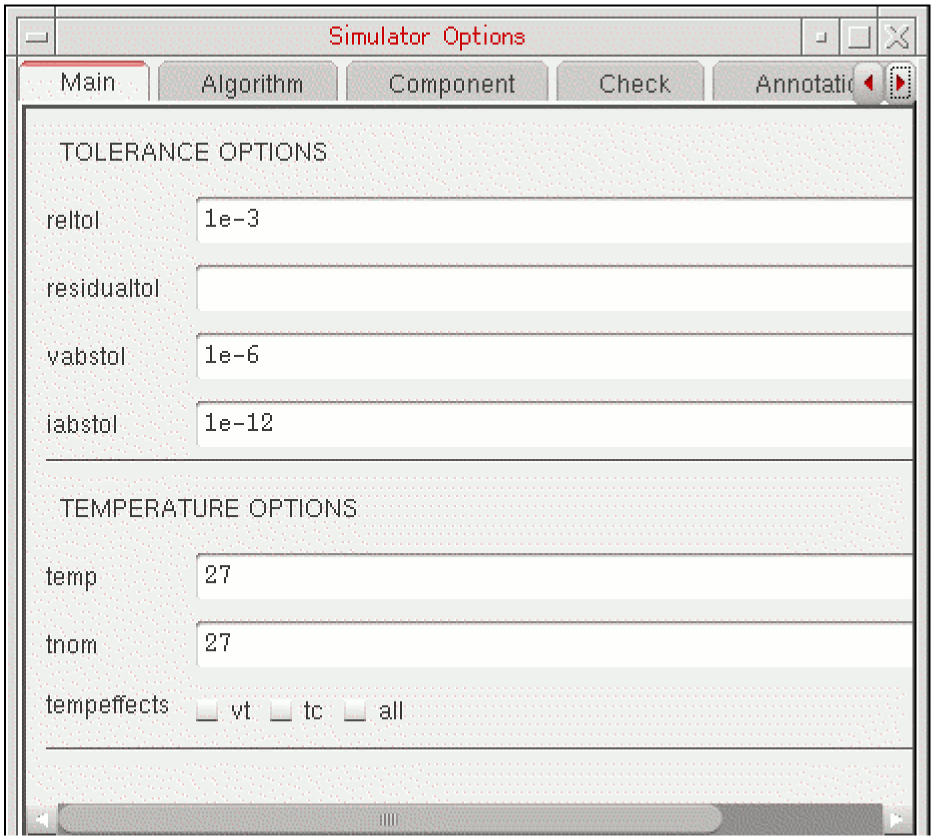
3. For each net you want to specify, do the following:
   a. For *Save or plot*, select *net*.
   b. In the *Choose net/term* field, choose the net you want to save and plot.
   c. Click *Apply*.
4. For each terminal you want to specify, do the following:
   a. For *Save or plot*, select *term*.
   b. In the *Choose net/term* field, choose the terminal you want to save and plot.
   c. Click *Apply*.
5. Click *OK*.

Specifying Spectre Options

To specify options for the Spectre solver, do the following:

1. Choose *AMS – Detailed Setup – Spectre Options*.

   The Analog (Spectre) Options form appears.



TOLERANCE OPTIONS

reltol	1e-3
residualtol	
vabstol	1e-6
iabstol	1e-12

TEMPERATURE OPTIONS

temp	27
tnom	27
tempeffects	☐ vt ☐ tc ☐ all

   For information about the options you can set using this form, see “Immediate Set Options (options)” in the “Analysis Statements” chapter of the *Virtuoso Spectre Circuit Simulator Reference*.

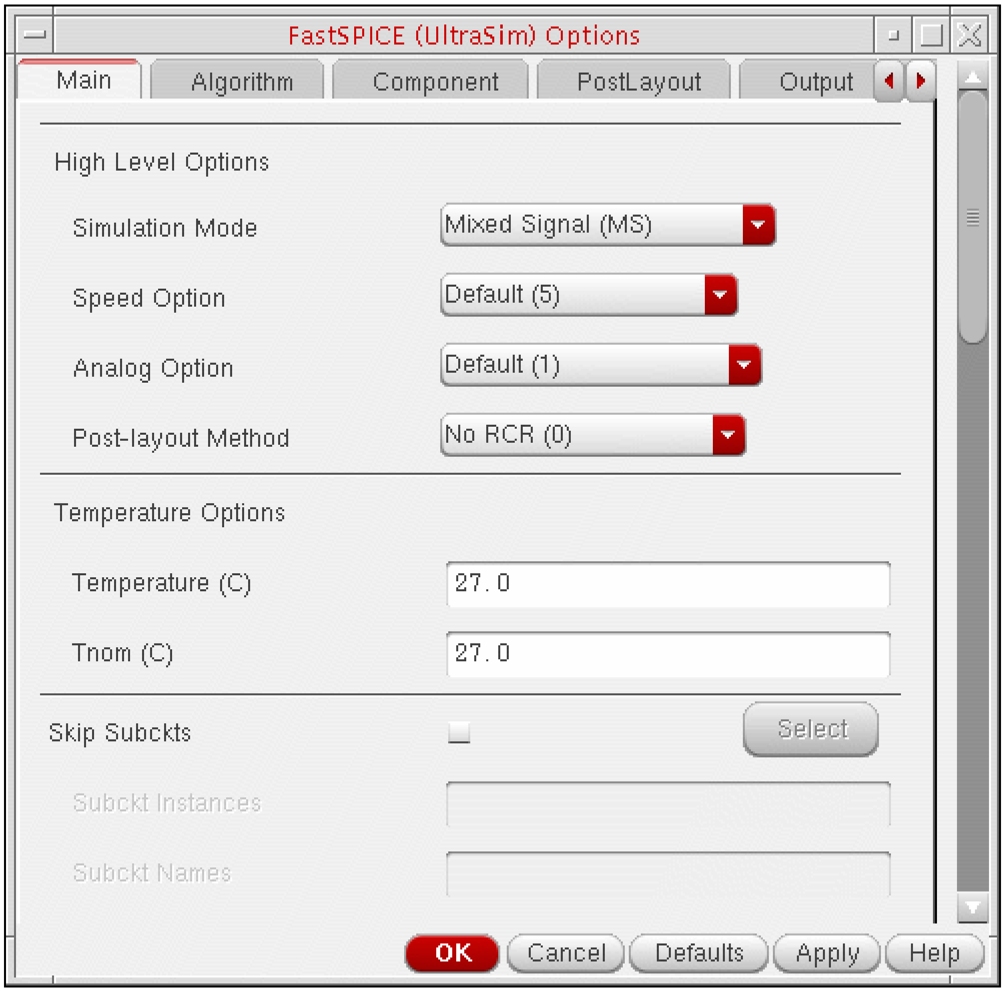
2. Fill out this form and click *OK*.

Specifying UltraSim Options

To specify options for the UltraSim solver, do the following:

1. Choose *AMS – Detailed Setup – UltraSim Options*.

   The FastSPICE (UltraSim) Options form appears.



High Level Options

Simulation Mode	Mixed Signal (MS)
Speed Option	Default (5)
Analog Option	Default (1)
Post-layout Method	No RCR (0)

Temperature Options

Temperature (C)	27.0
Tnom (C)	27.0
Skip Subckts	☐
Subckt Instances	
Subckt Names	

   For information about the options you can set using this form, see the *Virtuoso UltraSim Simulator User Guide*.

2. Fill out this form and click *OK*.

Specifying AMS Options

To specify options for the AMS Designer simulator, do the following:

1. Choose *AMS – Detailed Setup – AMS Options*.

   The AMS Options form appears.

   For information about the options you can set using this form, see Chapter 5, “Using the AMS Options Form.”

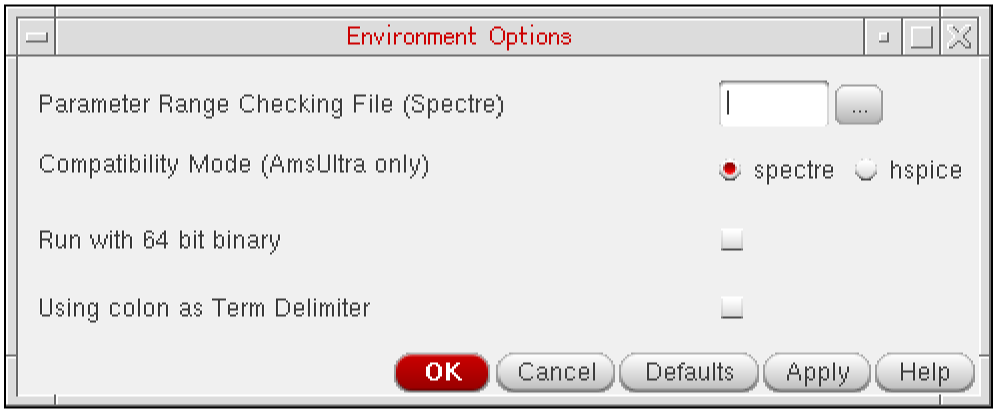
2. Fill out this form and click *OK*.

Specifying Environment Options

To specify environment options, do the following:

1. Choose *AMS – Detailed Setup – Environment Options*.

   The Environment Options form appears.



Parameter Range Checking File (Spectre)	
Compatibility Mode (AmsUltra only)	[✓ spectre] hspice
Run with 64 bit binary	☐
Using colon as Term Delimiter	☐

   For information about the options you can set using this form, see “Setting Environment Options for AMS” and “Environment Options” in the “Environment Setup” chapter of the *Virtuoso Analog Design Environment L User Guide*.

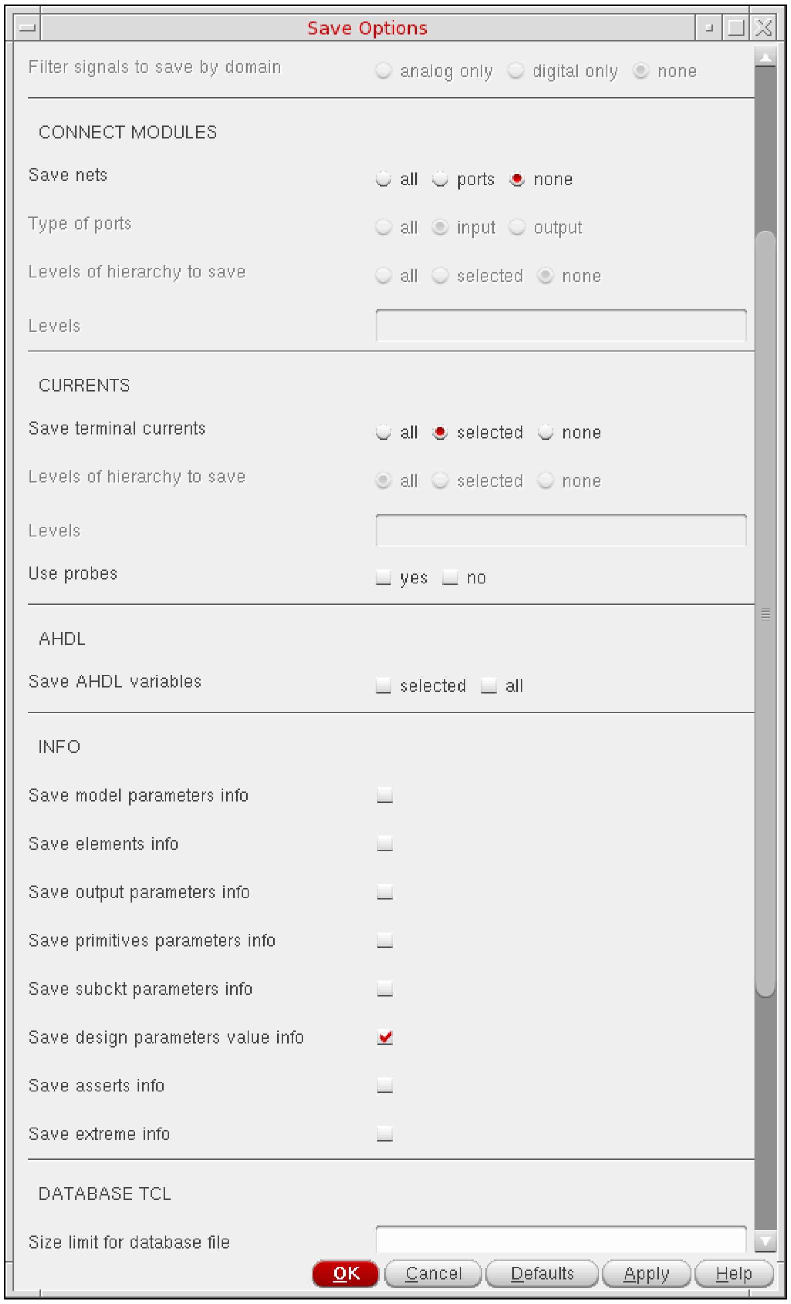
2. Fill out this form and click *OK*.

Specifying Data Save Options

To specify data save options, do the following:

1. Choose *AMS – Detailed Setup – Save Options*.

   The Save Options form appears.



Save Options

Filter signals to save by domain	analog only / digital only / [✓ none]
CONNECT MODULES	
Save nets	all / ports / [✓ none]
Type of ports	all / [✓ input] / output
Levels of hierarchy to save	all / selected / [✓ none]
Levels	
CURRENTS	
Save terminal currents	all / [✓ selected] / none
Levels of hierarchy to save	[✓ all] / selected / none
Levels	
Use probes	yes / no
AHDL	
Save AHDL variables	selected / all
INFO	
Save model parameters info	
Save elements info	
Save output parameters info	
Save primitives parameters info	
Save subckt parameters info	
Save design parameters value info	✓
Save asserts info	
Save extreme info	
DATABASE TCL	
Size limit for database file	

For information about the options you can set using this form, see the “Control Statements” chapter of the *Virtuoso Spectre Circuit Simulator User Guide*.

You can specify data you want to save for nets, currents, and other design information (such as model parameter data, element data, and output parameter data).

2. Fill out this form and click *OK*.

See also the following topics for information:

- Saving Data for Nets and Ports on page 120
- Saving Current Data on page 121
- Saving Other Design Information on page 121

Saving Data for Nets and Ports

To save data for nets and ports in your design, do the following in the *NETS* section on the Save Options form:

1. For *Save nets*, select the net data you want to save:
   - *selected* saves data for selected nets only
   - *ports* saves data for all ports

     You can use the *Type of ports* drop-down combo box to specify the port data you want to save:
     - *Input* saves data for all input ports
     - *Output* saves data for all output ports
     - *All* saves data for all ports
   - *all* saves data for all nets, ports, and internal signals
2. (Optional) For *Levels of hierarchy to save*, select one of the following:
   - *all* saves data for all levels of the design hierarchy, from the current level all the way down
   - *selected* saves data for the levels you specify in the *Levels* field
3. (Optional) To save analog signals only, turn on *save analog only*.
4. (Optional) To save digital signals only, turn on save *digital only*.

5. Click *Apply*.

For more information about the options you can set using this form, see the “Control Statements” chapter of the *Virtuoso Spectre Circuit Simulator User Guide*.

Saving Current Data

To save data for currents in your design, do the following in the *CURRENTS* section on the Save Options form:

1. For *Save all terminal currents*, select what you want to save:
   - ❑ *selected* saves data for selected terminal currents only
   - ❑ *all* saves data for all terminal currents
2. (Optional) For *Levels of hierarchy to save*, select one of the following:
   - ❑ *all* saves data for all levels of the design hierarchy, from the current level all the way down
   - ❑ *selected* saves data for the levels you specify in the *Levels* field
3. Click *Apply*.

For more information about the options you can set using this form, see the “Control Statements” chapter of the *Virtuoso Spectre Circuit Simulator User Guide*.

Saving Other Design Information

To save other design data—such as model parameter data, element data, and output parameter data—do the following in the *INFO* section on the Save Options form:

1. To save model parameter information, turn on *Save model parameters info*.
2. To save information about design elements, turn on *Save elements info*.
3. To save output parameter information, turn on *Save output parameters info*.
4. To save information about primitive parameters, turn on *Save primitives parameters info*.
5. To save information about subcircuit parameters, turn on *Save subckt parameters info*.
6. To save information about values of the design parameters, turn on *Save design parameters value info*.

7. To save all the information about the design, turn on *Save extreme info*.
8. Click *Apply*.

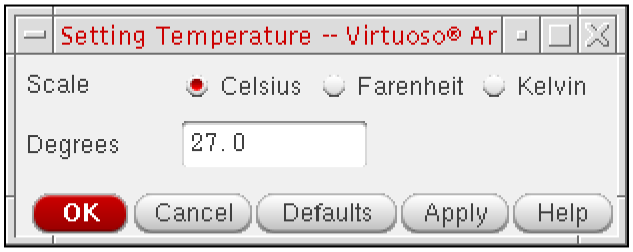
For more information about the options you can set using this form, see the "Control Statements" chapter of the *Virtuoso Spectre Circuit Simulator User Guide*.

Specifying Simulation Temperature

To specify the simulation temperature, do the following:

1. Choose *AMS – Detailed Setup – Temperature*.

   The Setting Temperature form appears.



Scale [✓ Celsius] Farenheit Kelvin
Degrees 27.0

2. For *Scale*, select the units you want to use for temperature.
3. In the *Degrees* field, type a value for temperature.
4. Click *OK*.

Specifying Simulation Files

To specify simulation files such as include paths, definition and vector files, do the following:

1. Choose *AMS – Detailed Setup – Simulation Files*.

   The Simulation Files Setup form appears.
2. Fill out this form and click *OK*.

Specifying Global Signals

To specify global signals, do the following:

1. Choose *AMS – Detailed Setup – Global Signals*.

   The Global Signals form appears.

2. Fill out this form and click *OK*.

Specifying Nodesets

To set node values for simulation, do the following:

1. Choose *AMS – Detailed Setup – Node Set*.

   The Select Node Set form appears.

2. For each node value you want to set, do the following:

   a. On the schematic, select the node.

   b. In the *Node Voltage* field on the Select Node Set form, type a value.

3. Click *OK*.

See also “Selecting Nodes and Setting Their Values” in the *Virtuoso Analog Design Environment L User Guide*.

Specifying Initial Conditions

To specify initial conditions, do the following:

1. Choose *AMS – Detailed Setup – Initial Condition*.

   The Select Initial Condition Set form appears.

2. For each initial condition you want to set, do the following:

   a. On the schematic, select the node.

   b. In the *Node Voltage* field on the Select Initial Condition Set form, type the initial condition.

3. Click *OK*.

See also “Selecting Nodes and Setting Their Values” in the *Virtuoso Analog Design Environment L User Guide*.

5

Using the AMS Options Form

You can use the AMS Options form to specify various settings for the Virtuoso® AMS Designer environment. See the following topics for details:

- Specifying Verilog Timing Options on page 157
- Specifying VHDL Timing Options on page 162
- Specifying Access Options on page 164
- Specifying Profiler Options on page 166
- Specifying Linter Checking Options on page 167
- Specifying Other Options on page 167

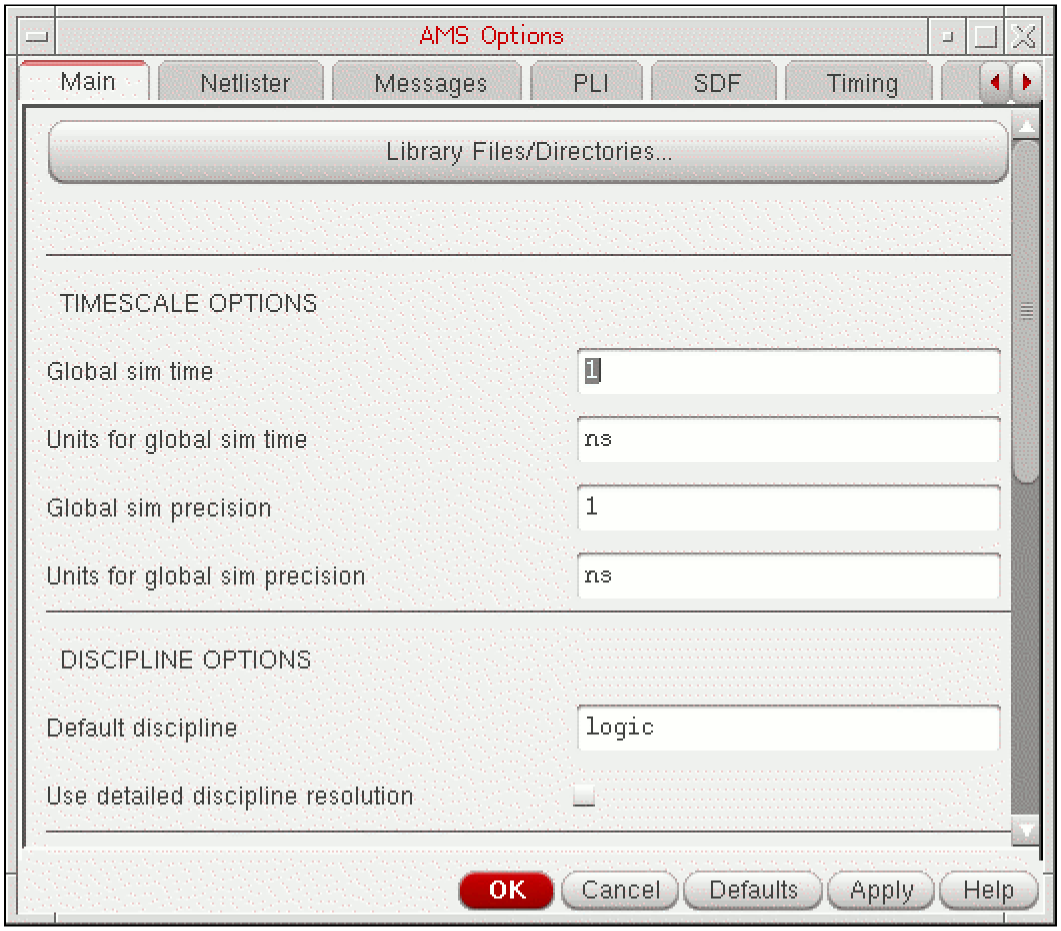
For more information about:	See:
Verilog compiler (`ncvlog`)	"Compiling Verilog Source Files with ncvlog" in *Cadence NC-Verilog Simulator Help*
VHDL compiler (`ncvhdl`)	"Compiling VHDL Source Files with ncvhdl" in *Cadence NC-VHDL Simulator Help*
Elaborator (`ncelab`)	"Elaborating the Design with ncelab" in *Cadence NC-Verilog Simulator Help*
Simulator (`ncsim`)	"Simulating Your Design with ncsim" in *Cadence NC-Verilog Simulator Help*
`ncverilog` simulation	"Running NC-Verilog with the ncverilog command" in *Cadence NC-Verilog Simulator Help*
`irun` simulation	*irun User Guide*

Opening the AMS Options Form

To open the AMS Options form, do the following:

➤ Choose *AMS – Detailed Setup – AMS Options*.

The AMS Options form appears.



TIMESCALE OPTIONS

Global sim time	1
Units for global sim time	ns
Global sim precision	1
Units for global sim precision	ns

DISCIPLINE OPTIONS

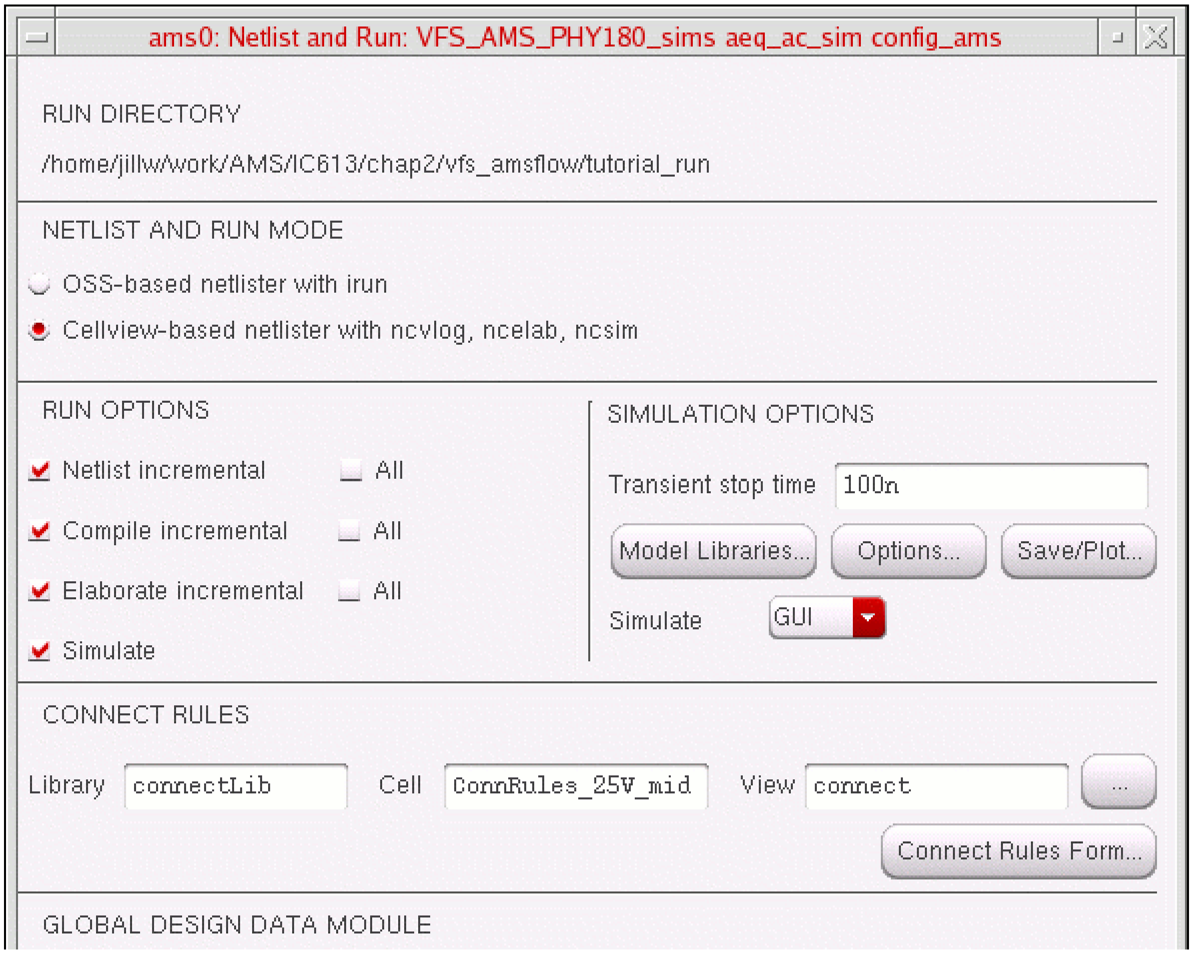
Default discipline	logic
Use detailed discipline resolution	☐

Note: You might need to resize the form window or use the arrows at the top right of the form to see all the tabs.

You can also open the AMS Options form from the Netlist and Run form:

1. Choose *AMS – Netlist and Run* .

   The Netlist and Run form appears.



ams0: Netlist and Run: VFS_AMS_PHY180_sims aeq_ac_sim config_ams

RUN DIRECTORY

/home/jillw/work/AMS/IC613/chap2/vfs_amsflow/tutorial_run

NETLIST AND RUN MODE

OSS-based netlister with irun
[✓ Cellview-based netlister with ncvlog, ncelab, ncsim]

RUN OPTIONS

✓ Netlist incremental ☐ All
✓ Compile incremental ☐ All
✓ Elaborate incremental ☐ All
✓ Simulate

SIMULATION OPTIONS

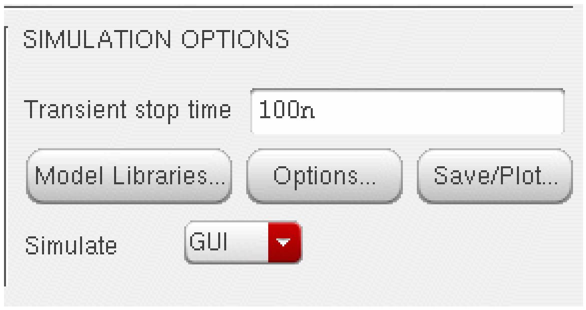
Transient stop time 100n
Simulate GUI

CONNECT RULES

Library connectLib Cell ConnRules_25V_mid View connect

GLOBAL DESIGN DATA MODULE

2. In the *SIMULATION OPTIONS* group box, click *Options*.



SIMULATION OPTIONS

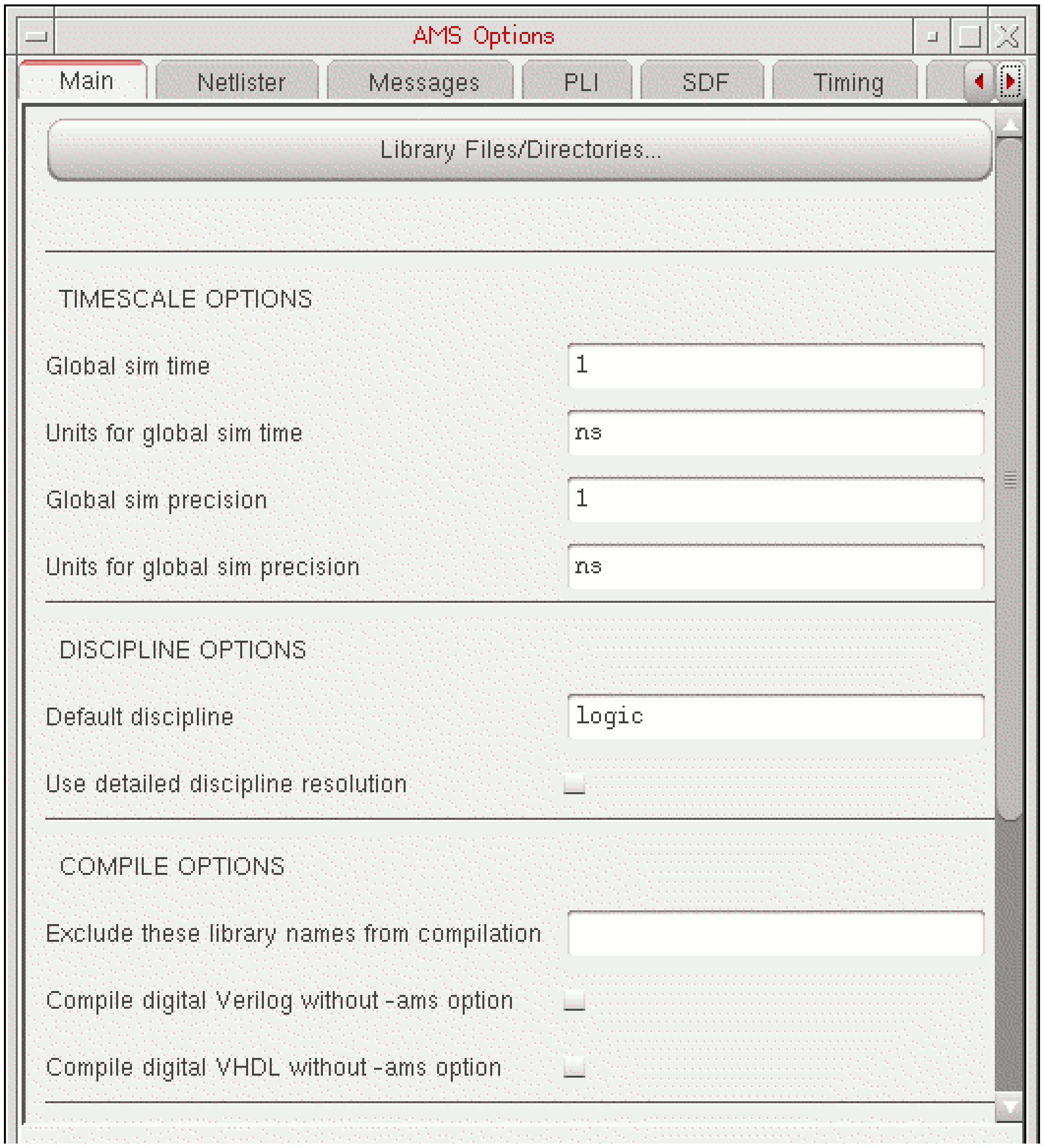
Transient stop time 100n
Simulate GUI

The AMS Options form appears.

Specifying a Tcl Input Script

To specify a Tcl input script to use, do the following:

1. In the AMS Options form, select the *Main* tab.



TIMESCALE OPTIONS

Global sim time	1
Units for global sim time	ns
Global sim precision	1
Units for global sim precision	ns

DISCIPLINE OPTIONS

Default discipline	logic
Use detailed discipline resolution	☐

COMPILE OPTIONS

Exclude these library names from compilation	
Compile digital Verilog without -ams option	☐
Compile digital VHDL without -ams option	☐

2. In the *INCLUDE OPTIONS* group box, click the browse button to the right of the *Tcl input script* field.
3. On the Choose form that appears, navigate to and select a Tcl input script.
4. On the Choose form, click *Open*.

The Tcl input script file name appears in the *Tcl input script* field.

5. Click *OK*.

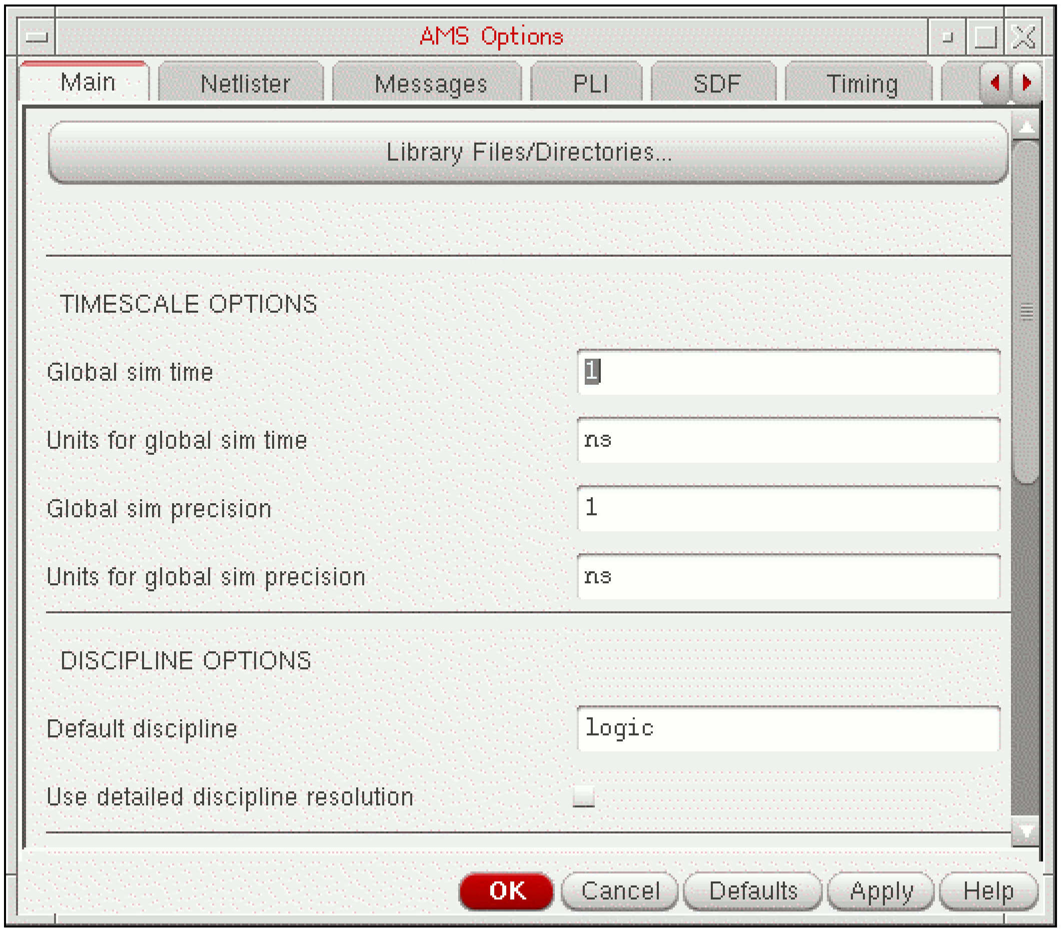
Tip

You can also specify a Tcl input script on the Quick Setup form.

Specifying Library Files and Directories for the Compiler

To specify library files (`-v` *`fileName`*) and library directories (`-y` *`libraryDirectory`*) for the compiler, do the following:

1. In the AMS Options form, select the *Main* tab.



TIMESCALE OPTIONS

Global sim time: 1

Units for global sim time: ns

Global sim precision: 1

Units for global sim precision: ns

DISCIPLINE OPTIONS

Default discipline: logic

Use detailed discipline resolution: ☐

2. Click *Library Files/Directories*.

   The Select Library Files/Directories form appears.

3. Fill out this form and click *OK*.

For instructions, see the “Running a Simulation” chapter in the *Virtuoso Analog Design Environment L User Guide*.

4. In the AMS Options form, click *OK*.

The compiler uses command options according to what you specify on the Select Library Files/Directories form.

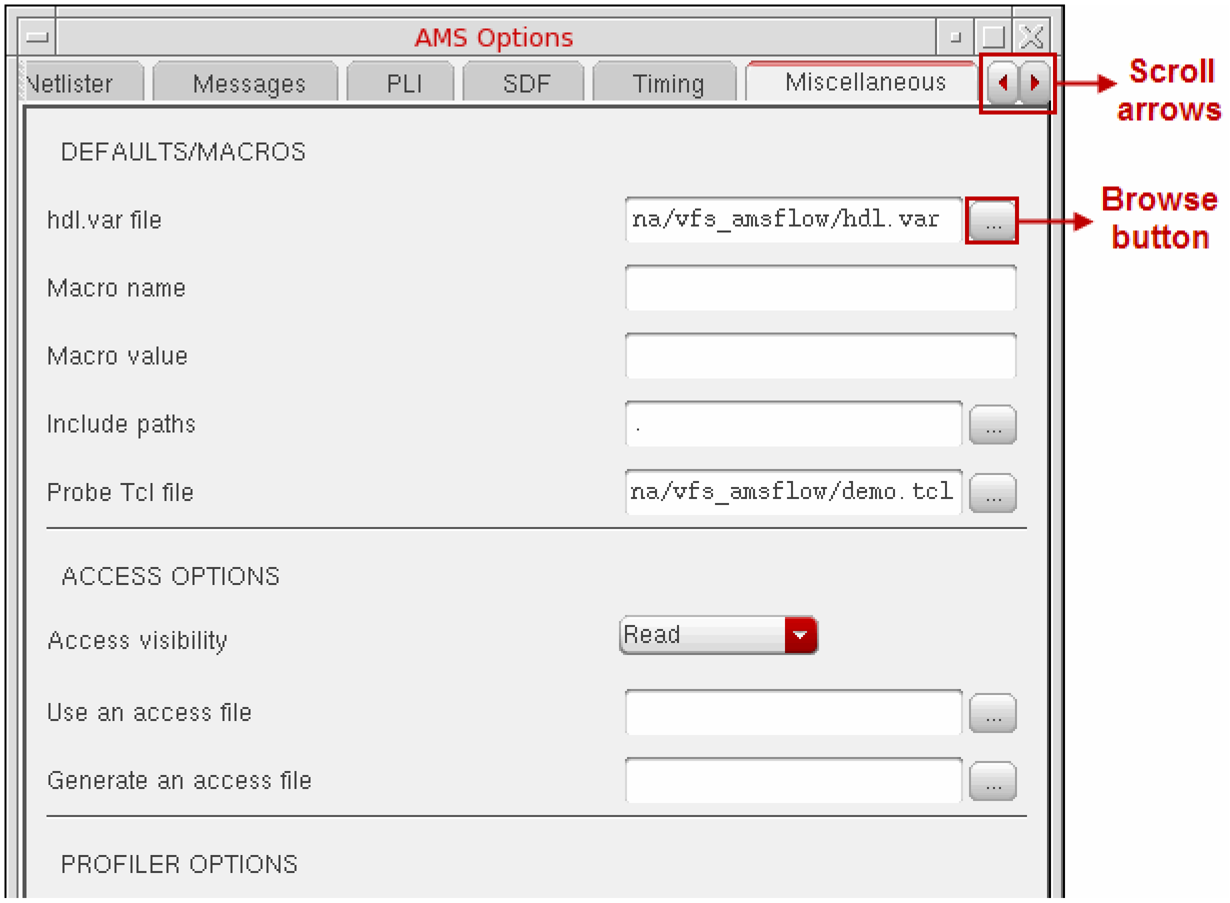
Field	**Command Option**
Library files	`-v` *`fileName`*
Library directories	`-y` *`libraryDirectory`*
Valid extensions for dirs	`+libext+`*`fileExtension`*
View to compile into	`-view` *`viewName`*
Library to compile into	`-specificunit` *`library.cell`*

For more information about these options, see the “Compiling Verilog Source Files with ncvlog” chapter of *NC-Verilog Simulator Help*.

Specifying an hdl.var File

To specify an `hdl.var` file to use, do the following:

1. In the AMS Options form, select the *Misc* tab.



DEFAULTS/MACROS

hdl.var file	na/vfs_amsflow/hdl.var
Macro name	
Macro value	
Include paths	.
Probe Tcl file	na/vfs_amsflow/demo.tcl

ACCESS OPTIONS

Access visibility	Read
Use an access file	
Generate an access file	

PROFILER OPTIONS

2. In the *DEFAULTS/MACROS* group box, click the browse button to the right of the *hdl.var file* field.
3. On the Choose form that appears, navigate to and select an `hdl.var` file.
4. On the Choose form, click *Open*.

   The `hdl.var` file name, including path, appears in the *hdl.var file* field.
5. Click *OK*.

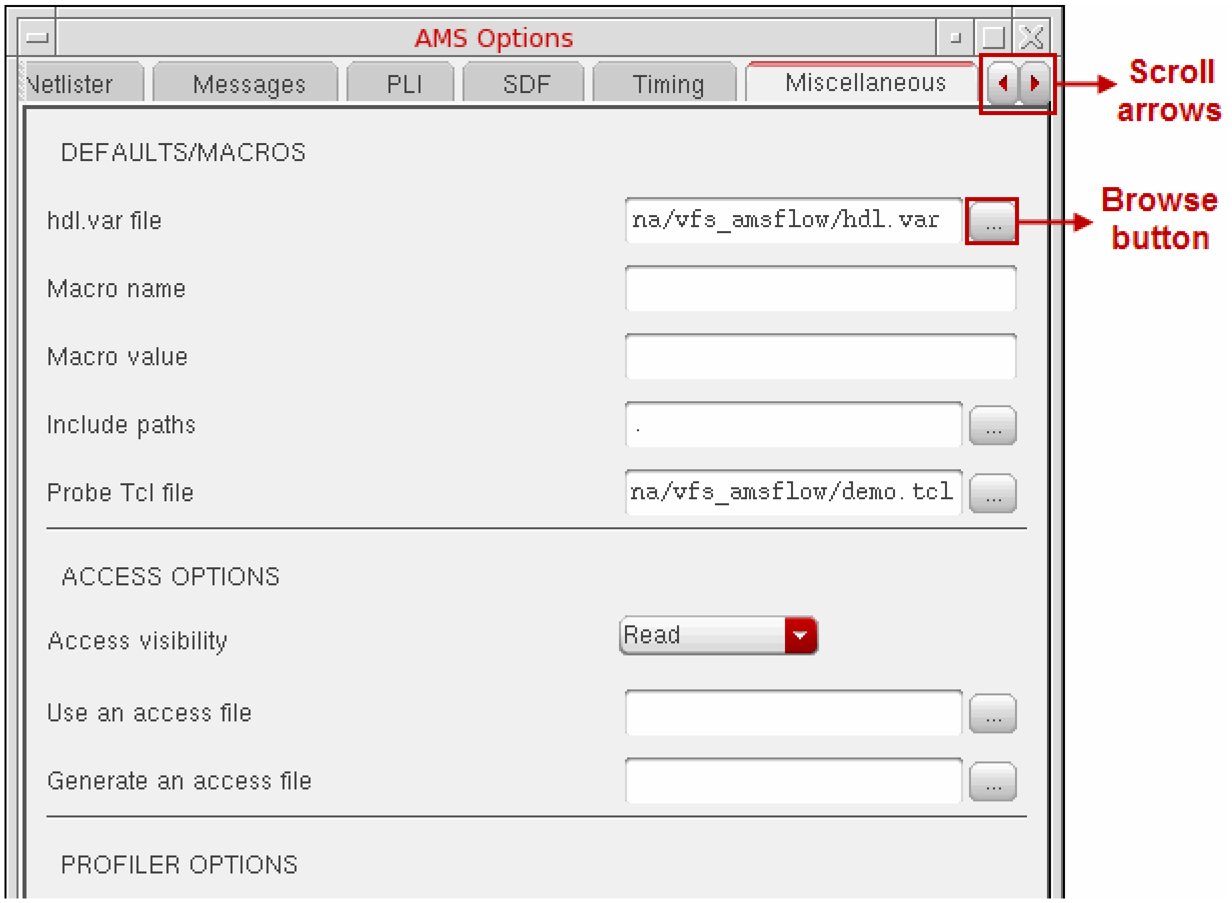
Tip

You can also specify an `hdl.var` file on the Quick Setup form.

Specifying a Verilog-AMS Macro to Use during Compilation

To specify a Verilog-AMS macro to use during compilation, do the following:

1. In the AMS Options form, select the *Misc* tab.



hdl.var file	na/vfs_amsflow/hdl.var
Macro name	
Macro value	
Include paths	.
Probe Tcl file	na/vfs_amsflow/demo.tcl
Access visibility	Read
Use an access file	
Generate an access file	

2. In the *DEFAULTS/MACROS* group box, click in the *Macro name* field and type a space-separated list of one or more macro names.

   This option applies the `-define` option to the `ncvlog` command for each macro name you specify. See “ncvlog Command Options” in “Compiling Verilog Source Files with ncvlog” in *Cadence NC-Verilog Simulator Help* for more information.

3. (Optional) In the *Macro value* field, type a value.

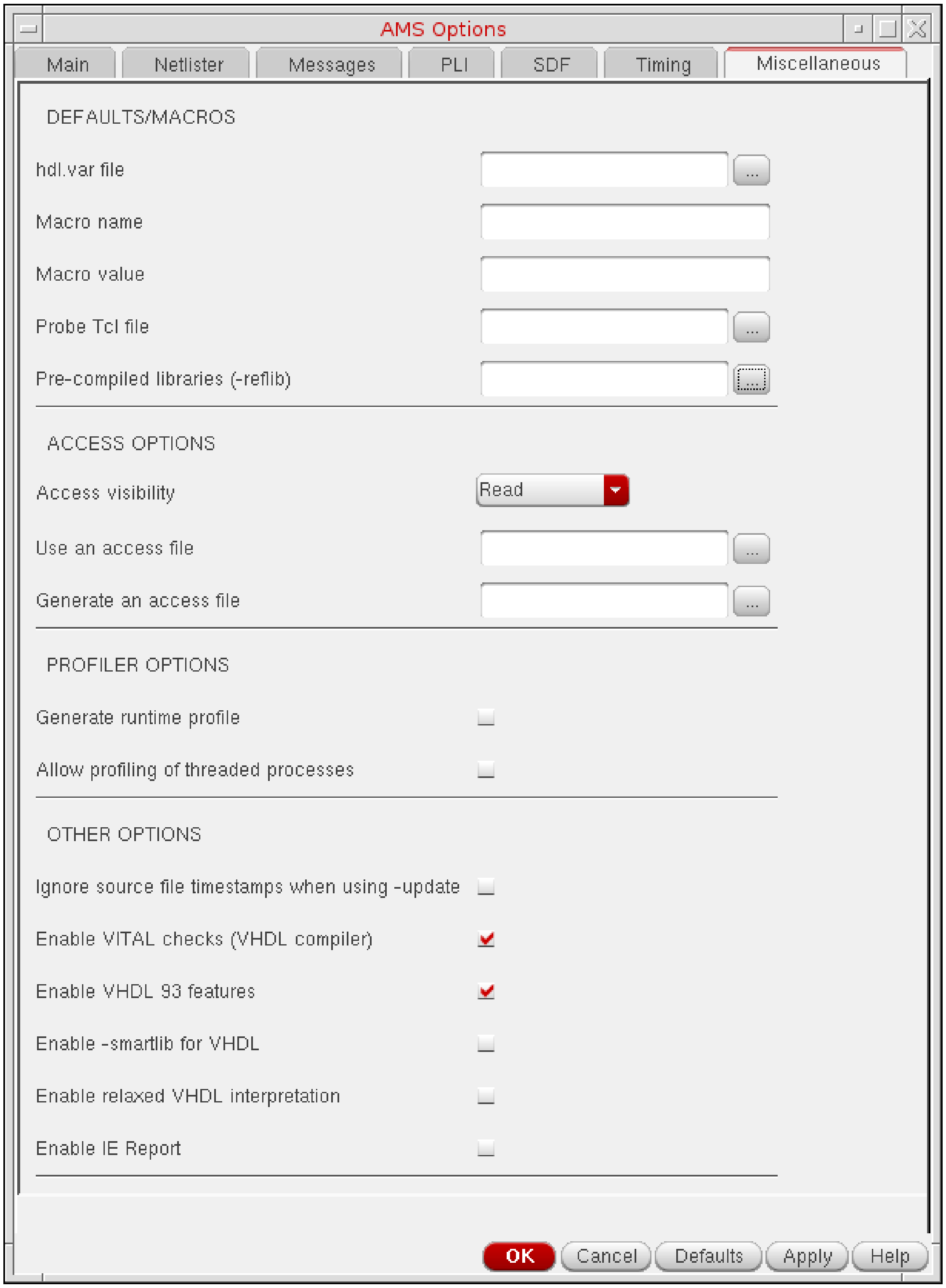
4. Click *OK*.

   The Verilog-AMS compiler uses the specified macro during compilation.

Specifying an Include Path

To specify an include path, do the following:

1. In the AMS Options form, select the *Misc* tab.



DEFAULTS/MACROS

hdl.var file	
Macro name	
Macro value	
Probe Tcl file	
Pre-compiled libraries (-reflib)	

ACCESS OPTIONS

Access visibility	Read
Use an access file	
Generate an access file	

PROFILER OPTIONS

Generate runtime profile	☐
Allow profiling of threaded processes	☐

OTHER OPTIONS

Ignore source file timestamps when using -update	☐
Enable VITAL checks (VHDL compiler)	✓
Enable VHDL 93 features	✓
Enable -smartlib for VHDL	☐
Enable relaxed VHDL interpretation	☐
Enable IE Report	☐

2. In the *DEFAULTS/MACROS* group box, click the browse button to the right of the *Include path* field.
3. On the Choose form that appears, navigate to and select an include files directory.
4. On the Choose form, click *Open*.

   The full path to the include directory appears in the *Include path* field.
5. Click *OK*.

   The program uses the path you specify to search for include files it encounters in any `` `include `` statements in your Verilog-AMS files. For example,

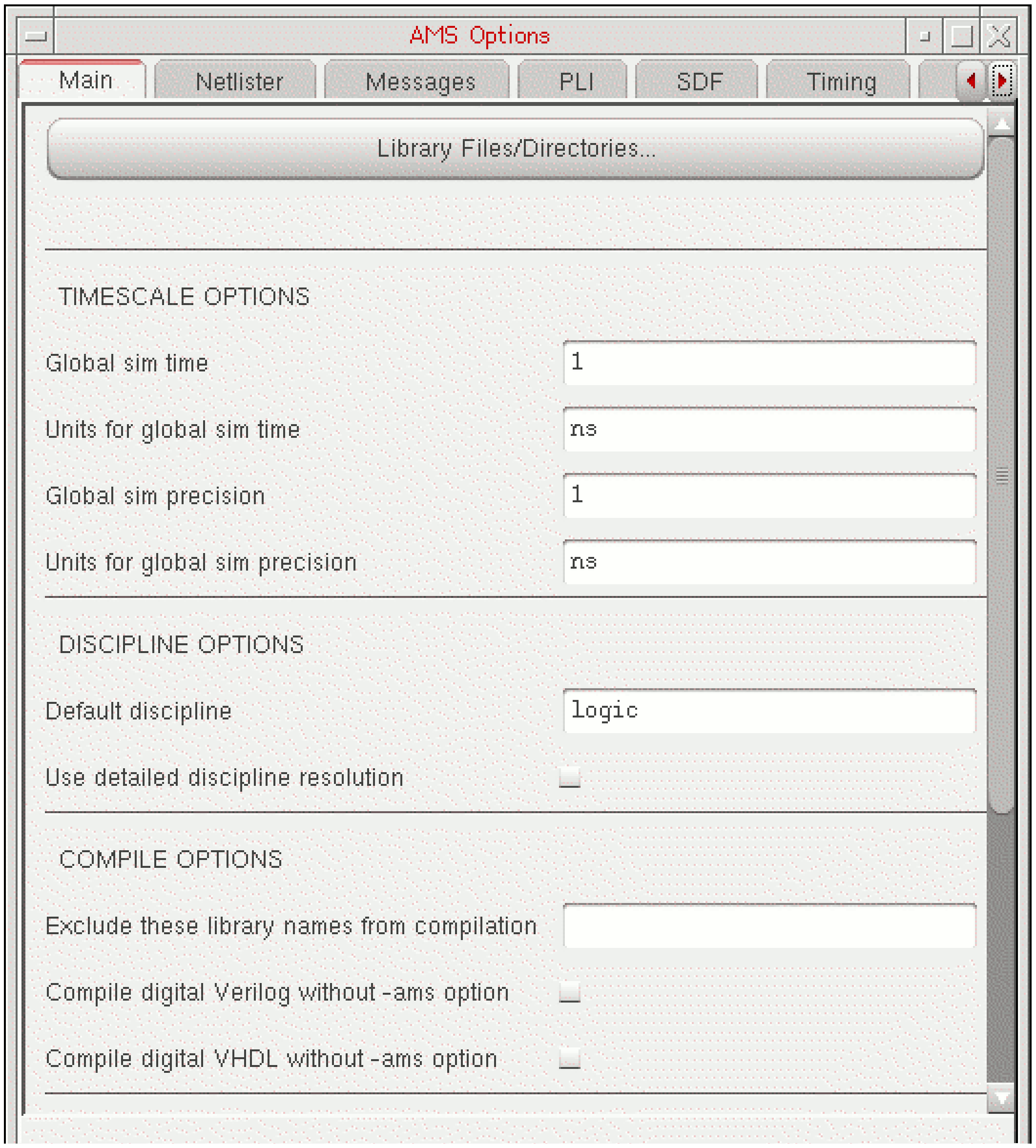
   ```
   `include "myfunctions.vams"
   ```

Specifying Default Timescale Options

Note: Timescale settings have no effect on VHDL code.

To specify a default timescale for the Verilog elaborator (`ncelab`), do the following:

1. In the AMS Options form, select the *Main* tab.



TIMESCALE OPTIONS

Global sim time	1
Units for global sim time	ns
Global sim precision	1
Units for global sim precision	ns

DISCIPLINE OPTIONS

Default discipline	logic
Use detailed discipline resolution	☐

COMPILE OPTIONS

Exclude these library names from compilation	
Compile digital Verilog without -ams option	☐
Compile digital VHDL without -ams option	☐

2. In the *TIMESCALE OPTIONS* group box, fill in the following fields with default timescale details:

Global sim time	Integer value for default `time_unit`
Units for global sim time	Engineering notation for unit of time, such as `ns` for nanoseconds
Global sim precision	Integer value for default `time_precision`
Units for global sim precision	Engineering notation for unit of precision, such as `ns` for nanoseconds

Specifying these options applies the `-timescale` option to the `ncelab` command like this:

`ncelab -timescale '`*<...time><Units for ... time>*`/`*<... precision><Units for ...precision>*`'`

For example, if you fill in these values:

Global sim time	`1`
Units for global sim time	`ns`
Global sim precision	`100`
Units for global sim precision	`ps`

the resulting `ncelab -timescale` specification is:

```
ncelab -timescale '1ns/100ps'
```

3. Click *OK*.

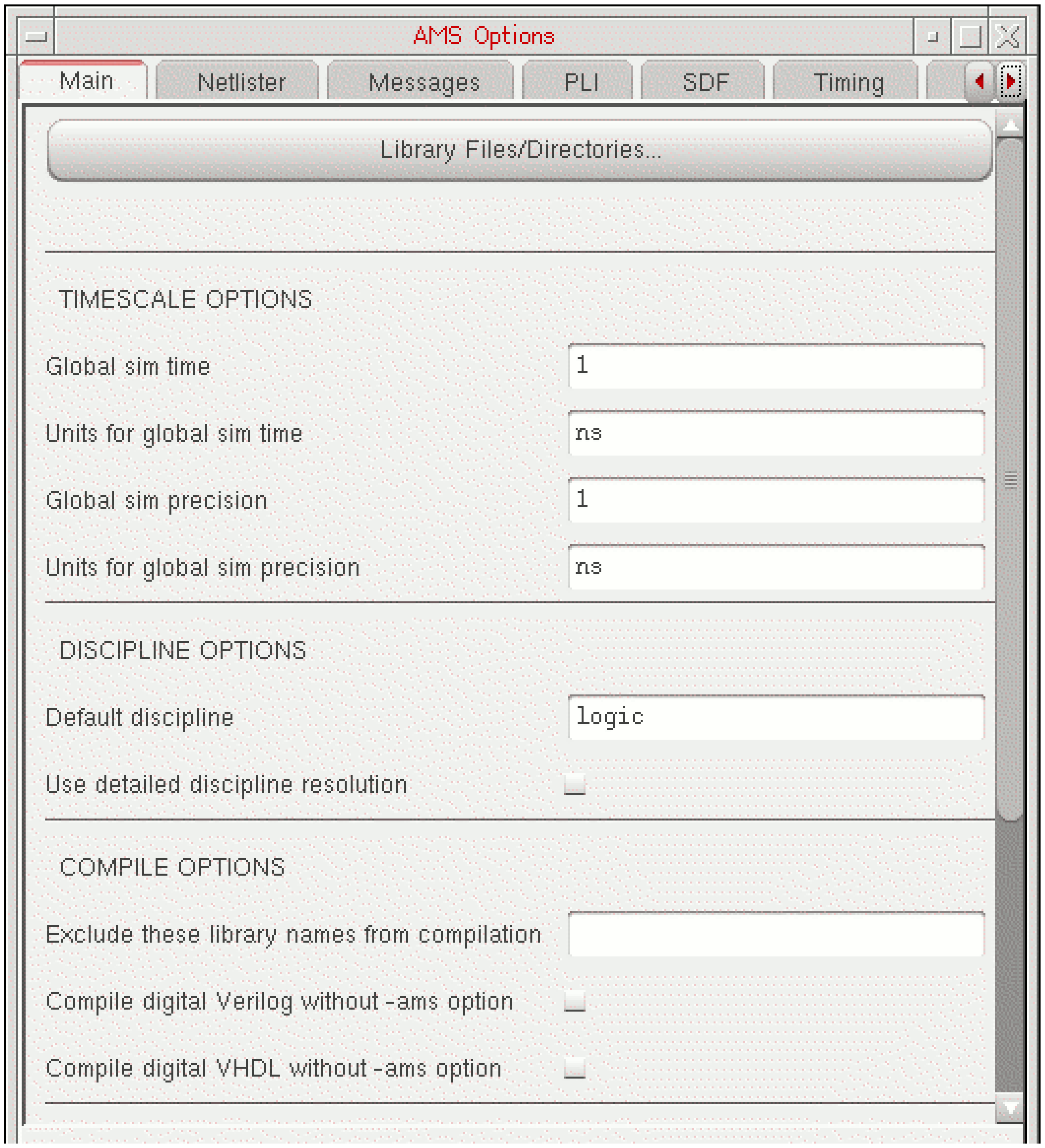
You can read more about using `-timescale` with `ncelab` in the “Elaborating the Design with ncelab” chapter of Cadence’s *NC-Verilog Simulator Help*.

Specifying Discipline Options

You can specify a discipline for the elaborator to use for discrete nets that do not have a discipline specified or when the elaborator cannot determine one through discipline resolution. You can also specify whether you want the elaborator to use detailed discipline resolution.

To specify the default discipline and discipline resolution options, do the following:

1. In the AMS Options form, select the *Main* tab.



TIMESCALE OPTIONS

Global sim time	1
Units for global sim time	ns
Global sim precision	1
Units for global sim precision	ns

DISCIPLINE OPTIONS

Default discipline	logic
Use detailed discipline resolution	☐

COMPILE OPTIONS

Exclude these library names from compilation	
Compile digital Verilog without -ams option	☐
Compile digital VHDL without -ams option	☐

2. In the *DISCIPLINE OPTIONS* group box, type the default discipline for discrete nets in the *Default discipline* field.

   The content of this field corresponds to the value of the `ncelab -discipline` option. For more information, see "-discipline Option" in the "Elaborating" chapter of the *Virtuoso AMS Designer Simulator User Guide*.

3. (Optional) To specify detailed discipline resolution, place a mark in the *Use detailed discipline resolution* check box.

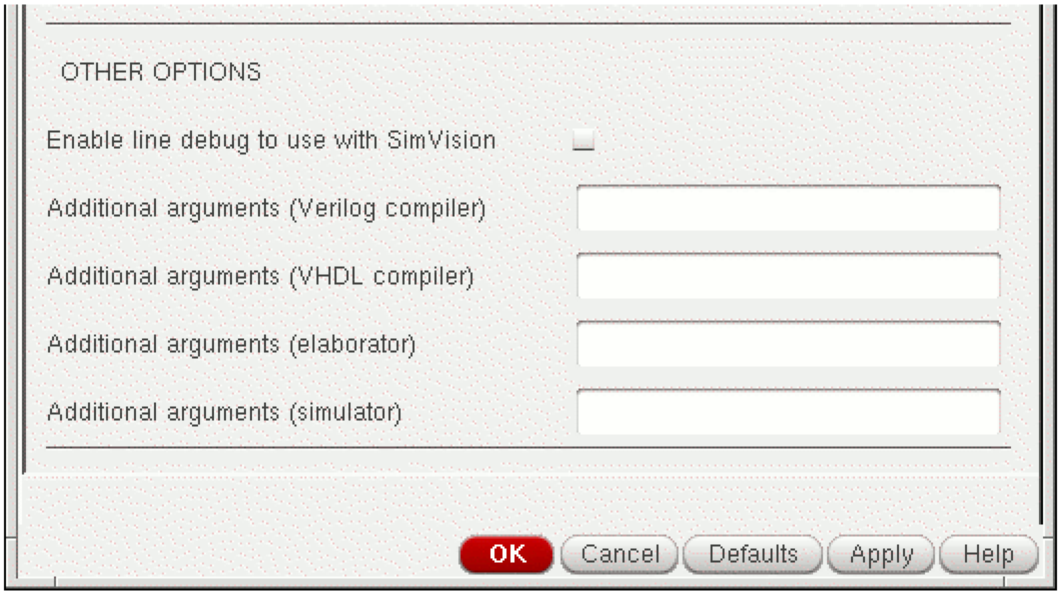
   Marking this check box applies the `-dresolution` option on the `ncelab` command line. For more information, see "-dresolution Option" in the "Elaborating" chapter of the *Virtuoso AMS Designer Simulator User Guide*.

4. Click *OK*.

Specifying Additional Arguments for the Elaborator

To specify additional arguments for the elaborator (`ncelab`), do the following:

1. In the AMS Options form, select the *Main* tab.
2. Scroll down to the *OTHER OPTIONS* group box.



OTHER OPTIONS

Enable line debug to use with SimVision	☐
Additional arguments (Verilog compiler)	
Additional arguments (VHDL compiler)	
Additional arguments (elaborator)	
Additional arguments (simulator)	

3. In the *Additional arguments (elaborator)* field, type any additional arguments you want the elaborator to use.

   For information about elaborator options, see “Elaborating the Design with ncelab” in *Cadence NC-Verilog Simulator Help*.

Important

You must not specify a `-log` argument because the elaborator automatically writes the default log file, `ncelab.log`, to the run directory (unless you select *No log file*).

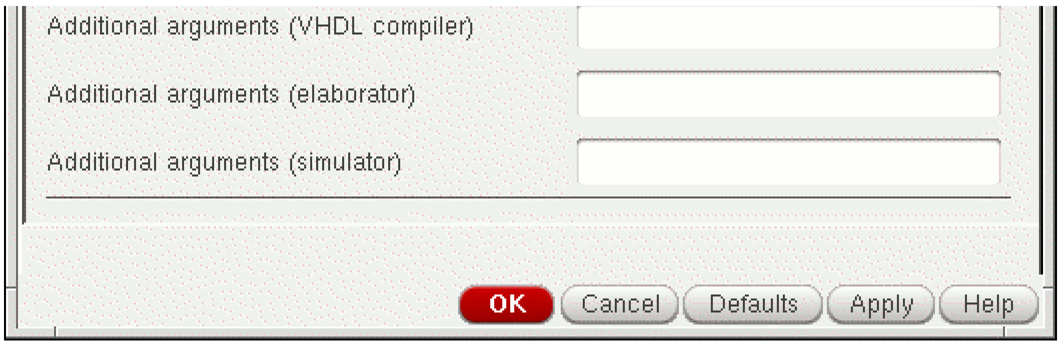
4. Click *OK*.

   The elaborator uses the options you specified.

Specifying Additional Arguments for the Simulator

To specify additional arguments for the simulator (`ncsim`), do the following:

1. In the AMS Options form, select the *Main* tab.
2. Scroll down to the *OTHER OPTIONS* group box.



Additional arguments (VHDL compiler)	
Additional arguments (elaborator)	
Additional arguments (simulator)	

3. In the *Additional arguments (simulator)* field, type any additional arguments you want the simulator to use.

   For information about simulator options, see “Simulating Your Design with ncsim” in *Cadence NC-Verilog Simulator Help*.

Important

You must not specify a `-log` argument because the simulator automatically writes the default log file, `ncsim.log`, to the run directory (unless you select *No log file*).

4. Click *OK*.

   The simulator uses the options you specified.

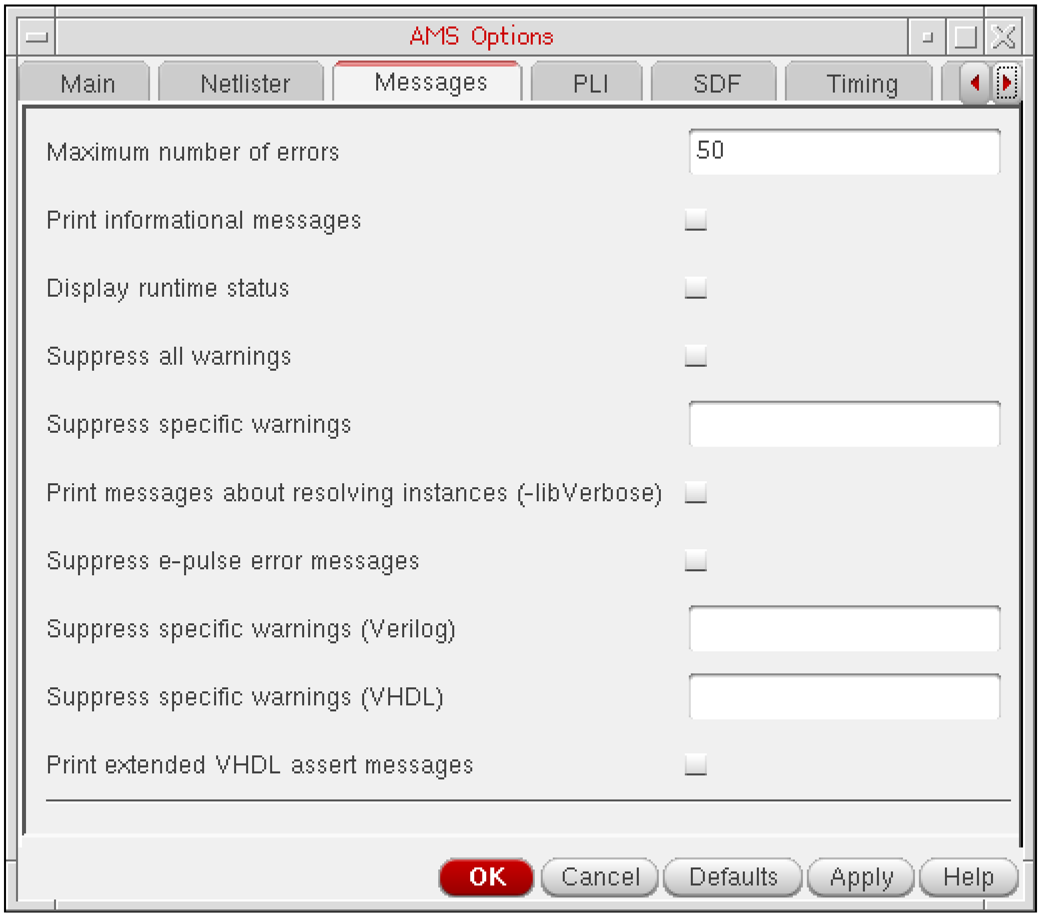
Controlling Messages for the Compiler, Elaborator, and Simulator

On the *Messages* tab in the AMS Options form, you can control message output from the compilers, elaborator, and simulator. Each item you specify on this tab applies a command-line option to a compiler, the elaborator, or the simulator, as indicated. For more information about these options, see the following documents:

For more information about:	See:
Verilog compiler (`ncvlog`)	“Compiling Verilog Source Files with ncvlog” in *Cadence NC-Verilog Simulator Help*
VHDL compiler (`ncvhdl`)	“Compiling VHDL Source Files with ncvlog” in *Cadence NC-VHDL Simulator Help*
Elaborator (`ncelab`)	“Elaborating the Design with ncelab” in *Cadence NC-Verilog Simulator Help*
Simulator (`ncsim`)	“Simulating Your Design with ncsim” in *Cadence NC-Verilog Simulator Help*

To specify options for controlling message output from the compilers, the elaborator, and the simulator, do the following:

1. In the AMS Options form, select the *Messages* tab.



Maximum number of errors	50
Print informational messages	☐
Display runtime status	☐
Suppress all warnings	☐
Suppress specific warnings	
Print messages about resolving instances (-libVerbose)	☐
Suppress e-pulse error messages	☐
Suppress specific warnings (Verilog)	
Suppress specific warnings (VHDL)	
Print extended VHDL assert messages	☐

2. (Optional) In the *Maximum number of errors* field, type the maximum number of errors the compilers, elaborator, and simulator can encounter before they stop processing the design.

   If any of these applications (the compiler, the elaborator, the simulator) encounters more errors than the number you specified in the *Maximum number of errors* field, it stops processing the design.

3. (Optional) Turn on *Print informational messages*.

   The compiler, elaborator, and simulator will print more numerous and more extensive informational messages which can help you if you are trying to debug a problem.

   This option applies the `-messages` option to the `ncvlog`, `ncvhdl`, `ncelab`, and `ncsim` commands.

4. (Optional) Turn on *Display runtime status*.

   The program prints statistics on memory and CPU usage after compilation, elaboration, and simulation.

   This option applies the `-status` option to the `ncvlog`, `ncvhdl`, `ncelab`, and `ncsim` commands.

5. (Optional) Turn on *Suppress all warnings*.

   The program suppresses all warning messages.

   This option applies the `-neverwarn` option to the `ncvlog`, `ncvhdl`, `ncelab`, and `ncsim` commands.

6. (Optional) In the *Suppress specific warnings* field, type a comma- or space-separated list of warning message codes.

   The program suppresses the specified warning messages.

   This option applies the `-nowarn` option to the `ncelab` and `ncsim` commands.

7. (Optional) Turn on *Print messages about resolving instances (Verilog)*.

   The elaborator displays messages about module and Verilog UDP instantiations.

   This option applies the `-libverbose` option to the `ncelab` command.

8. (Optional) Turn on *Suppress e-pulse error messages*.

   The simulator suppresses error messages for pulses smaller than a specified percentage error (0 to 100).

   This option applies the `-epulse_no_msg` option to the `ncsim` command.

9. (Optional) In the *Suppress specific warnings (Verilog)* field, type a comma- or space-separated list of warning message codes.

   The `ncvlog` compiler suppresses the specified warning messages.

   This option applies the `-nowarn` option to the `ncvlog` command.

10. (Optional) In the *Suppress specific warnings (VHDL)* field, type a comma- or space-separated list of warning message codes.

    The compiler (`ncvhdl`) suppresses the specified warning messages.

    This option applies the `-nowarn` option to `ncvhdl` command.

11. (Optional) Turn on *Print extended VHDL assert messages*.

The simulator displays extended VHDL assert messages.

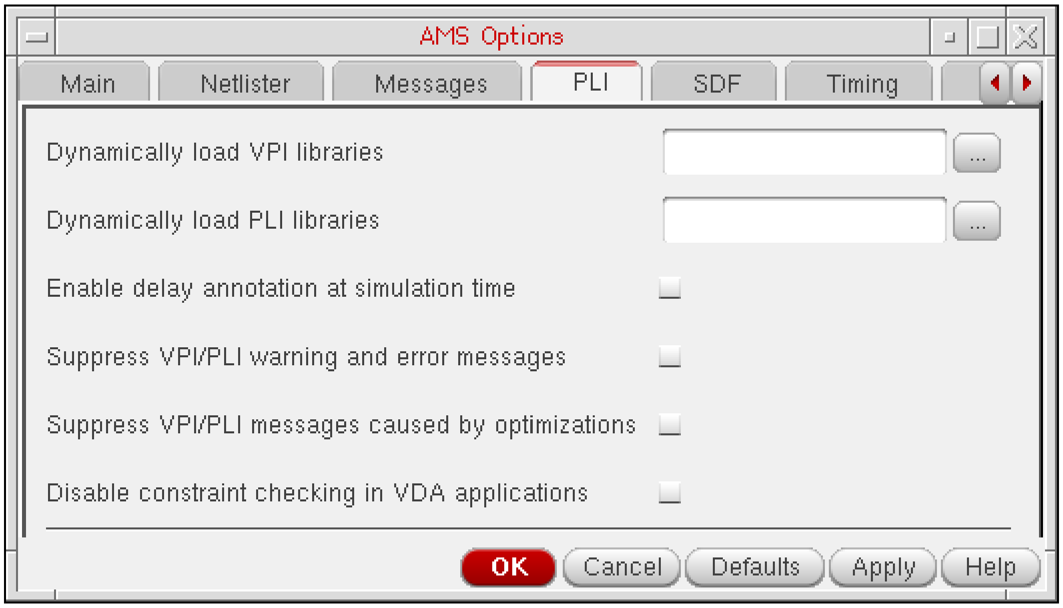
This option applies the `-extassertmsg` option to the `ncsim` command.

12. Click *OK*.

Specifying VPI and PLI Options

You can set options that affect how the elaborator handles VPI[1] and PLI[2] applications (user system tasks and functions) and whether the program issues error or warning messages related to these. These options apply only to elaboration and simulation of Verilog design units.

1. In the AMS Options form, select the *PLI* tab.



Dynamically load VPI libraries	
Dynamically load PLI libraries	
Enable delay annotation at simulation time	☐
Suppress VPI/PLI warning and error messages	☐
Suppress VPI/PLI messages caused by optimizations	☐
Disable constraint checking in VDA applications	☐

2. (Optional) In the *Dynamically load VPI libraries* field, type the name or full path to the dynamic shared library that contains the VPI application you want the elaborator to load.

   This option registers the system tasks and VPI callbacks defined in the application at run time.

   This option applies the `-loadvpi` option to the `ncelab` command.

3. (Optional) Turn on *Dynamically load VPI libraries for AMS/Matlab* to load the dynamic shared library that contains the VPI code for AMS/MATLAB cosimulation.

   This option registers the system tasks and VPI callbacks at run time.

1. Verilog Procedural Interface; see the *VPI User Guide and Reference* for more information
2. Programming Language Interface; see the *PLI 1.0 User Guide and Reference* for more information

This option applies the `-amsmatlab` option to the `irun` command.

4. (Optional) In the *Dynamically load PLI libraries* field, type the name or full path to the dynamic shared library that contains the PLI1.0 application you want the elaborator to load.

   This option registers the system tasks and PLI1.0 callbacks defined in the application at run time.

   This option applies the `-loadpli1` option to the `ncelab` command.

5. (Optional) Turn on *Enable delay annotation at simulation time* to enable the use of VPI/PLI routines that modify delays at simulation time.

   This option applies the `-anno_simtime` option to the `ncelab` command.

6. (Optional) Turn on *Suppress VPI/PLI warning and error messages* to disable the printing of VPI/PLI warning and error messages during elaboration and simulation.

   This option applies the `-plinowarn` option to the `ncelab` and `ncsim` commands.

7. (Optional) Turn on *Suppress VPI/PLI messages caused by optimizations* to print a single warning message the first time (only) the program detects a PLI read, write, or connectivity access violation.

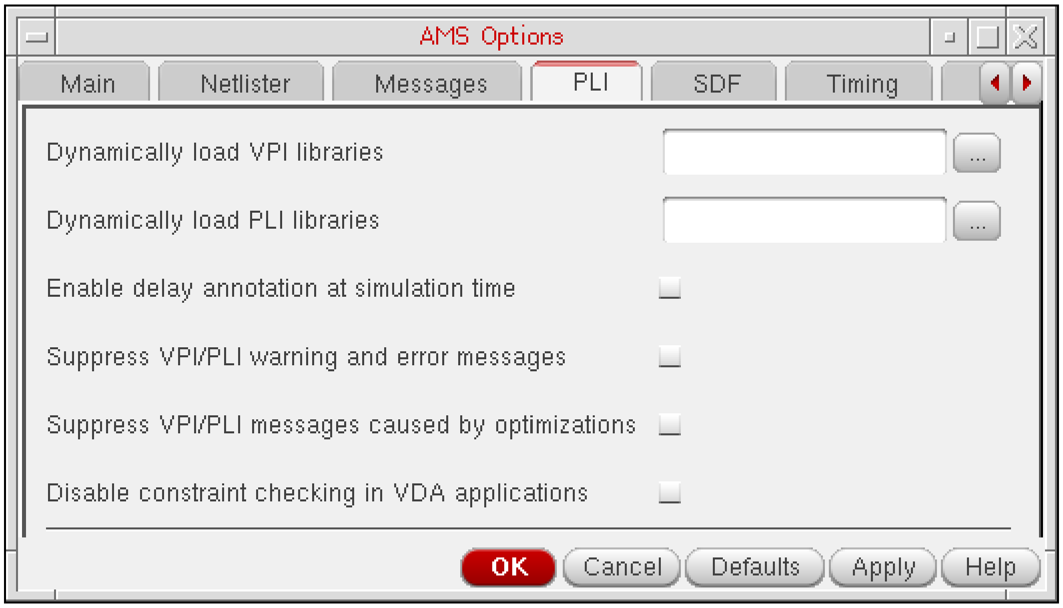
   This option applies the `-plinooptwarn` option to the `ncelab` and `ncsim` commands.

8. Click *OK*.

Disabling Constraint Checking in VHDL Design Access Functions

To disable constraint checking in VHDL Design Access (VDA) functions for increased performance, do the following:

1. In the AMS Options form, select the *PLI* tab.



Dynamically load VPI libraries
Dynamically load PLI libraries
Enable delay annotation at simulation time
Suppress VPI/PLI warning and error messages
Suppress VPI/PLI messages caused by optimizations
Disable constraint checking in VDA applications

2. Turn on *Disable constraint checking in VDA applications*.

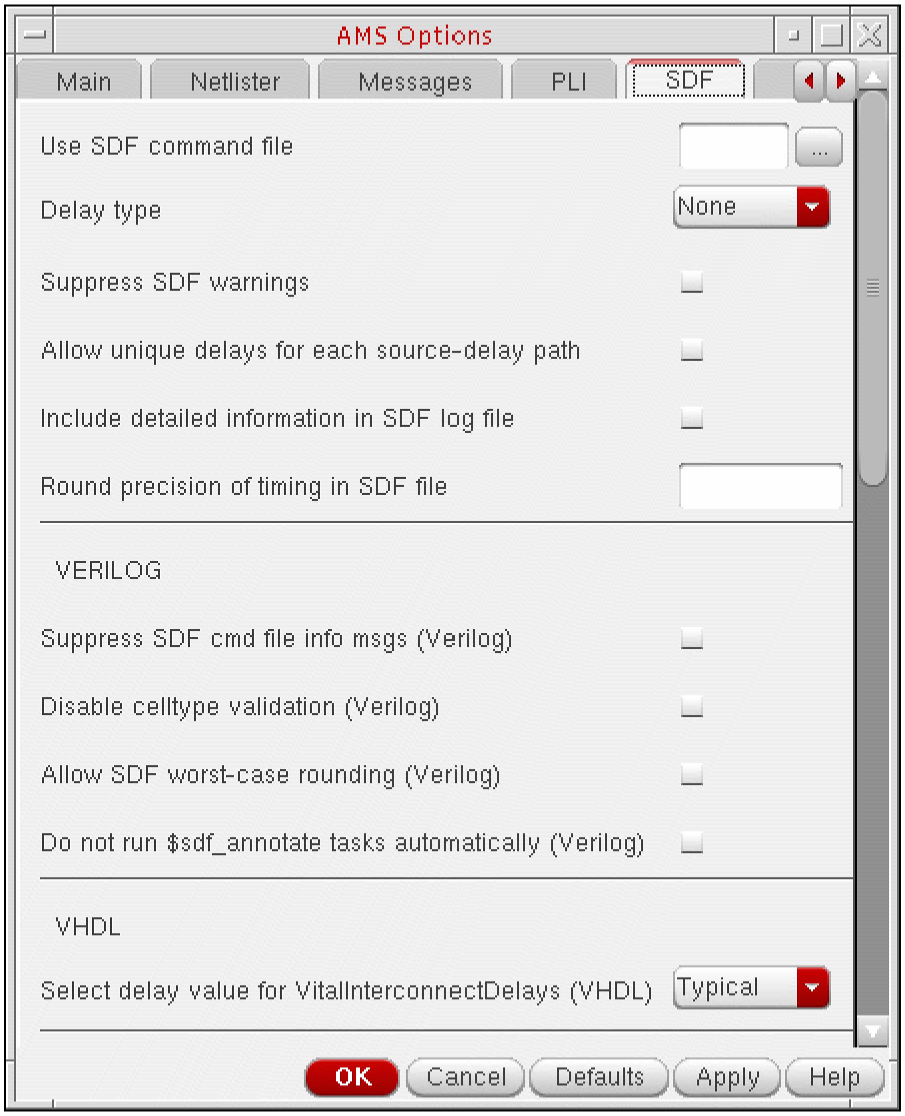
   This option applies the `-nocifcheck` option to the `ncsim` command.

3. Click *OK*.

Specifying SDF Annotation Options for the Elaborator

To specify SDF annotation options for the elaborator, do the following:

1. In the AMS Options form, select the *SDF* tab.



Use SDF command file	
Delay type	None
Suppress SDF warnings	☐
Allow unique delays for each source-delay path	☐
Include detailed information in SDF log file	☐
Round precision of timing in SDF file	
VERILOG	
Suppress SDF cmd file info msgs (Verilog)	☐
Disable celltype validation (Verilog)	☐
Allow SDF worst-case rounding (Verilog)	☐
Do not run $sdf_annotate tasks automatically (Verilog)	☐
VHDL	
Select delay value for VitalInterconnectDelays (VHDL)	Typical

2. (Optional) In the *Use SDF command file* field, type the name of the SDF command file you want the elaborator to use to control SDF annotation.

   Note: Relative paths are relative to the run directory, not to the directory where you started your Cadence software.

   This option applies the `-sdf_cmd_file` option to the `ncelab` command.

3. (Optional) In the *Delay type* drop-down combo box, select one of the following delay types to use from the `min:typ:max` timing triplet in the SDF file while annotating to Verilog or to VITAL:

Minimum	Use the minimum delay value (`min`) This value applies the `-mindelays` option to the `ncelab` command.
Typical	Use the typical delay value (`typ`) This value applies the `-typdelays` option to the `ncelab` command.
Maximum	Use the maximum delay value (`max`) This value applies the `-maxdelays` option to the `ncelab` command.

4. (Optional) Turn on *Suppress SDF warnings* to suppress warning messages from the SDF annotator.

   This option applies the `-sdf_no_warnings` option to the `ncelab` command.

5. (Optional) Turn on *Allow unique delays for each source-delay path* to enable multisource and transport delay behavior with pulse control for interconnect delays.

   This option applies the `-intermod_path` option to the `ncelab` command.

6. (Optional) Turn on *Include detailed information in SDF log file* to include detailed information in the SDF log file.

   This option applies the `-sdf_verbose` option to the `ncelab` command.

7. (Optional) In the *Round precision of timing in SDF file* field, type a precision value (integer `1`, `10`, or `100`, and time unit `fs`, `ps`, `ns`, `us`, or `s`; for example, `1ns`) to which you want the program to round timing values in the compiled SDF file.

   This option applies the `-sdf_precision` option to the `ncelab` command.

8. (Optional; Verilog only) Turn on *Suppress SDF cmd file info msgs (Verilog)* to disable printing of elaborator messages that display information contained in the SDF command file.

   This option applies the `-no_sdfa_header` option to the `ncelab` command.

9. (Optional; Verilog only) Turn on *Disable celltype validation (Verilog)* to disable celltype validation between the SDF annotator and the Verilog description.

   This option applies the `-sdf_nocheck_celltype` option to the `ncelab` command.

10. (Optional; Verilog only) Turn on *Allow SDF worst-case rounding (Verilog)* to truncate the minimum timing value, round the typical timing value, and round up the maximum timing value in the SDF file.

    This option applies the `-sdf_worstcase_rounding` option to the `ncelab` command.

11. (Optional; Verilog only) Turn on *Do not run $sdf_annotate tasks automatically (Verilog)* to disable automatic SDF annotation.

    This option applies the `-noautosdf` option to the `ncelab` command.

12. (Optional; VHDL only) In the *Select delay value for VitalInterconnectDelays (VHDL)* drop-down combo box, select one of the following delay values to use when more than one interconnect specification maps to the same interconnect path delay generic.

Typical	Use the typical delay value This value applies no `-vipd*` option to the `ncelab` command.
Minimum	Use the minimum delay value This value applies the `-vipdmin` option to the `ncelab` command.
Maximum	Use the maximum delay value This value applies the `-vipdmax` option to the `ncelab` command.

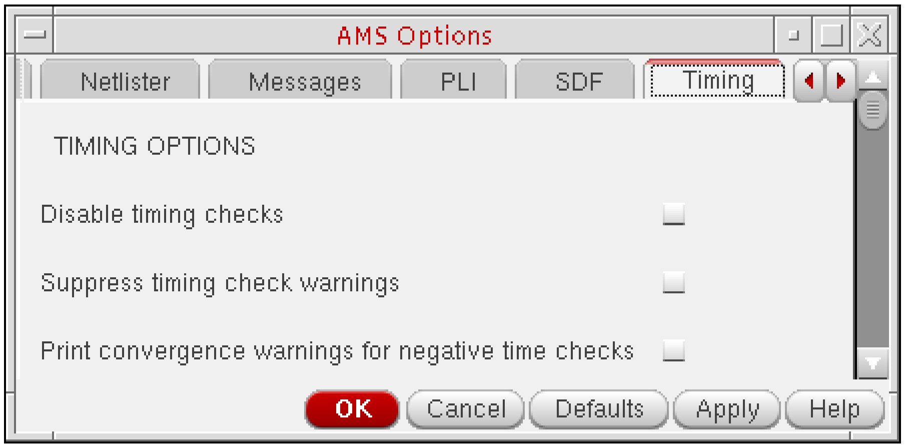
13. Click *OK*.

For more information about these options, see “Elaborating the Design with ncelab” in *Cadence NC-Verilog Simulator Help*.

Specifying Timing Check Options

To specify timing check options that apply during elaboration, do the following:

1. In the AMS Options form, select the *Timing* tab.



TIMING OPTIONS

Disable timing checks	☐
Suppress timing check warnings	☐
Print convergence warnings for negative time checks	☐

2. (Optional) Turn on *Disable timing checks* to suppress the execution of timing checks during elaboration.

   This option applies the `-notimingchecks` option to the `ncelab` command.

3. (Optional) Turn on *Suppress timing check warnings* to suppress the display of timing check warning messages during elaboration.

   This option applies the `-no_tchk_msg` option to the `ncelab` command.

4. (Optional) Turn on *Print convergence warnings for negative time checks* to print convergence warnings for negative timing checks for both Verilog and VITAL if the program cannot calculate delays given the current limit values during elaboration.

   This option applies the `-ntc_warn` option to the `ncelab` command.

5. Click *OK*.

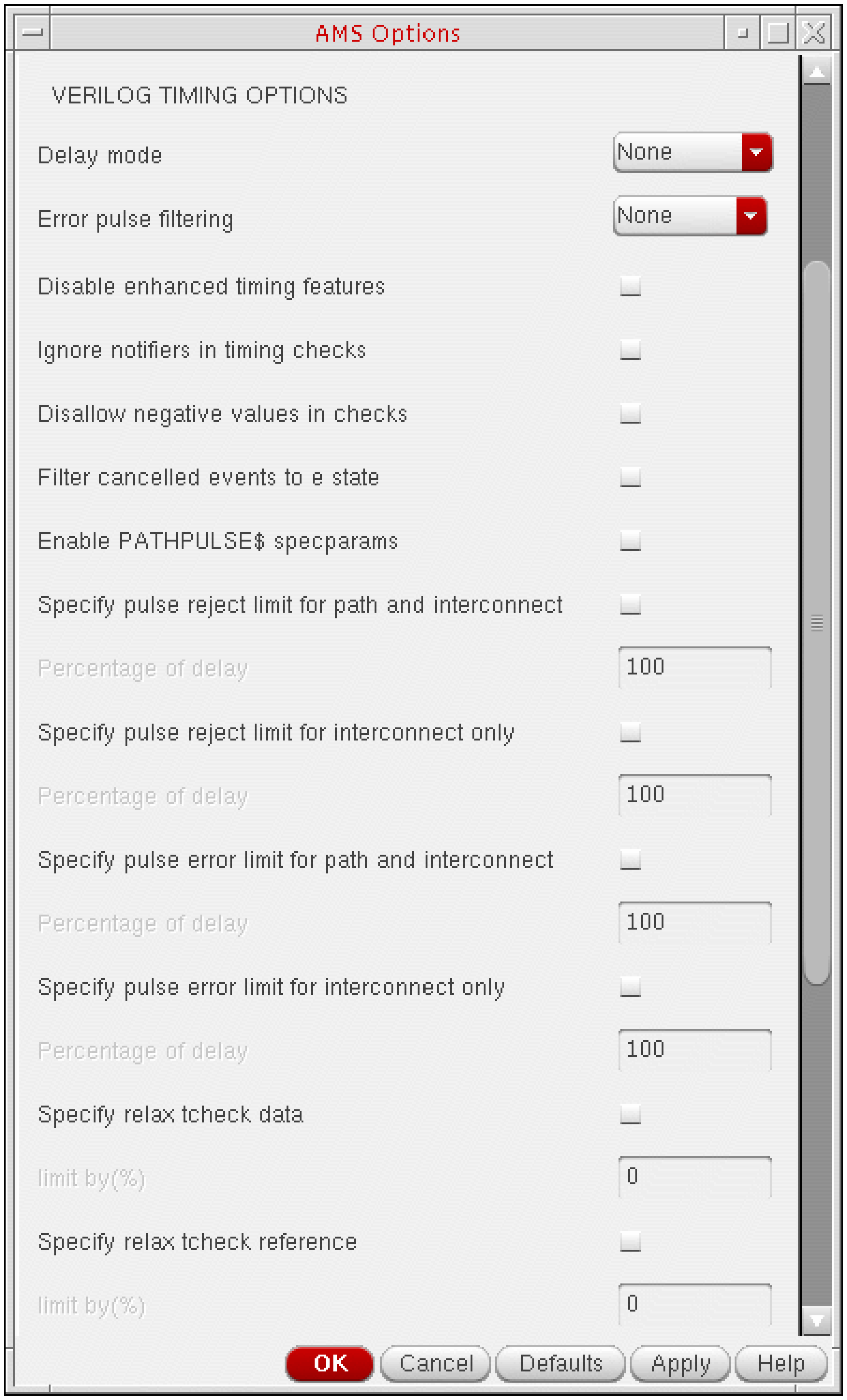
For more information about these options, see “Elaborating the Design with ncelab” in *Cadence NC-Verilog Simulator Help*.

Specifying Verilog Timing Options

To specify timing options that apply during Verilog elaboration, do the following:

1. In the AMS Options form, select the *Timing* tab.

2. Scroll down to the *VERILOG TIMING OPTIONS* group box.



VERILOG TIMING OPTIONS

Delay mode	None
Error pulse filtering	None
Disable enhanced timing features	☐
Ignore notifiers in timing checks	☐
Disallow negative values in checks	☐
Filter cancelled events to e state	☐
Enable PATHPULSE$ specparams	☐
Specify pulse reject limit for path and interconnect	☐
Percentage of delay	100
Specify pulse reject limit for interconnect only	☐
Percentage of delay	100
Specify pulse error limit for path and interconnect	☐
Percentage of delay	100
Specify pulse error limit for interconnect only	☐
Percentage of delay	100
Specify relax tcheck data	☐
limit by(%)	0
Specify relax tcheck reference	☐
limit by(%)	0

3. (Optional) In the *Delay mode* drop-down combo box, select one of the following global digital delay modes for Verilog portions of the design hierarchy:

Zero	Use zero delay This value applies the `-delay_mode zero` option to the `ncelab` command.
Unit	Use unit delay This value applies the `-delay_mode unit` option to the `ncelab` command.
Path	Use path delay, except for modules with no module path delay This value applies the `-delay_mode path` option to the `ncelab` command.
Distributed	Use distributed delay This value applies the `-delay_mode distributed` option to the `ncelab` command.

4. (Optional) In the *Error pulse filtering* drop-down combo box, select one of the following error-pulse filtering options for Verilog portions of the design hierarchy:

On-detect	Use on-detect filtering of error pulses This value applies the `-epulse_ondetect` option to the `ncelab` command.
On-event	Use on-event filtering of error pulses This value applies the `-epulse_onevent` option to the `ncelab` command.

5. (Optional) Turn on *Disable enhanced timing features*.

   You turn on enhanced timing features using special properties in a specify block.

   This option applies the `-disable_enht` option to the `ncelab` command.

6. (Optional) Turn on *Ignore notifiers in timing checks* to tell the elaborator to ignore notifiers in timing checks.

   This option applies the `-nonotifier` option to the `ncelab` command.

7. (Optional) Turn on *Disallow negative values in checks* to disallow negative values in `$setuphold` and `$recrem` timing checks in the Verilog description and in `SETUPHOLD` and `RECREM` timing checks in SDF annotation.

   This option applies the `-noneg_tchk` option to the `ncelab` command.

8. (Optional) Turn on *Filter canceled events to e state* to filter cancelled events (negative pulses) to the e state.

   This option applies the `-epulse_neg` option to the `ncelab` command.

9. (Optional) Turn on *Enable PATHPULSE$ specparams* to enable `PATHPULSE$` specparams, which are used to set module path pulse control on a specific module or on specific paths within modules.

   This option applies the `-pathpulse` option to the `ncelab` command.

10. (Optional) To specify a pulse reject limit for paths and interconnects, do the following:

    a. Turn on *Specify pulse reject limit for path and interconnect* to activate the corresponding *Percentage of delay* field.

    b. In the *Percentage of delay* field, type the percentage of delay for the pulse reject limit for module paths and interconnects.

    This option applies the `-pulse_r` option to the `ncelab` command.

11. (Optional) To specify a pulse reject limit for interconnects only, do the following:

    a. Turn on *Specify pulse reject limit for interconnect only* to activate the corresponding *Percentage of delay* field.

    b. In the *Percentage of delay* field, type the percentage of delay for the pulse reject limit for interconnects only.

    This option applies the `-pulse_int_r` option to the `ncelab` command.

12. (Optional) To specify a pulse error limit for paths and interconnects, do the following:

    a. Turn on *Specify pulse error limit for path and interconnect* to activate the corresponding *Percentage of delay* field.

    b. In the *Percentage of delay* field, type the percentage of delay for the pulse error limit for module paths and interconnects.

    This option applies the `-pulse_e` option to the `ncelab` command.

13. (Optional) To specify a pulse error limit for interconnects only, do the following:

   a. Turn on *Specify pulse error limit for interconnect only* to activate the corresponding *Percentage of delay* field.

   b. In the *Percentage of delay* field, type the percentage of delay for the pulse error limit for interconnects only.

   This option applies the `-pulse_int_e` option to the `ncelab` command.

14. (Optional) To extend the violation regions by a specified percentage to create an overlap so that the negative timing check algorithm can converge, do one of the following:

   ❑ Turn on *Specify relax tcheck data* and type the percentage in the corresponding *limit by(%)* field.

     This option applies the `-extend_tcheck_data_limit` option to the `ncelab` command.

   ❑ Turn on *Specify relax tcheck reference* and type the percentage in the corresponding *limit by(%)* field.

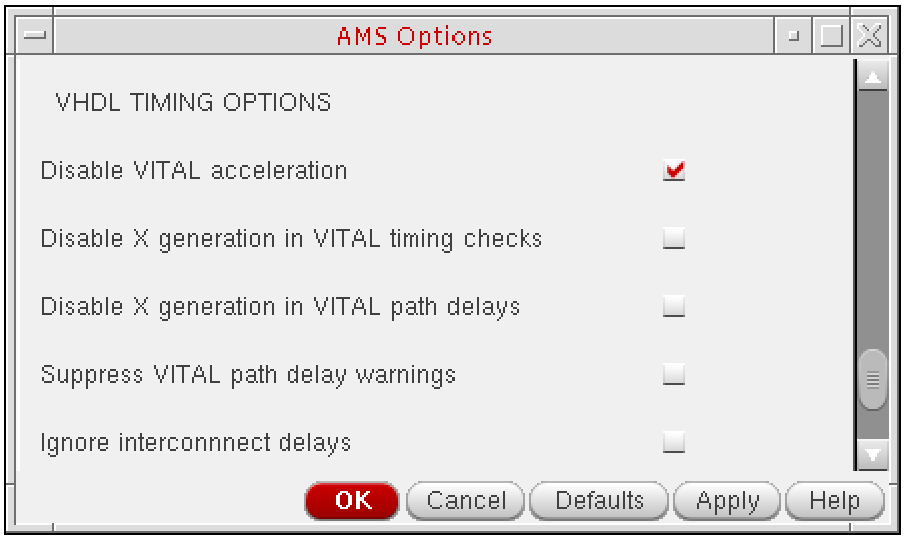
     This option applies the `-extend_tcheck_reference_limit` option to the `ncelab` command.

15. Click *OK*.

Specifying VHDL Timing Options

To specify timing options that apply during VHDL elaboration, do the following:

1. In the AMS Options form, select the *Timing* tab.
2. Scroll down to the *VHDL TIMING OPTIONS* group box.



VHDL TIMING OPTIONS

Disable VITAL acceleration	✓
Disable X generation in VITAL timing checks	
Disable X generation in VITAL path delays	
Suppress VITAL path delay warnings	
Ignore interconnnect delays	

3. (Optional) Turn on *Disable VITAL acceleration* to suppress the acceleration of VITAL level 1 compliant cells.

   This option applies the `-novitalaccl` option to the `ncelab` command.

4. (Optional) Turn on *Disable X generation in VITAL timing checks* to turn off X generation in accelerated VITAL timing checks.

   This option applies the `-no_tchk_xgen` option to the `ncelab` command.

5. (Optional) Turn on *Disable X generation in VITAL path delays* to turn off X generation in accelerated VITAL pathdelay procedures.

   This option applies the `-no_vpd_xgen` option to the `ncelab` command.

6. (Optional) Turn on *Suppress VITAL path delay warnings* to turn off warning messages from accelerated VITAL path delay procedures.

   This option applies the `-no_vpd_msg` option to the `ncelab` command.

7. (Optional) Turn on *Ignore interconnect delays* to turn off recognition of input path delays in a VITAL level 1 cell and uses the non-delayed input signals directly.

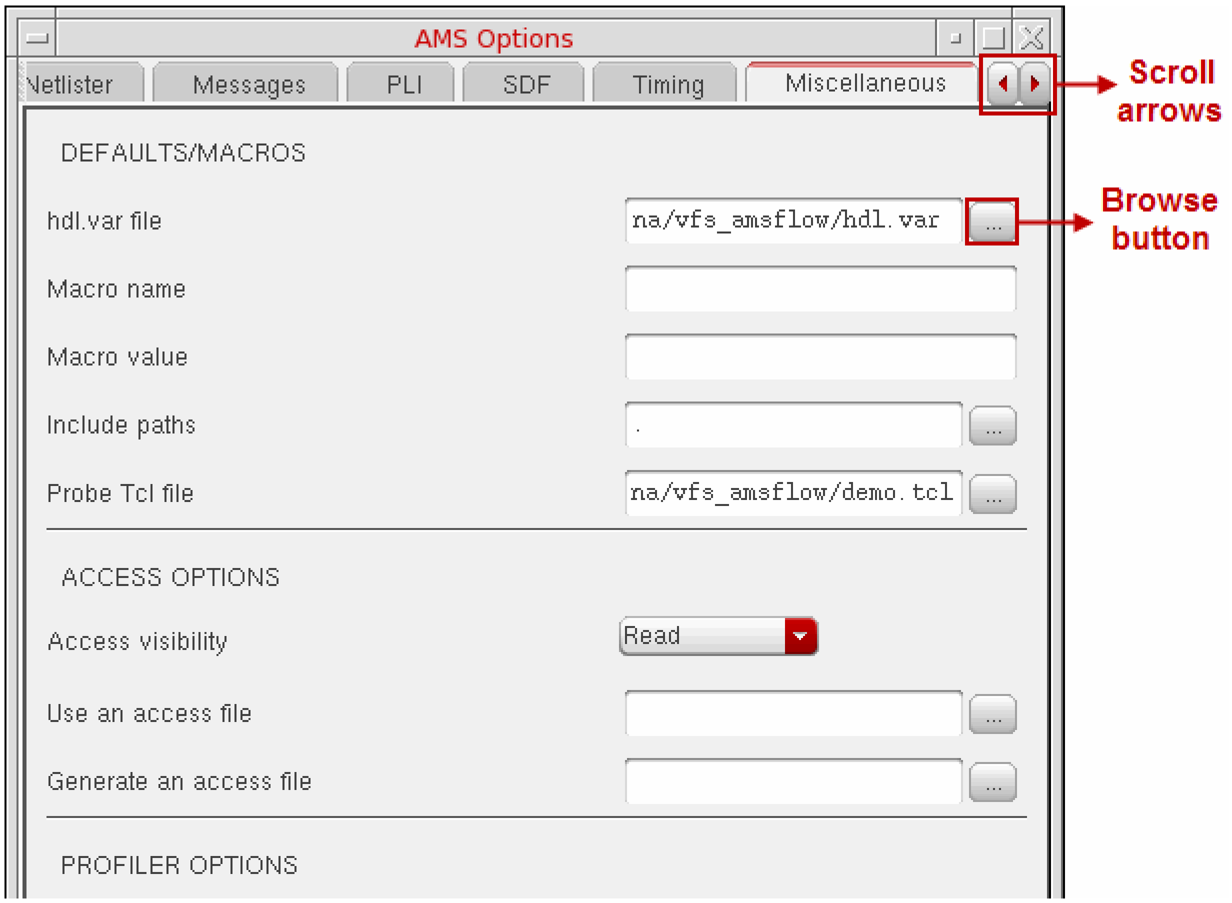
   This option applies the `-noipd` option to the `ncelab` command.

8. Click *OK*.

Specifying Access Options

To specify read, write, and connectivity access options for design objects, do the following:

1. In the AMS Options form, select the *Misc* tab.
2. Scroll down to the *ACCESS OPTIONS* group box.



DEFAULTS/MACROS

hdl.var file	na/vfs_amsflow/hdl.var
Macro name	
Macro value	
Include paths	.
Probe Tcl file	na/vfs_amsflow/demo.tcl

ACCESS OPTIONS

Access visibility	Read
Use an access file	
Generate an access file	

PROFILER OPTIONS

3. (Optional) Use the drop-down combo box in the *Access visibility* field to select one of the following choices to set the visibility access for all objects in the design:

Off	No visibility access This setting applies the `-access -r-w-c` option to the `ncelab` command.
Read	Read access This setting applies the `-access +r` option to the `ncelab` command.

Read/Write	Read and write access This setting applies the `-access +r+w` option to the `ncelab` command.
Connectivity	Connectivity access (includes read and write access by default) This setting applies the `-access +c` option to the `ncelab` command.
All	Read, write, and connectivity access This setting applies the `-access +r+w+c` option to the `ncelab` command.

4. (Optional) If you have an access file that sets visibility access for particular instances or portions of a design, use the browse button to the right of the *Use an access file* field to open the Choose form, then navigate to and select the access file.

   Note: Relative paths are relative to the run directory, not to the directory where you started your Cadence software.

   This setting applies the `-afile` option to the `ncelab` command.

5. (Optional) If you want to generate an access file that reflects how Tcl commands or a PLI application accessed each design object during simulation, type the name of a file with a relative or absolute path in the *Generate an access file* field.

   Note: Relatvie paths are relative to the run directory, not to the directory where you started your Cadence software.

   This setting applies the `-genafile` option to the `ncelab` command.

Tip

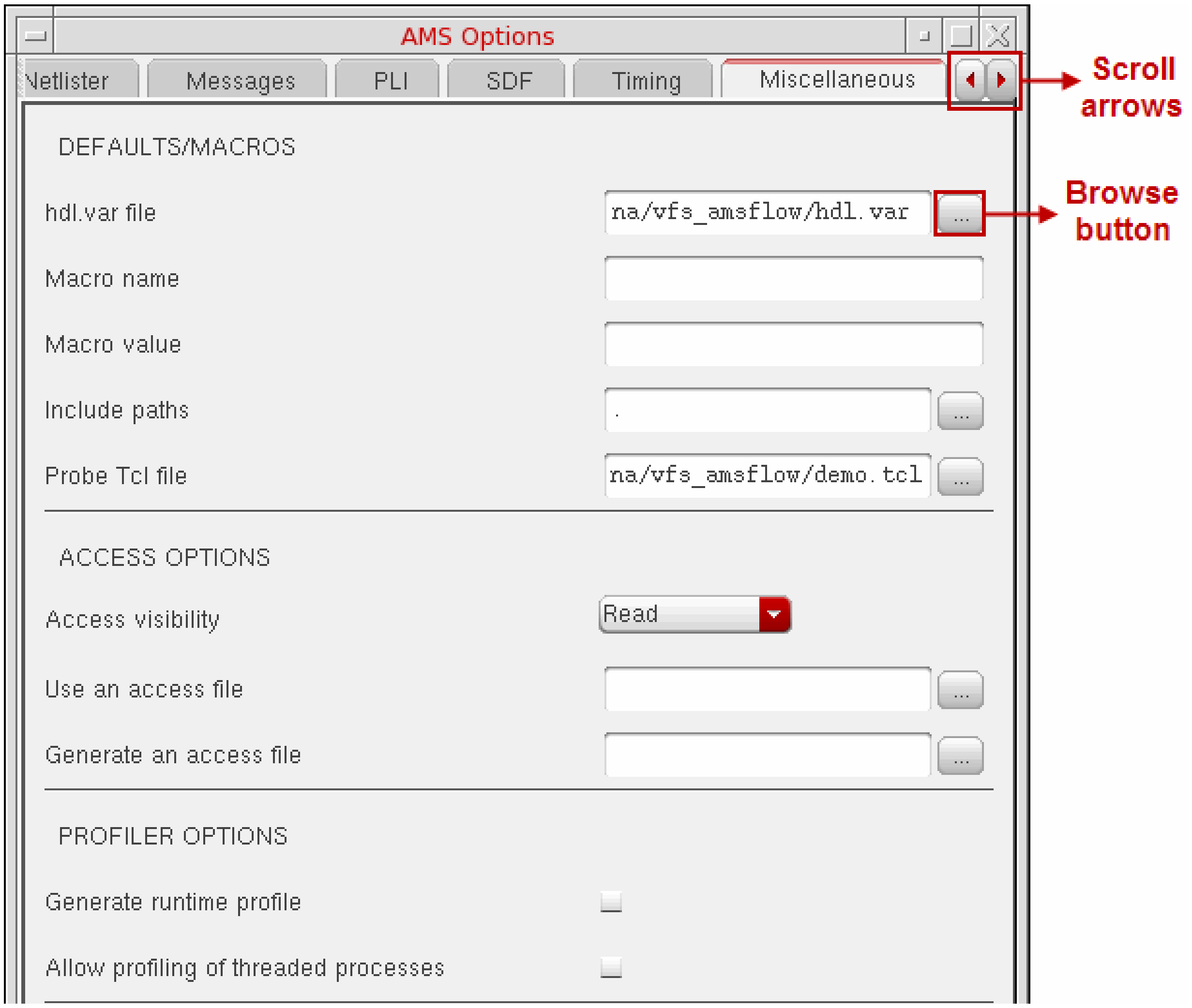
You can later include the generated access file by specifying it in the *Use an access file* field (above).

6. Click *OK*.

Specifying Profiler Options

To specify profiler options, do the following:

1. In the AMS Options form, select the *Misc* tab.
2. Scroll down to the *PROFILER OPTIONS* group box.



DEFAULTS/MACROS

hdl.var file	na/vfs_amsflow/hdl.var
Macro name	
Macro value	
Include paths	.
Probe Tcl file	na/vfs_amsflow/demo.tcl

ACCESS OPTIONS

Access visibility	Read
Use an access file	
Generate an access file	

PROFILER OPTIONS

Generate runtime profile	☐
Allow profiling of threaded processes	☐

3. (Optional) To generate a runtime profile of the design, turn on *Generate runtime profile*.

   The simulator writes profiling information to the `ncprof.out` file only after you exit the simulator session. Depending on the simulator you are using, this command might be restricted to digital only.

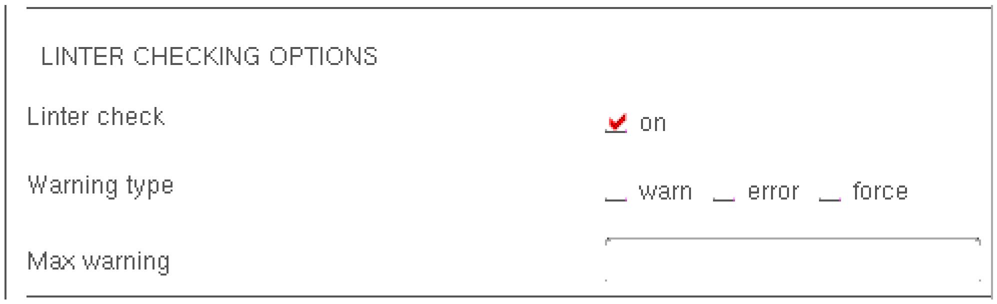
   This setting applies the `-profile` option to the `ncsim` command.

4. (Optional) To allow profiling of threaded processes, turn on *Allow profiling of threaded processes*.

   This setting applies the `-profthread` option to the `ncsim` command.

Specifying Linter Checking Options

1. In the AMS Options form, select the *Misc* tab.
2. Scroll down to the *LINTER CHECKING OPTIONS* group box.



LINTER CHECKING OPTIONS

Linter check	✓ on
Warning type	warn error force
Max warning	

3. To enable linter check, turn on *Linter check* option and pass `-ahdllint` option to the command line.
4. (Optional) Select any of the*Warning type* options and pass `-ahdllint=xxx` option to the command line. `xxx` refers to *warn*, *error*, or *force*.
5. (Optional) Select *Max warning* to specify number input indicating the number of warning messages that will be printed in linter.log. By default, the value is `null`. This text field accepts only the values greater than or equal to zero. If the *Max warning* field remains empty, the option passed to the command line is,

   ```
   -ahdllint=warn -ahdllint_log=../psf/linter.log
   ```

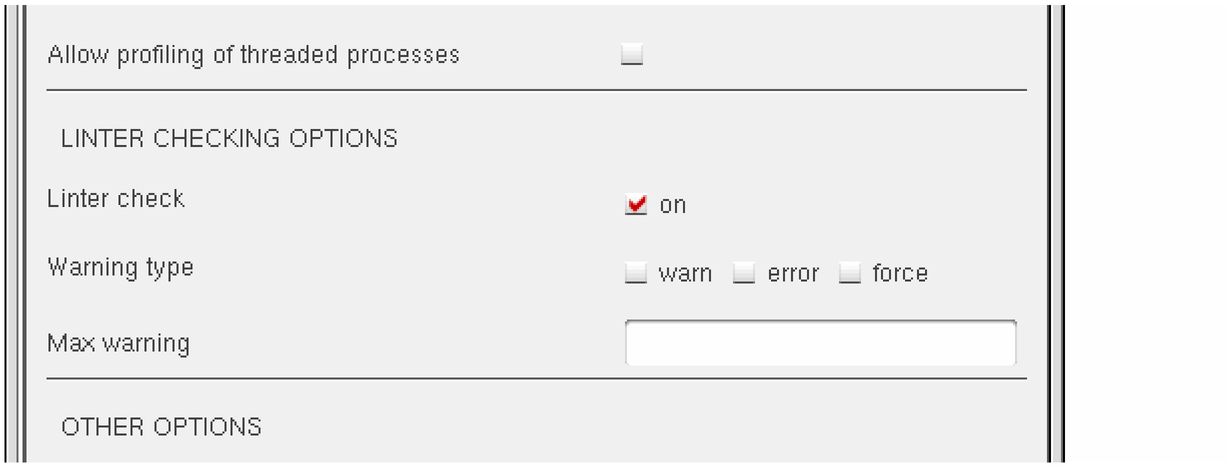
   If the *Max warning* field has a number input, the option passed to the command line is,

   ```
   -ahdllint=warn -ahdllint_maxwarn=3 -ahdllint_log=../psf/linter.log
   ```

Specifying Other Options

1. In the AMS Options form, select the *Misc* tab.

2. Scroll down to the *OTHER OPTIONS* group box.



Allow profiling of threaded processes ☐

LINTER CHECKING OPTIONS

Linter check	✓ on
Warning type	☐ warn ☐ error ☐ force
Max warning	

OTHER OPTIONS

3. (Optional) If you want the elaborator and simulator to ignore source file timestamp information when you use the `-update` option, turn on *Ignore source file timestamps when using -update*.

   This setting applies the `-nosource` option to the `ncelab` and `ncsim` commands.

4. (Optional) If you want to turn on VITAL compliance checking, turn on *Enable VITAL checks (VHDL compiler)*.

   This setting applies the `-novitalcheck` option to the `ncvhdl` command.

5. (Optional) If you want enable VHDL 93 features during compilation and elaboration, turn on *Enable VHDL 93 features*.

   This setting applies the `-v93` option to the `ncvhdl` and `ncelab` commands.

6. (Optional) If you want to apply a more relaxed binding search order, turn on *Enable relaxed VHDL interpretation*.

   This option makes design units visible for default binding when the design units exist in a library that does not have a corresponding `LIBRARY` declaration in the VHDL source and when the design units do not exist in the work library.

   This setting applies the `-relax` option to the `ncvhdl` and `ncelab` commands.

7. Click *OK*.

6

Performing Miscellaneous Tasks in the AMS Designer Environment

See the following topics for details about miscellaneous tasks you can perform in the Virtuoso® AMS Designer environment:

Displaying the Netlist

To display the design netlist in a viewing window, do the following:

➤ Choose *AMS – Miscellaneous – Display Netlist*.

The view-only design netlist (`runDirectory/netlist/completeDesignInfo.ckt`) appears in a viewing window.

Note: This menu selection performs the same task as clicking *Display Netlist* on the Netlist and Run form.

Loading State Files

To load an ADE state file in the AMS Designer environment, do the following:

➤ Choose *AMS – Miscellaneous – Load State*.

The Loading State form appears.

For details about this form, see “Environment Setup” in the *Virtuoso Analog Design Environment L User Guide*.

Note: For information on state files, see the *Virtuoso Analog Design Environment L User Guide*.

Saving State Files

To save an ADE state file in the AMS Designer environment, do the following:

➤ Choose *AMS – Miscellaneous – Save State*.

The Saving State form appears.

For details about this form, see “Environment Setup” in the *Virtuoso Analog Design Environment L User Guide*.

Note: For information on state files, see the *Virtuoso Analog Design Environment L User Guide*.

Generating the AMS Netlist for a Cell

To generate the AMS netlist for an individual cell in your design hierarchy, do the following:

1. In the Virtuoso® Hierarchy Editor, select the *Table View*.
2. On the *Table View* tab, right click the cell for which you want to generate an AMS netlist and select *Generate AMS Netlist*.

   Netlisting messages appear in the CIW.

   For example, you might see messages like the following:

```
Info: Verilog-AMS netlist successfully written to
        /user/name/project/library/cell/schematic/verilog.vams.
Info: Found 0 errors and 10 warnings.
```

Viewing the AMS Netlist for a Cell

To view the AMS netlist for an individual cell in your design hierarchy, do the following:

1. In the Virtuoso® Hierarchy Editor, select the *Table View*.
2. On the *Table View* tab, right click the cell for which you want to view the AMS netlist and select *View AMS Netlist*.

   The netlist (`lib/cell/view/netlist`) appears in a viewing window.

   For example, you might right-click the row for `myLibrary/myDecoder/behavioral` and see `myLibrary/myDecoder/behavioral/verilog.v` appear in a viewing window. Or, you might right-click the row for `myLibrary/myCell/schematic` and see `myLibrary/myCell/schematic/verilog.vams` appear in a viewing window.

Compiling the AMS Netlist for a Cell

To compile the AMS netlist for an individual cell in your design hierarchy, do the following:

1. In the Virtuoso® Hierarchy Editor, select the *Table View*.
2. On the *Table View* tab, right click the cell for which you want to compile the AMS netlist and select *Compile Netlist*.

   Upon successful compilation, a message like the following appears in the CIW:

```
Info: Verilog-AMS file
        /user/name/project/library/cell/schematic/verilog.vams
successfully compiled.
```

7 Using Design Configurations

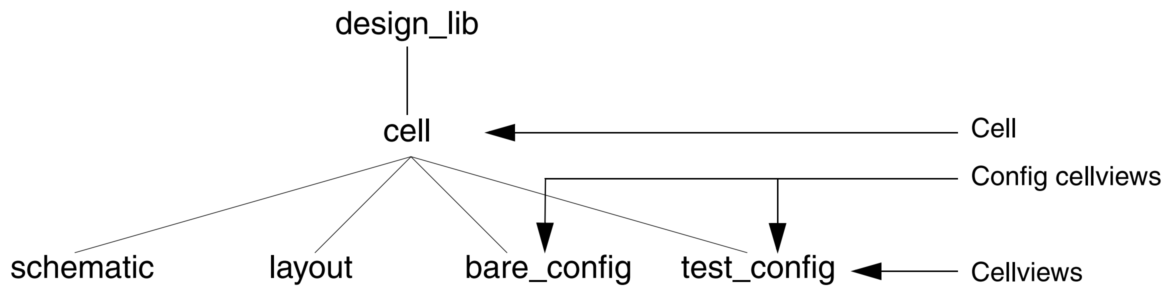
You can use the Virtuoso® Hierarchy Editor to specify the cellviews you want to use in your design. A *configuration* is a set of rules that defines which cellviews under a cell are part of the design for a given purpose (such as netlisting). Using the Virtuoso Hierarchy Editor, you can view the hierarchy of these cellviews.

See the following topics for more information:

- Understanding Configurations on page 174
- Creating a Config Cellview on page 175
- Using VHDL Design Units in a Configuration on page 176
- Netlisting to Make HDL Design Unit Information Current on page 176
- Using a Configuration on page 177

Understanding Configurations

A configuration (or *config*) is a view of the cell (or cellview). You can have different config cellviews for different purposes.



Important

To simulate your design in the AMS Designer environment, you must have a top-level config cellview. The top-level config can contain other config cellviews lower in the hierarchy.

Note: You can run the AMS Designer simulator outside the AMS Designer environment without using a config cellview.

Configurations let you work with different cellviews as your design evolves from concept to finish. For example, you might begin the design process using high-level behavioral models of your design components; later, you might insert modules into test fixtures; finally, you might replace behavioral descriptions with schematics.

You can create configuration rules that define what views to include in the hierarchy at three different levels:

Global level	Using a global view list and stop list
Cell level	Using cell-based view lists—which affect the cell as well as structures below the cell in the hierarchy—and cell bindings
Instance level	Using instance-based view lists—which affect the instance as well as structures lower in the hierarchy—and instance bindings

Creating a Config Cellview

To simulate your design in the AMS Designer environment, you must specify a top-level config cellview for your design hierarchy. The design can contain other config cellviews at lower levels in the hierarchy. You can use the Virtuoso® Hierarchy Editor to create a config cellview as follows:

1. Open the New Configuration form either from the CIW or from the Library Manager.
2. On the New Configration form, click *Use Template*.

   The Use Template form appears.
3. Using the drop-down combo box in the *Name* field, select one of the following templates:
   - ❑ *AMS*, if you are creating and using designs in the AMS Designer environment

     The resulting view list is

     ```
     verilogams veriloga behavioral functional schematic symbol
     ```
   - ❑ *AMS_Compatibility*, if you are working with external text designs

     The resulting view list is

     ```
     stimulus dataflow behavioral behavior functional structure hdl verilogams
     veriloga verilogNetlist system spectre spice cmos.sch cmos_sch schematic
     symbol
     ```

   Note: Symbol views must have associated models that describe the represented device for AMS simulation.
4. Click *OK*.

   All of the design instances and their cell bindings appear in the Virtuoso® Hierarchy Editor window.
5. (Optional) In the *Global Bindings* group box, you can type directly in the fields to edit the lists as required. For example, VHDL users might want to add a wildcard asterisk to *View List*.
6. Choose *View – Update* to check and save the new configuration.

   An Update prompt appears.
7. Click *OK* to save the new config view.

For more information, see the *Virtuoso Hierarchy Editor User Guide*.

Using VHDL Design Units in a Configuration

To instantiate a VHDL design unit in a Verilog®-AMS module, use the `architecture` view of the VHDL design unit in the configuration.

You can also instantiate Verilog-AMS modules in VHDL modules. See “Importing Verilog-AMS Modules into VHDL” in the *Virtuoso AMS Designer Simulator User Guide* for more information.

Netlisting to Make HDL Design Unit Information Current

If you edit an HDL design unit (such as one of the `verilog.vams` netlist files) using a text editor (such as `vi`), you must netlist your design to ensure that the Virtuoso® Hierarchy Editor has up-to-date information. Otherwise, design expansion might not result in what you expect.

Using a Configuration

To use a configuration, you pass it to the elaborator and simulator on the command line. When you use the AMS Designer environment, the software prepares the `ncelab` (elaborator) and `ncsim` (simulator) commands for you. For example:

```
ncelab amslib.top:config amslib.cds_globals:top_config ConnRules_5V_full
       -discipline logic -timescale 1ns/1ns -noparamerr -use5x4vhdl
ncsim amslib.top:config -amslic -analogcontrol top.sce -GUI -input text.tcl
```

The first object after each command is the config view.

The following guidelines apply to using configurations:

Note: When you develop and simulate your design in the AMS Designer environment, the software follows these guidelines automatically.

- Compile the design using the `-use5x` option so that the program creates a lib/cell/view Cadence library structure for the design.

  See “-use5x” in the “Compiling Verilog Source Files with ncvlog” chapter of the *Cadence NC-Verilog Simulator Help* for more information.

- Elaborate designs that contain VHDL design units using the `-use5x4vhdl` option so that the elaborator first uses VHDL language rules to determine a binding and then uses the binding rules from the config view before using the default binding rules.

  See “-use5x4vhdl Option” in the “Elaborating the Design with ncelab” chapter of the *Cadence NC-VHDL Simulator Help* for more information.

- Elaborate using the `-snapshot` option if you want to specify a different location for the simulation snapshot. The default location is the *`library.cell:view`* of the first design unit on the `ncelab` command line.

  See “-snapshot” in the “Elaborating the Design with ncelab” chapter of the *Cadence NC-Verilog Simulator Help* for more information.

8

Netlisting

Virtuoso® AMS Designer offers you the choice of two different netlisters:

- Cellview-based netlister
- OSS netlister

Using Virtuoso® AMS Designer, you can set up the netlister to run automatically whenever you check and save a schematic so that your design is always ready for simulation. You can also run the AMS netlister explicitly when necessary.

Using the OSS Netlister

Important

For OSS-based netlisting, you must run AMS from the Virtuoso® Analog Design Environment (ADE) by selecting *ams* as the simulator, then choosing *Simulation – Netlist and Run Options*.

You use Cadence's Open Simulation System (OSS) netlister to create a single netlist of the entire design hierarchy in the netlist directory. The OSS netlister uses `spectre` simInfo. You do not need to add `ams` simInfo or convert PDKs (as you do if you use the cellview-based netlister).

Spectre and UltraSim circuit simulators use this netlister. The OSS-based netlister is a hierarchical netlister that netlists the entire hierarchy. While having a single netlist (`netlist.vams`) might be helpful for debugging purposes (because everything is in one place), you cannot share individually netlisted or compiled cells when you use the OSS-based netlister. For information about OSS-based netlisting, see the *Open Simulation System Reference*. For important considerations, see "OSS-based AMS Netlister" and "Choosing the Netlister" in the *Virtuoso Analog Design Environment L User Guide*.

Using the Cellview-Based Netlister

You use the cellview-based netlister (the original netlister for AMS Designer) to create a Verilog-AMS netlist for one cell at a time in the lib/cell/view for the cell. If you use the cellview-based netlister, you must add `ams` simInfo to your library.

The AMS cellview-based netlister translates schematic cellviews into Verilog®-AMS netlists. The output of a successful netlisting run is one or more files named `verilog.vams`, each containing a valid Verilog-AMS module that corresponds to a schematic cellview. The netlister places each output file in the corresponding cellview directory.

This chapter focuses primarily on the cellview-based netlister. Using the AMS cellview-based netlister, you can:

- Use an application-specific operation—such as *Check and Save* in the Virtuoso® Schematic Editor—to trigger automatic netlisting of a cellview
- Use the Virtuoso Hierarchy Editor to update a netlist or to netlist an entire design
- Use the AMS Netlister form from the command interpreter window (CIW) to netlist an entire library, all the views of a cell, or a single cellview
- Create a netlist in response to changes in CDF
- Use the `amsdirect` command to netlist a cellview from the UNIX command line
- Use the `amsdesigner` command to netlist a cellview from the UNIX command line

See the following topics for more information:

- Specifying AMS Netlister Options on page 191
- Specifying Netlist Format for Component Instances for AMS Simulation on page 203
- Excluding Parameters from Netlisting on page 203
- Viewing the AMS Netlister Log File on page 208
- Understanding How the Cellview-Based Netlister Operates on page 208

See also the following related topics:

- Chapter 9, "Working with Schematic Designs"
- Chapter 10, "Using External Text Designs"
- Chapter 11, "Using Existing Analog Design Units"
- Chapter 12, "Creating and Using a Test Fixture Module"

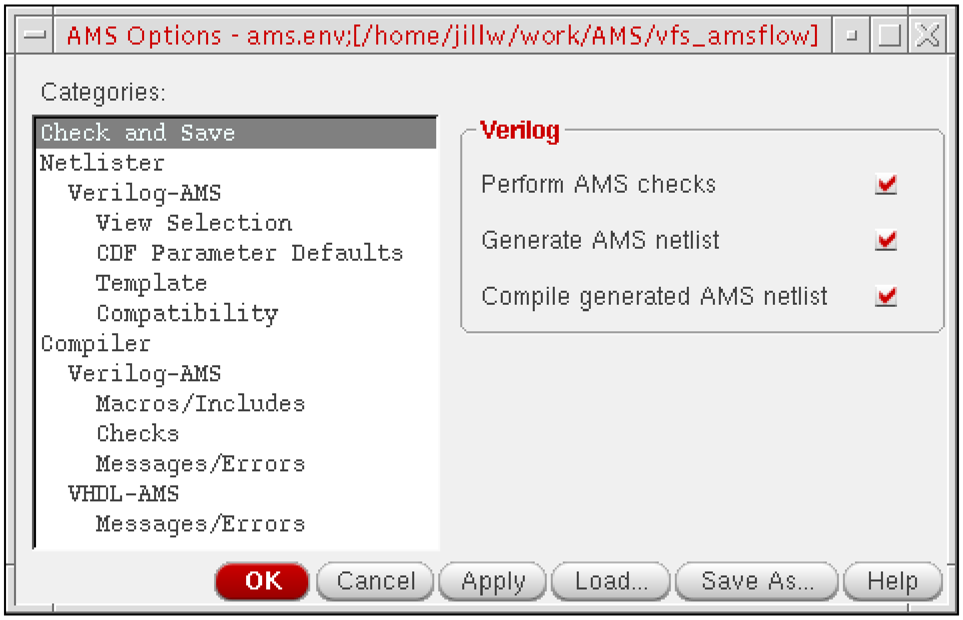
Automatic Netlisting

Using this method, netlisting occurs automatically and transparently whenever you save a cellview that has valid connectivity. You can specify that you want both netlisting and compiling to take place automatically so that the AMS Designer simulator always has the required information and can run quickly.

To specify automatic netlisting, do the following:

1. In the CIW, choose *Tools – AMS – Options*.

   The AMS Options form appears.


AMS Options - ams.env;[/home/jillw/work/AMS/vfs_amsflow]

Categories:

```
Check and Save
Netlister
  Verilog-AMS
    View Selection
    CDF Parameter Defaults
    Template
    Compatibility
Compiler
  Verilog-AMS
    Macros/Includes
    Checks
    Messages/Errors
  VHDL-AMS
    Messages/Errors
```

Verilog

Perform AMS checks	✓
Generate AMS netlist	✓
Compile generated AMS netlist	✓

2. In the *Categories* list area, select *Check and Save*.

3. In the *Verilog* group box, select automated actions for the *Check and Save* operation:

   ❑ *Perform AMS checks*

   ❑ *Generate AMS netlist*

   Note: If you do not turn on *Generate AMS netlist*, when you check-and-save a cellview, the AMS netlister removes any previously-created netlist for the cellview, whether you have enabled AMS or not. This process of removing existing netlists ensures that you do not inadvertently simulate an out-of-date netlist.

   ❑ *Compile generated AMS netlist*

   When you check-and-save a cellview, the software performs the tasks you select.

4. If necessary, select other items in the *Categories* list area and set the options that control the AMS netlister. For more information, see "Specifying AMS Netlister Options" on page 191.
5. Click *OK* to save your settings and close the form.

Now, you can use the Virtuoso® Schematic Editor to create or edit schematic views, then click *Check and Save* to run the AMS netlister automatically. The netlister creates a Verilog-AMS netlist in the cellview directory of your saved schematic view. This netlist is available to all users of the block: None of the users needs to recreate the netlist unless the block changes.

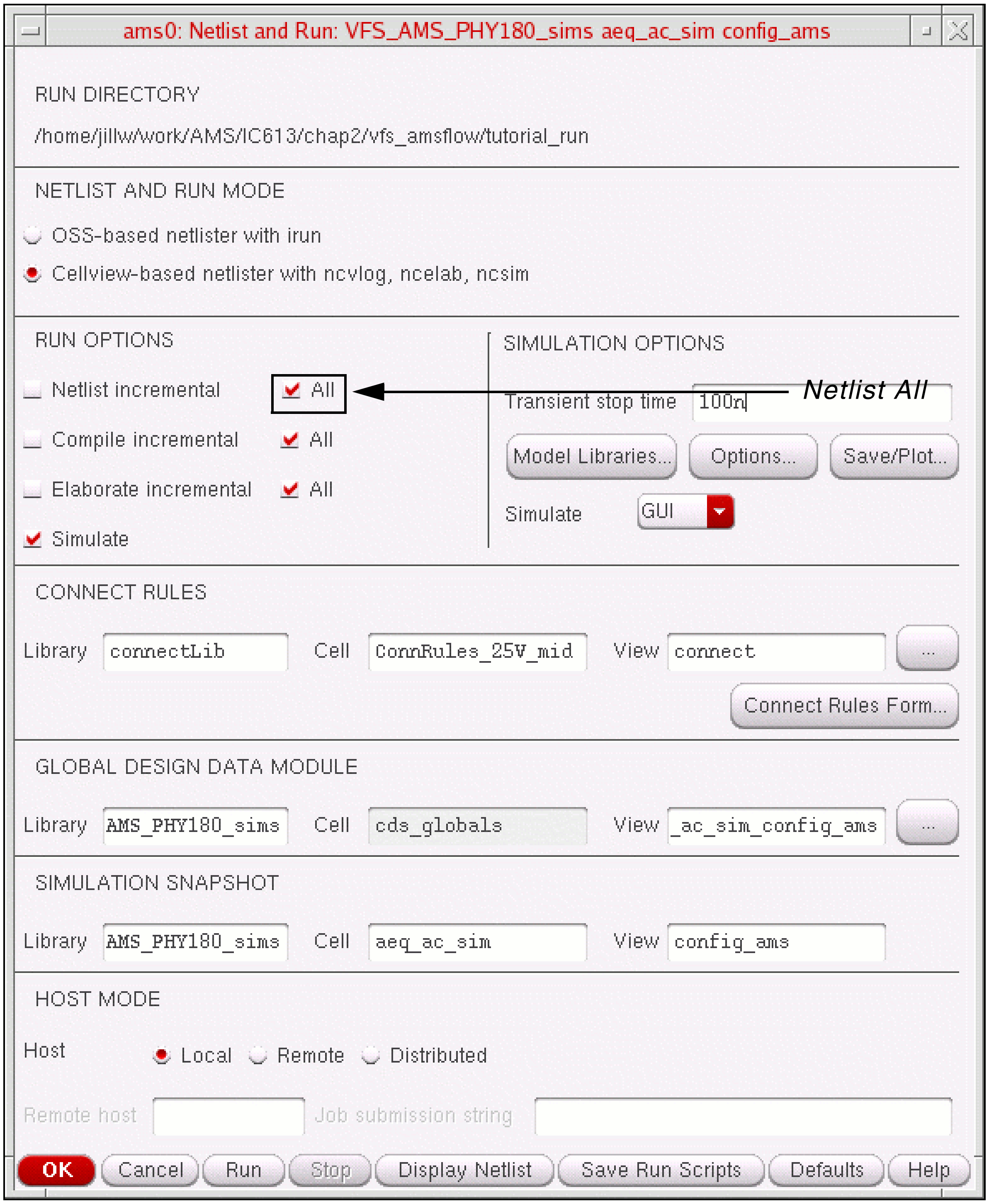
Netlisting the Entire Design

To netlist the entire design, do the following:

1. In the Virtuoso® Hierarchy Editor, choose *AMS – Netlist and Run*.

   The Netlist and Run form appears.

2. In the *RUN OPTIONS* group box, mark the *All* check box for netlisting.



ams0: Netlist and Run: VFS_AMS_PHY180_sims aeq_ac_sim config_ams

RUN DIRECTORY
/home/jillw/work/AMS/IC613/chap2/vfs_amsflow/tutorial_run

NETLIST AND RUN MODE
OSS-based netlister with irun [✓ Cellview-based netlister with ncvlog, ncelab, ncsim]

RUN OPTIONS
- Netlist incremental [✓ All]
- Compile incremental [✓ All]
- Elaborate incremental [✓ All]
- [✓ Simulate]

SIMULATION OPTIONS
Transient stop time: 100n
Simulate: GUI

CONNECT RULES
Library: connectLib Cell: ConnRules_25V_mid View: connect

GLOBAL DESIGN DATA MODULE
Library: AMS_PHY180_sims Cell: cds_globals View: _ac_sim_config_ams

SIMULATION SNAPSHOT
Library: AMS_PHY180_sims Cell: aeq_ac_sim View: config_ams

HOST MODE
Host: [✓ Local] Remote Distributed
Remote host: Job submission string:

When you click *Run* or *Display Netlist*, the program netlists your entire design.

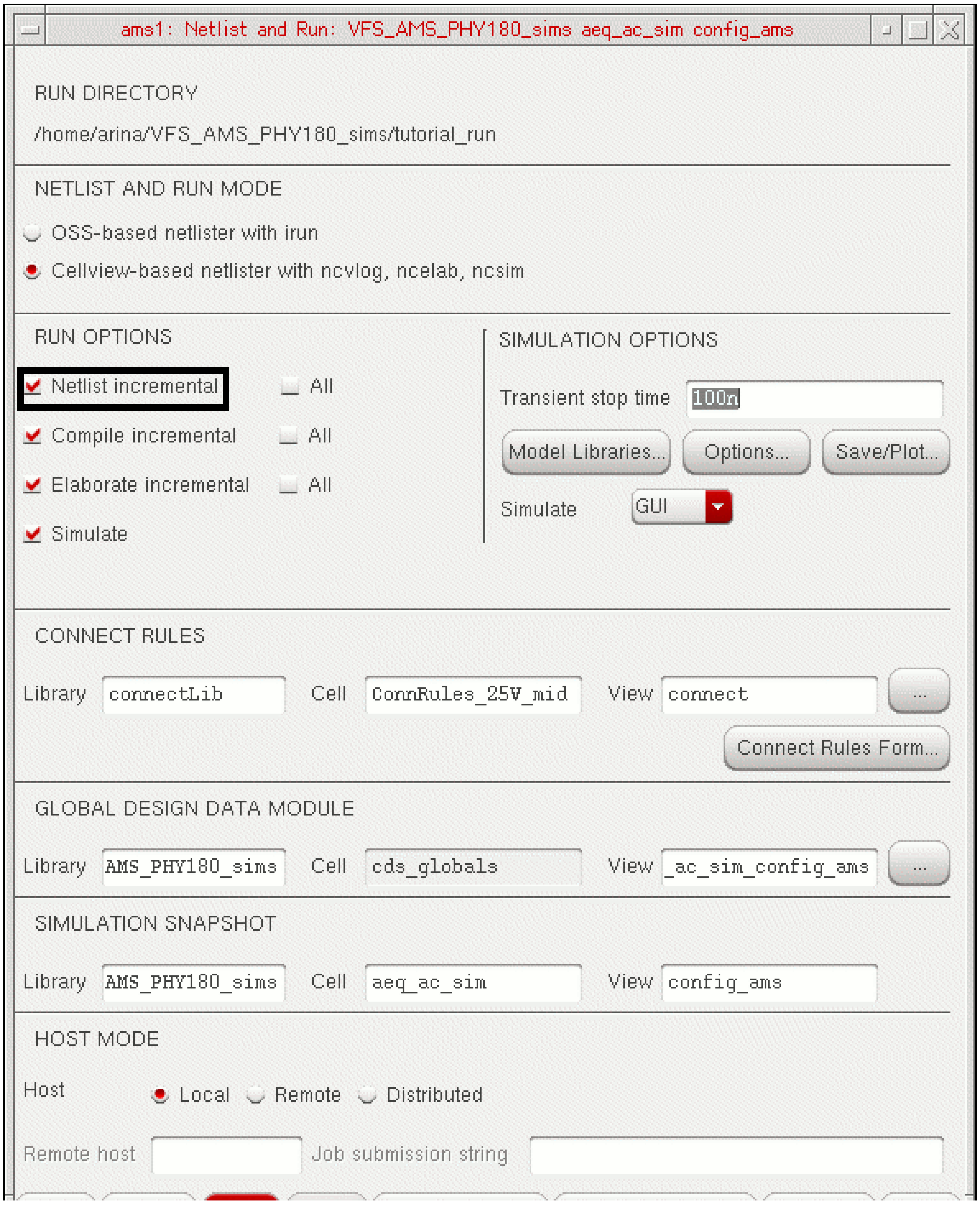
Netlisting Incrementally

To netlist just those parts of the design that have changed, do the following:

1. In the Virtuoso® Hierarchy Editor, choose AMS – *Netlist and Run*.

   The Netlist and Run form appears.

2. In the *RUN OPTIONS* group box, turn on *Netlist incremental*.



ams1: Netlist and Run: VFS_AMS_PHY180_sims aeq_ac_sim config_ams

RUN DIRECTORY

/home/arina/VFS_AMS_PHY180_sims/tutorial_run

NETLIST AND RUN MODE

OSS-based netlister with irun [✓ Cellview-based netlister with ncvlog, ncelab, ncsim]

RUN OPTIONS

- [✓] Netlist incremental — [ ] All
- [✓] Compile incremental — [ ] All
- [✓] Elaborate incremental — [ ] All
- [✓] Simulate

SIMULATION OPTIONS

Transient stop time: 100n

Simulate: GUI

CONNECT RULES

Library: connectLib — Cell: ConnRules_25V_mid — View: connect

GLOBAL DESIGN DATA MODULE

Library: AMS_PHY180_sims — Cell: cds_globals — View: _ac_sim_config_ams

SIMULATION SNAPSHOT

Library: AMS_PHY180_sims — Cell: aeq_ac_sim — View: config_ams

HOST MODE

Host: [✓ Local] Remote Distributed

Remote host: — Job submission string:

When you click *Run* or *Display Netlist*, the program netlists only new or changed cellviews.

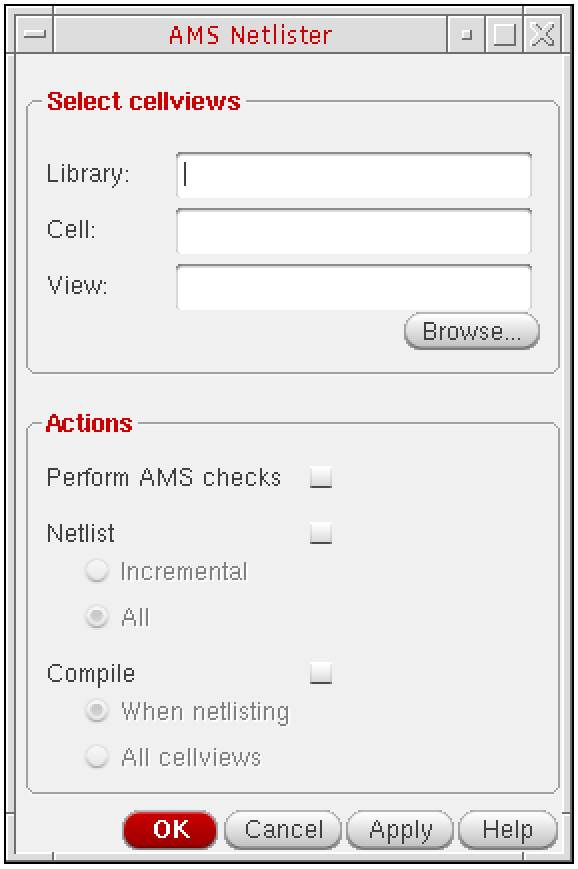
Library Netlisting from the CIW

Using this method, you can check, netlist, and compile an entire library, all the views of a cell, or a single cellview from the command interpreter window (CIW). You can also netlist and compile only new or revised cellviews.

To use library netlisting from the CIW, do the following:

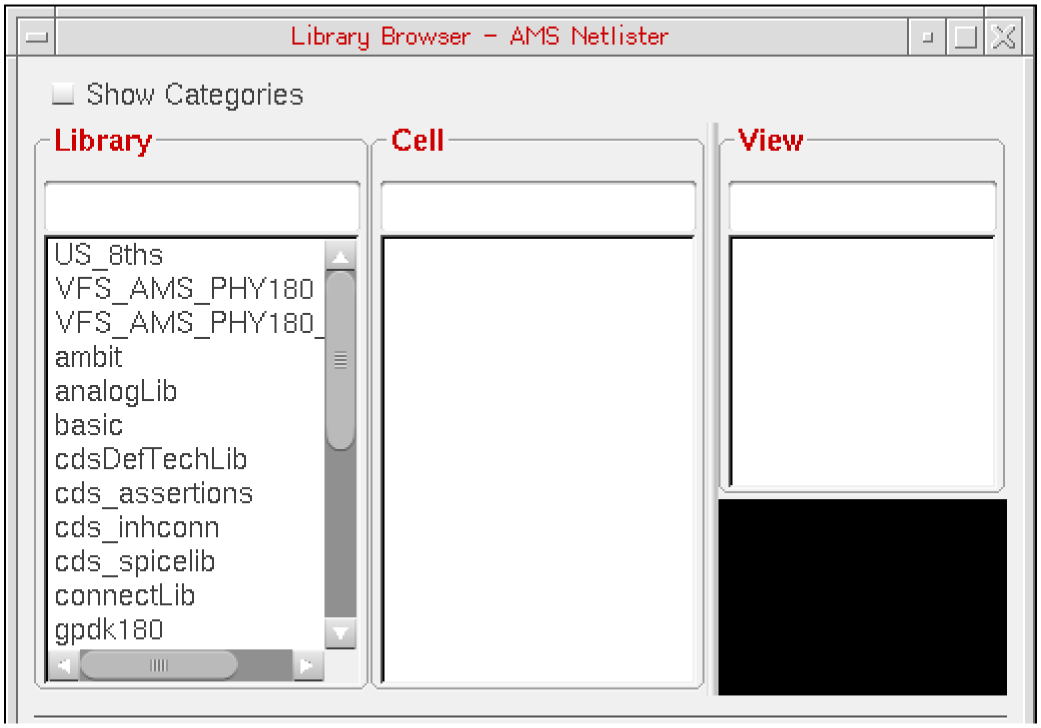
1. In the CIW, choose *Tools – AMS – Netlist*.

   The AMS Netlister form appears.



2. Click *Browse*.

The Library Browser – AMS Netlister form appears.



3. In the *Library* column, select the library containing the cellviews you want to netlist.

   The library name appears in the *Library* field on the AMS Netlister form.

4. (Optional) In the *Cell* column, select a cell.

   The cell name appears in the *Cell* field on the AMS Netlister form.

   If you do not select a cell, the AMS netlister operates on eligible views for every cell in the library.

5. (Optional) In the *View* column, select a view.

   The view name appears in the *View* field on the AMS Netlister form.

   Note: You can further specify view names to process or to exclude. For more information, see Eligible View Types and View Names to Exclude on page 619 and View Names to Process on page 621.

6. In the *Actions* group box on the AMS Netlister form, select what you want the AMS netlister to do with the specified cellviews in the current run. You can choose incremental netlisting if you want to netlist only new or changed cellviews. Any selections you make on this form are for this run only and do not affect the settings on the AMS Options form (for automatic netlisting).

7. Click *OK* to begin the run.

The AMS netlister creates Verilog-AMS netlists according to your specifications.

Netlisting Cells in Response to Changes in CDF

This netlisting method automatically runs the AMS netlister when you use the CDF editor to update the CDF information for a cell.

Netlisting from the UNIX Command Line

Using this method, you can netlist an entire library, all the views of a cell, or a single cellview without starting the graphical user interface. You use the `amsdirect` command, which has the following syntax:

```
amsdirect -LIb libName[ -Cell cellName][ -VIew viewName][ -VERIlog]
        [ -LOg logFileName][ -Incremental][ -Help][ -VERSion]
        [ -CDS_IMPLICIT_TMPDIR implicitTmpDir[ -CDS_IMPLICIT_TMPONLY]]
```

The following table describes the `amsdirect` command options.

amsdirect Option	Description
`-LIb` *`libName`*	Specifies the library containing the cellviews for which you want the netlister to create Verilog-AMS netlists. If you do not also specify a cell using `-cell`, the AMS netlister creates netlists for all eligible views for every cell in the library.
`-Cell` *`cellName`*	Specifies the cell containing the cellviews for which you want the netlister to create Verilog-AMS netlists.
`-VIew` *`viewName`*	Specifies the cellview for which you want the netlister to create a Verilog-AMS netlist. The type of the cellview must be `schematic`, `symbolic`, `maskLayout`, or `netlist`, but the name of the cellview can be any legal name.
`-VERIlog`	Indicates that you want the AMS netlister to generate Verilog-AMS netlists for the processed cellviews.

amsdirect Option	Description
-LOg *logFileName*	Tells the AMS netlister to write messages to *logFileName*. ■ If *logFileName* is an absolute path, the log file is written to *logFileName*. ■ If *logFileName* is a relative path and ❑ `CDS_LOG_PATH` is null, *logFileName* is placed in the current directory ❑ `CDS_LOG_PATH` is non-null, the value of `CDS_LOG_PATH` is prepended to *logFileName*
-Incremental	Tells the AMS netlister to netlist only new or revised cellviews.
-Help	Returns a brief description of the `amsdirect` command and its options.
-VERSion	Returns the version number of the AMS netlister.
-CDS_IMPLICIT_TMPDIR *implicitTmpDir*	Specifies an implicit TMP directory to search for design data and to hold new design data. The *implicitTmpDir* must be an existing directory.
-CDS_IMPLICIT_TMPONLY	Forces the software to look at design data within the *implicit_TmpDir* specified by the `-CDS_IMPLICIT_TMPDIR` option only. If you do not use the `-CDS_IMPLICIT_TMPONLY` option, the elaborator also considers design data found in the master libraries or explicit TMP directories defined by `cds.lib` files.

For example, the following command tells the AMS netlister to generate Verilog-AMS netlists for all of the eligible views in the `mycell` cell:

```
amsdirect -li mylib -cell mycell -veri
```

The following command tells the AMS netlister to netlist the schematic views of all the cells in the `mylib` library:

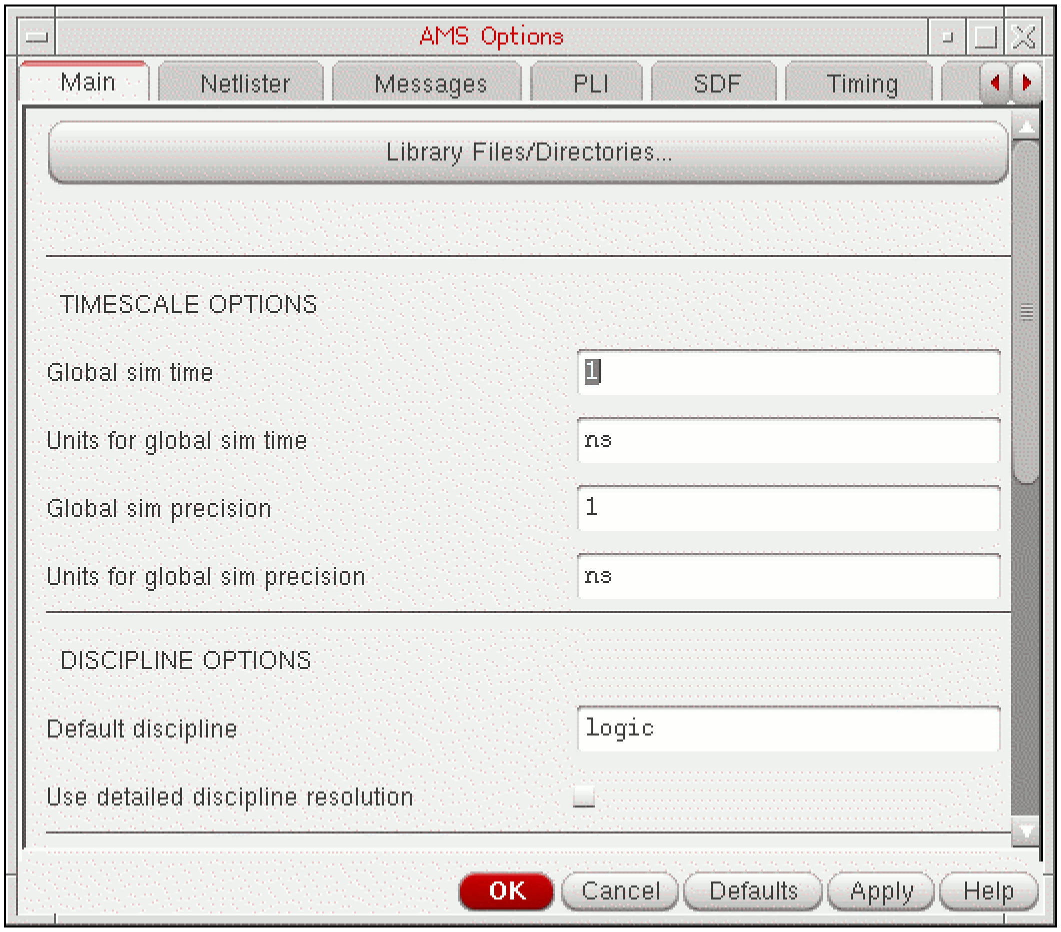
```
amsdirect -lib mylib -view schematic -verilog
```

Specifying AMS Netlister Options

You can use the AMS Options form to specify options for the AMS cellview-based netlister in the AMS Designer environment. To open the AMS Options form, do the following:

➤ Choose *AMS – Detailed Setup – AMS Options*.

The AMS Options form appears.



TIMESCALE OPTIONS

Global sim time	1
Units for global sim time	ns
Global sim precision	1
Units for global sim precision	ns

DISCIPLINE OPTIONS

Default discipline	logic
Use detailed discipline resolution	☐

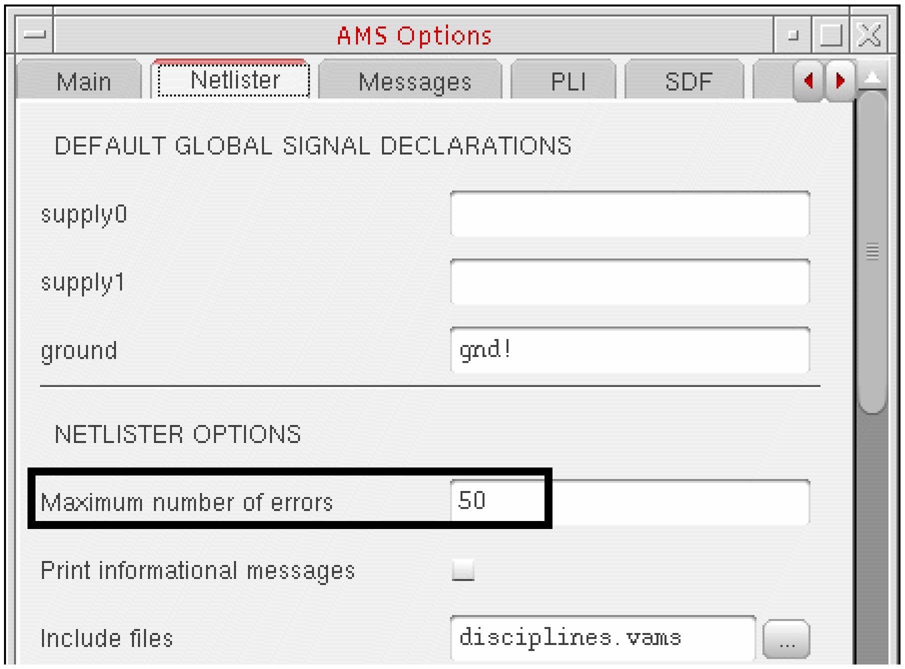
You can specify the following options for the AMS cellview-based netlister:

- Maximum Number of Errors on page 193
- Print Informational Messages on page 194
- Include Files on page 195
- Header Text on page 196
- Default Global Signal Declarations on page 198
- Global Signals on page 199
- Global Design Data Module (cds_globals) on page 200

Maximum Number of Errors

To specify the maximum number of errors the AMS netlister can encounter before it stops processing the design, do the following:

1. In the AMS Options form, select the *Netlister* tab.
2. Scroll down to the *NETLISTER OPTIONS* group box.
3. In the *Maximum number of errors* field, type the maximum number of errors the AMS netlister can encounter before it stops processing the design.



DEFAULT GLOBAL SIGNAL DECLARATIONS

supply0	
supply1	
ground	gnd!

NETLISTER OPTIONS

Maximum number of errors	50
Print informational messages	☐
Include files	disciplines.vams

4. Click *OK*.

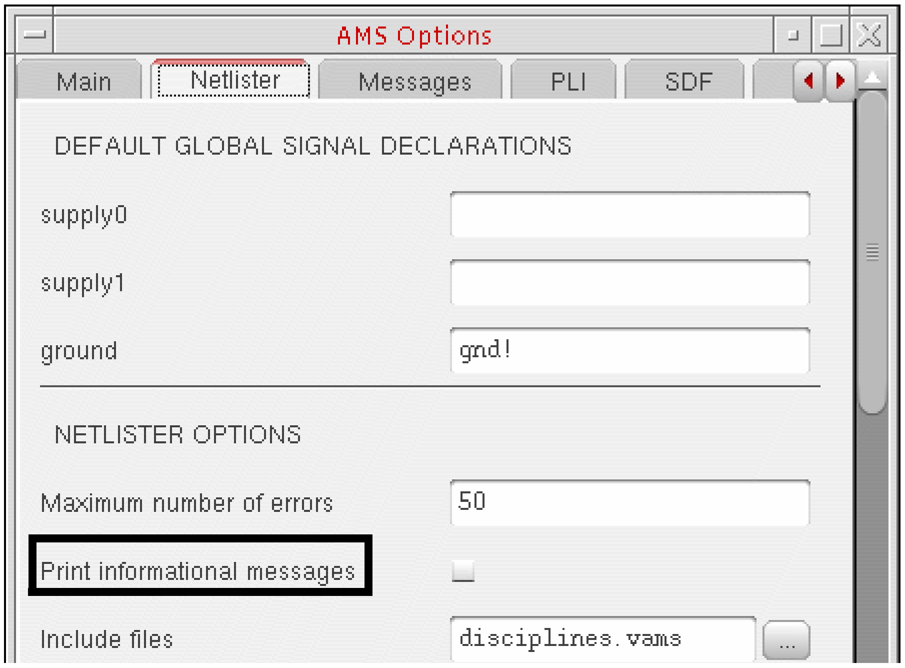
   If the AMS netlister encounters more errors than the number you specified in the *Maximum number of errors* field, it stops processing the design.

   If the AMS netlister encounters any errors, it does not generate a netlist and it removes any existing netlist so that you cannot inadvertently simulate an out-of-date netlist.

Print Informational Messages

To specify that you want the AMS netlister to print informational messages, do the following:

1. In the AMS Options form, select the *Netlister* tab.
2. Scroll down to the *NETLISTER OPTIONS* group box.
3. Turn on *Print informational messages*.



DEFAULT GLOBAL SIGNAL DECLARATIONS

supply0	
supply1	
ground	gnd!

NETLISTER OPTIONS

Maximum number of errors	50
Print informational messages	☐
Include files	disciplines.vams

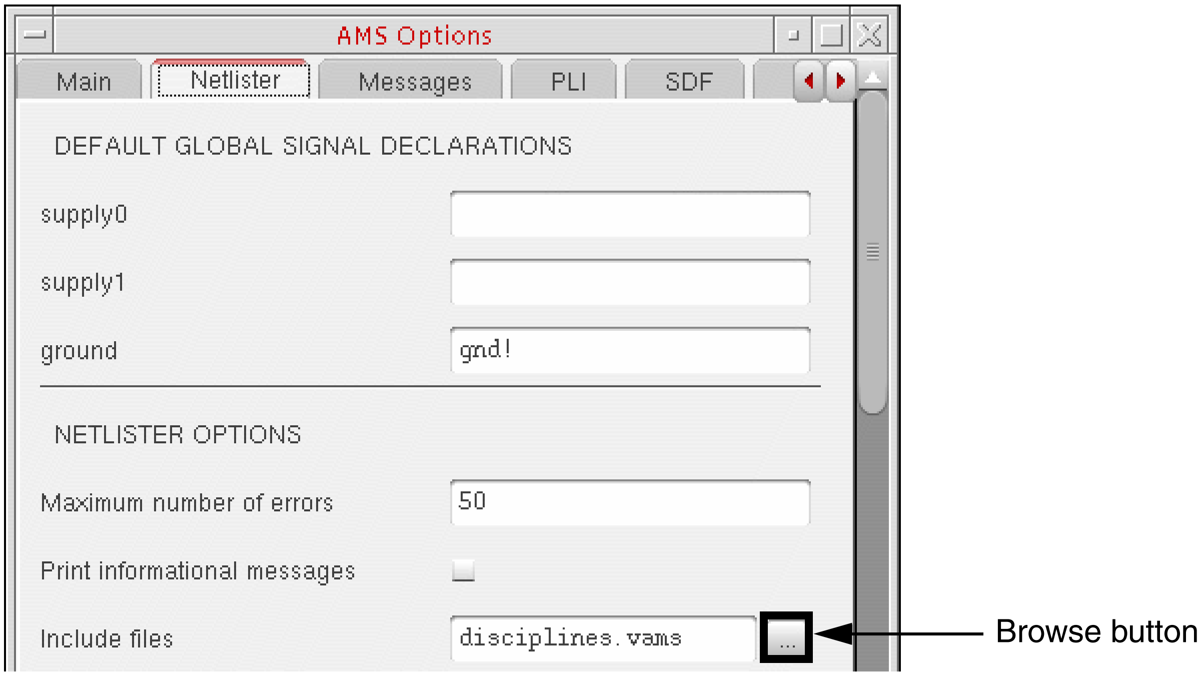
4. Click *OK*.

   The AMS netlister will print more numerous and more extensive informational messages which can help you if you are trying to debug a netlisting problem.

Include Files

To specify files you want the AMS netlister to include in the Verilog-AMS netlist, do the following:

1. In the AMS Options form, select the *Netlister* tab.
2. To the right of the *Include files* field, click the browse button.



DEFAULT GLOBAL SIGNAL DECLARATIONS

supply0	
supply1	
ground	gnd!

NETLISTER OPTIONS

Maximum number of errors	50
Print informational messages	☐
Include files	disciplines.vams

3. On the Choose form that appears, navigate to and select the file you want to include.
4. Click *Open*.

   The path and file name appear in the *Include files* field on the *Netlister* tab.

   If you have more than one include file, you can separate the file names in this field using a space.

   The AMS netlister writes include files to the netlist in the order that they appear in this list. The order is important if you have files that use declarations in another file. For example, if `File2` uses a declaration from `File1`, `File1` must appear above `File2` in the list.

   The AMS netlister writes a `` `include `` directive in each netlist to include each file you specify here. You can specify the directories the AMS netlister searches for include files on the *Misc* tab.

Header Text

You can specify header text that you want the AMS netlister to insert in every Verilog-AMS netlist it generates using one of the following choices:

- Include Header Text from a Particular File on page 196
- Include Header Text That Results from a Script File on page 197

In either case, the AMS netlister inserts the header text you specify after the default header text, which is as follows:

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

Include Header Text from a Particular File

To include header text from a particular file, do the following:

1. In the AMS Options form, select the *Netlister* tab.
2. Using the drop-down combo box in the *Header text* field, select *file*.

   The *Template header file included* field becomes active.
3. To the right of the *Template header file included* field, click the browse button.
4. On the Choose form that appears, navigate to and select the template header file you want to include.

   Note: You can also type the path and name of the text file that contains the header text you want to include. If you specify a relative path, the program resolves that path with respect to the directory where you started the AMS software.
5. Click *OK*.

   The AMS netlister inserts the default header text followed by the contents of the file you specify at the top of each netlist it generates. For example, if you have a file containing the following text:

   ```
   // Module produced by
   // ASIC Team: Ocelot
   // San Jose Development Center
   ```

   the AMS netlister inserts the following text at the top of each generated netlist:

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

   // Module produced by
   // ASIC Team: Ocelot
   // San Jose Development Center
   ```

Include Header Text That Results from a Script File

To include header text that results from a script file, do the following:

1. In the AMS Options form, select the *Netlister* tab.
2. Using the drop-down combo box in the *Header text* field, select *script*.

   The *Template header output from script* field becomes active.
3. To the right of the *Template header file included* field, click the browse button.
4. On the Choose form that appears, navigate to and select the script file that contains the commands that will generate your header text.

   Note: You can also type the path and name of the script file that contains the commands that will generate your header text. If you specify a relative path, the program resolves that path with respect to the directory where you started the AMS software.
5. Click *OK*.

   The AMS netlister inserts the default header text followed by the text from the script at the top of each netlist it generates. For example, if you have a file containing the following text:

   ```
   echo '// Module produced by:'
   echo '// ASIC Interactive, Ltd.'
   printf '// (c) '
   date '+DATE: %m/%d/%y%n'
   ```

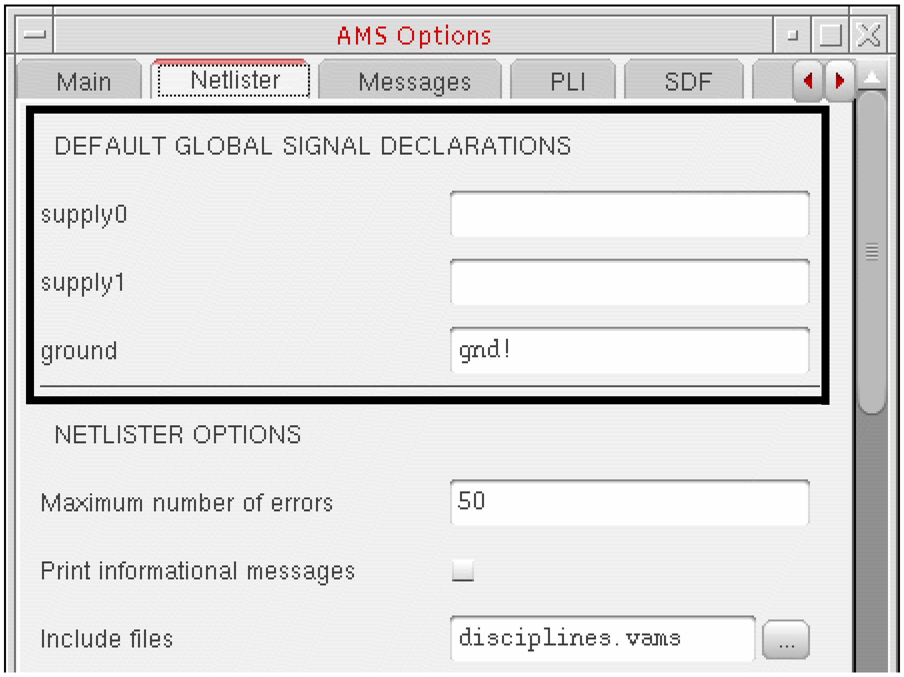
   the AMS netlister inserts the following text at the top of each generated netlist:

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

   // Module produced by:
   // ASIC Interactive, Ltd.
   // (c) DATE: 10/10/01
   ```

Default Global Signal Declarations

To specify global default signal declarations, do the following:

1. In the AMS Options form, select the *Netlister* tab.



DEFAULT GLOBAL SIGNAL DECLARATIONS	
supply0	
supply1	
ground	gnd!
NETLISTER OPTIONS	
Maximum number of errors	50
Print informational messages	☐
Include files	disciplines.vams

2. In the *DEFAULT GLOBAL SIGNAL DECLARATIONS* group box, type a space-separated list of names of global signals in the fields according to how you want to declare them:

   ❑ In the *supply0* field, type names of global signals that you want to declare as type `supply0`.

   ❑ In the *supply1* field, type names of global signals that you want to declare as type `supply1`.

   ❑ In the *ground* field, type names of global signals that you want to declare as type `ground`.

3. Click *OK*.

   AMS Designer assigns default wire types to signals it encounters whose names match those you have typed in these fields. For example, if you type `vdd! dvdd!` in the *supply1* field, when AMS Designer encounters a global signal named `dvdd!`, it assigns `supply1` as the wire type for `dvdd!`. You can override these assignments using the Global Signals form.

Global Signals

A global signal is a signal that connects by name to other signals with the same name across all levels of the design hierarchy without using explicit pin connections. Global signals can come from master schematic data or from out-of-module references in master Verilog (digital) or Verilog-AMS HDL data. In the schematic editor, any signal name that ends with an exclamation point (`!`) is a global signal. AMS Designer is aware of global signals that come from master schematic data only. You can use the Global Signals form to declare out-of-module reference signals in the master Verilog (digital) or Verilog-AMS HDL data.

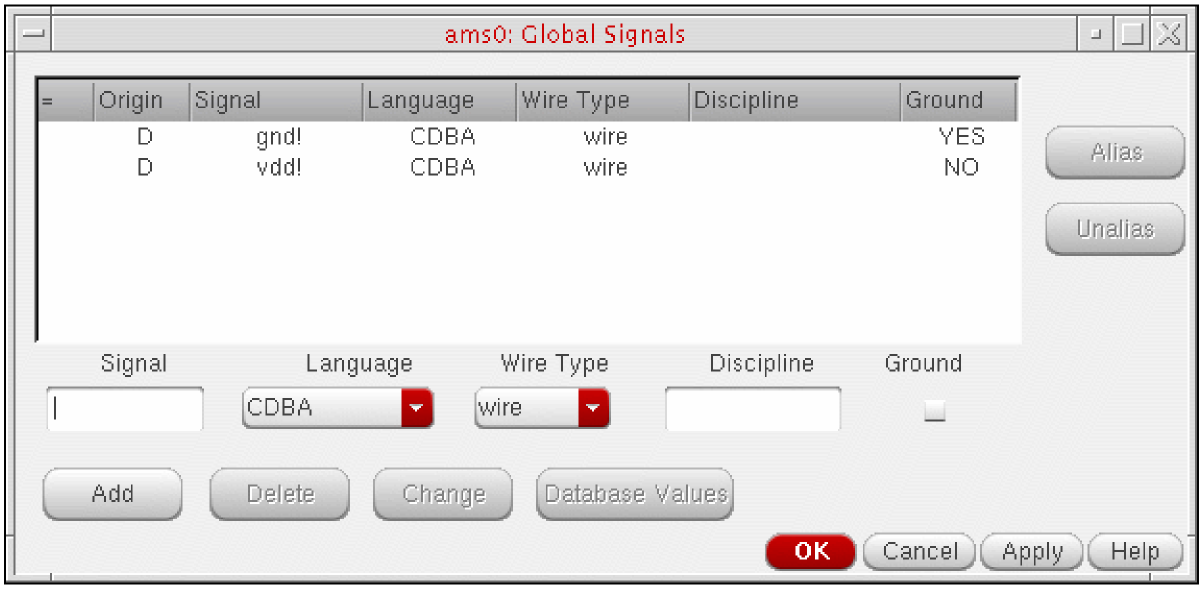
You can use global signal names (`cds_globals.*`) in a VHDL-AMS scope using inherited connection attributes in that scope. You can reference only Verilog-AMS global signals. See “Using Inherited Connections in VHDL-AMS in the *Virtuoso AMS Designer Simulator User Guide* for more information.

You can add, remove, alias, and unalias global signals using the Global Signals form. For information about the Global Signals form, see “Working with Global Signals in AMS” in the *Virtuoso Analog Design Environment L User Guide*.

To open the Global Signals form, do the following:

➤ In the Virtuoso® Hierarchy Editor, choose *AMS – Detailed Setup – Global Signals*.

The Global Signals form appears.



=	Origin	Signal	Language	Wire Type	Discipline	Ground
	D	gnd!	CDBA	wire		YES
	D	vdd!	CDBA	wire		NO

Note: You can also open this form by clicking *Global Signals* on the *Netlister* tab in the AMS Options form.

For signals that come from master schematic data, a *D* appears in the *Origin* column.

When you click *OK*, the program creates and compiles the `cds_globals` module if it does not already exist, or regenerates and recompiles the `cds_globals` module if it does already exist. See "Global Design Data Module (cds_globals)" on page 200 for information about the `cds_globals` module.

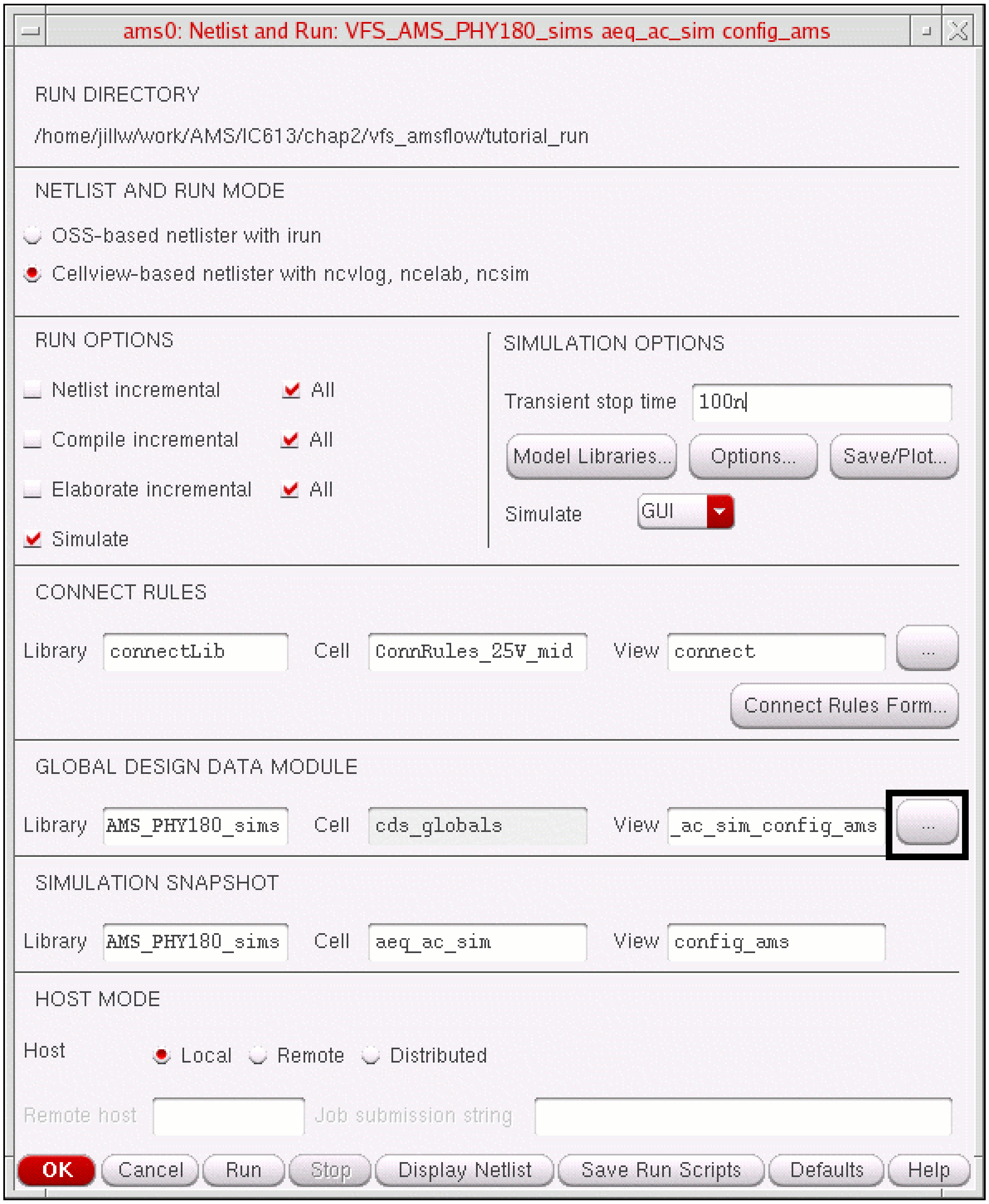
Global Design Data Module (cds_globals)

AMS Designer automatically generates the `cds_globals` cell that contains global signals and design variables. If you have your own set of design variable values, for example, you might want to specify your own `cds_globals` module. While the cell name, `cds_globals`, is fixed, you can specify a library name and view name as follows:

1. In the Virtuoso® Hierarchy Editor, choose *AMS – Netlist and Run*.

   The Netlist and Run form appears.

2. In the *GLOBAL DESIGN DATA MODULE* group box, click the browse button.



ams0: Netlist and Run: VFS_AMS_PHY180_sims aeq_ac_sim config_ams

RUN DIRECTORY
/home/jillw/work/AMS/IC613/chap2/vfs_amsflow/tutorial_run

NETLIST AND RUN MODE
- OSS-based netlister with irun
- [✓ Cellview-based netlister with ncvlog, ncelab, ncsim]

RUN OPTIONS
- Netlist incremental [✓ All]
- Compile incremental [✓ All]
- Elaborate incremental [✓ All]
- [✓ Simulate]

SIMULATION OPTIONS
- Transient stop time: 100n
- Simulate: GUI

CONNECT RULES

Library: connectLib	Cell: ConnRules_25V_mid	View: connect

GLOBAL DESIGN DATA MODULE

Library: AMS_PHY180_sims	Cell: cds_globals	View: _ac_sim_config_ams

SIMULATION SNAPSHOT

Library: AMS_PHY180_sims	Cell: aeq_ac_sim	View: config_ams

HOST MODE
- Host: [✓ Local] Remote Distributed
- Remote host
- Job submission string

3. Using the Library Browser form that appears, select a *Library*, *Cell*, and *View* for the `cds_globals` module.

4. Click *OK*.

   AMS Designer will use the `cds_globals` module you specified.

Special Notes about Design Variables and Spectre primitive parameters in the cds_globals Module when Using the AMS Designer Simulator with the SFE Parser

AMS Designer netlists your design variables as dynamic parameters in the `cds_globals` module. The AMS Designer simulator with the simulation front end (SFE) parser publishes these dynamic parameters as global parameters. If you also supply a Spectre or SPICE model file that contains a global parameter of the same name, the SFE parser encounters the global parameter twice and issues a warning message such as the following:

```
Warning from spectre in `cds_globals', during circuit read-in.
    `cds_globals': Parameter `idc' redefines parameter of same name defined at
     higher level.
```

This warning message is harmless and you can ignore it.

When you use the AMS Designer simulator with the SFE parser, the software appends a special suffix to the following Spectre primitive parameters if you have defined them in a `cds_globals` module:

Spectre Primitive Parameter	with Suffix
`temp`	`temp_mmsim_keyword_`
`tnom`	`tnom_mmsim_keyword_`
`freq`	`freq_mmsim_keyword_`
`time`	`time_mmsim_keyword_`
`scalem`	`scalem_mmsim_keyword_`
`scale`	`scale_mmsim_keyword_`
`shrink`	`shrink_mmsim_keyword_`

The software applies the suffix in cases such as the following:

```
// The following contains an out-of-module reference to a cds_globals parameter:
vsource #( .type("sine"), .delay(1n), .ampl(1), .freq(cds_globals.freq))V0 (a,b)

module cds_globals;
  dynamicparam real freq = 1M;
endmodule
```

Note: For more information about the AMS Designer simulator with the SFE parser, see "Using the Simulation Front End (SFE) Parser" in the *Virtuoso AMS Designer Simulator User Guide*.

Specifying Netlist Format for Component Instances for AMS Simulation

Cadence netlisters format instances of analog devices according to the instructions specified in the simulation information (simInfo) section of the device's component description format (CDF). The simInfo section is composed of one or more sets of directions, parameters, and terminal names, with each set representing the formatting instructions for that device for a given simulator. To support the AMS Designer simulator, the simInfo section for analog primitives contains an `ams` section. (The OSS netlister uses `spectre` simInfo and you do not need to add `ams` simInfo.)

For information about editing simulation information in a device's CDF, see "Modifying Simulation Information" in the *Component Description Format User Guide*.

Cadence provides a conversion tool that generates `ams` simInfo from the corresponding `spectre` information. For more information, see Appendix C, "Updating Legacy SimInfo for Analog Primitives."

Important

You must rewrite any Spectre-specific custom netlist procedures if you want to use them with the AMS Designer simulator. See "Using Netlisting Procedures to Customize Netlists" on page 303.

Excluding Parameters from Netlisting

You can specify parameters you want to exclude from netlisting for a library or cell by editing the AMS simulation information (simInfo) for the library or cell. See the following topics for details:

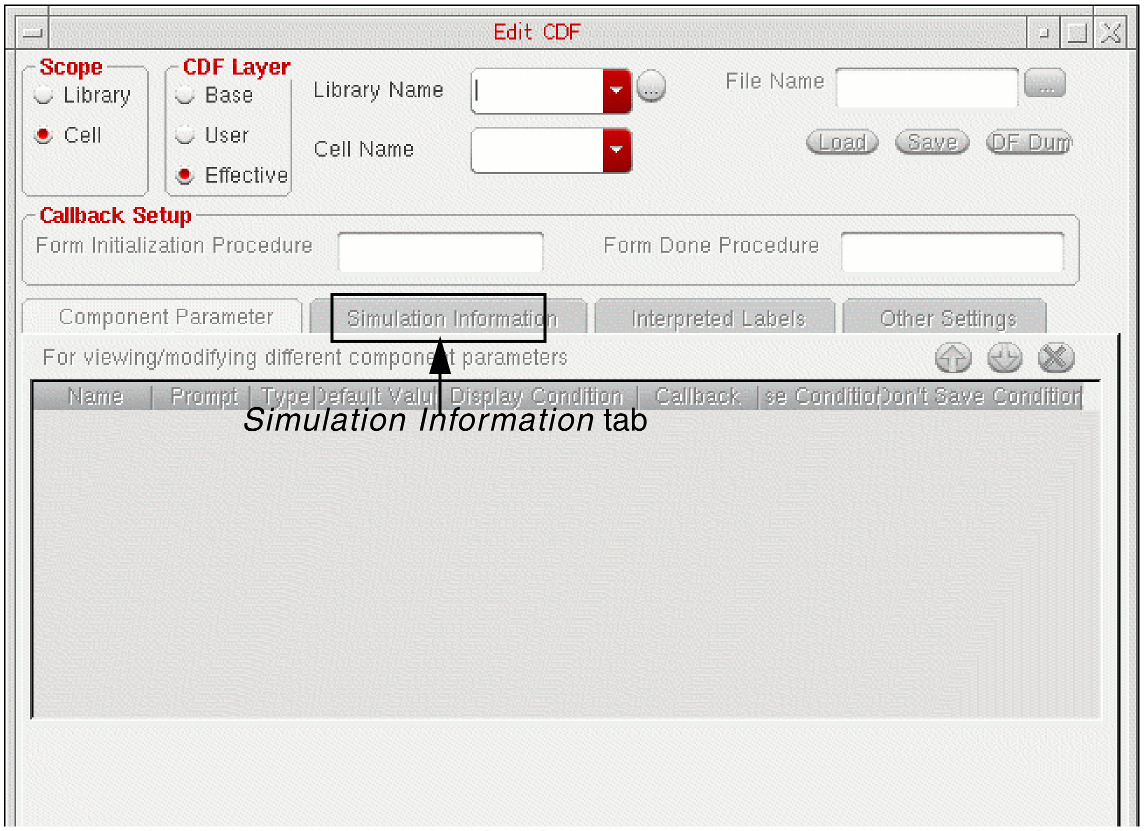
- Excluding Parameters from Netlisting for an Entire Library on page 203
- Excluding Parameters from Netlisting for a Cell on page 206

Excluding Parameters from Netlisting for an Entire Library

To specify parameters to exclude from netlisting at the library level, do the following:

1. In the command interpreter window (CIW), choose *Tools – CDF – Edit*.

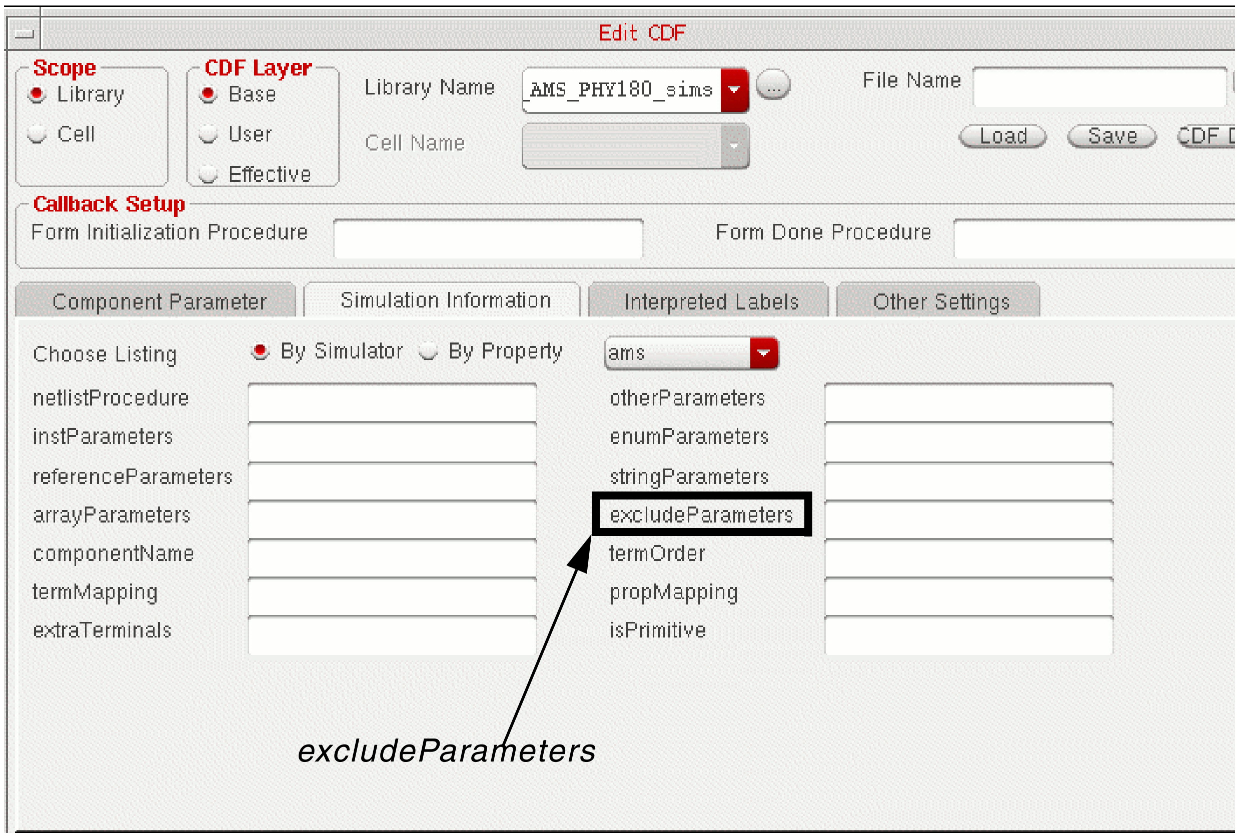
   The Edit CDF form appears.



Note: For detailed information about this form, see the *Component Description Format User Guide*.

2. In the *Scope* group box, select *Library*.
3. In the *CDF Layer* group box, select *Base*.
4. In the *Library Name* drop-down combo box, select a library whose parameters you want to exclude from netlisting.
5. Select the *Simulation Information* tab.
6. In the *Choose Simulator* drop-down combo box, select *ams*.

AMS simulation information fields appear on the form.



Scope: [✓ Library] Cell

CDF Layer: [✓ Base] User Effective

Library Name: AMS_PHY180_sims

Choose Listing: [✓ By Simulator] By Property — ams

7. In the *excludeParameters* field, type a list of one or more parameters that you want to exclude from netlisting.

8. Click *OK*.

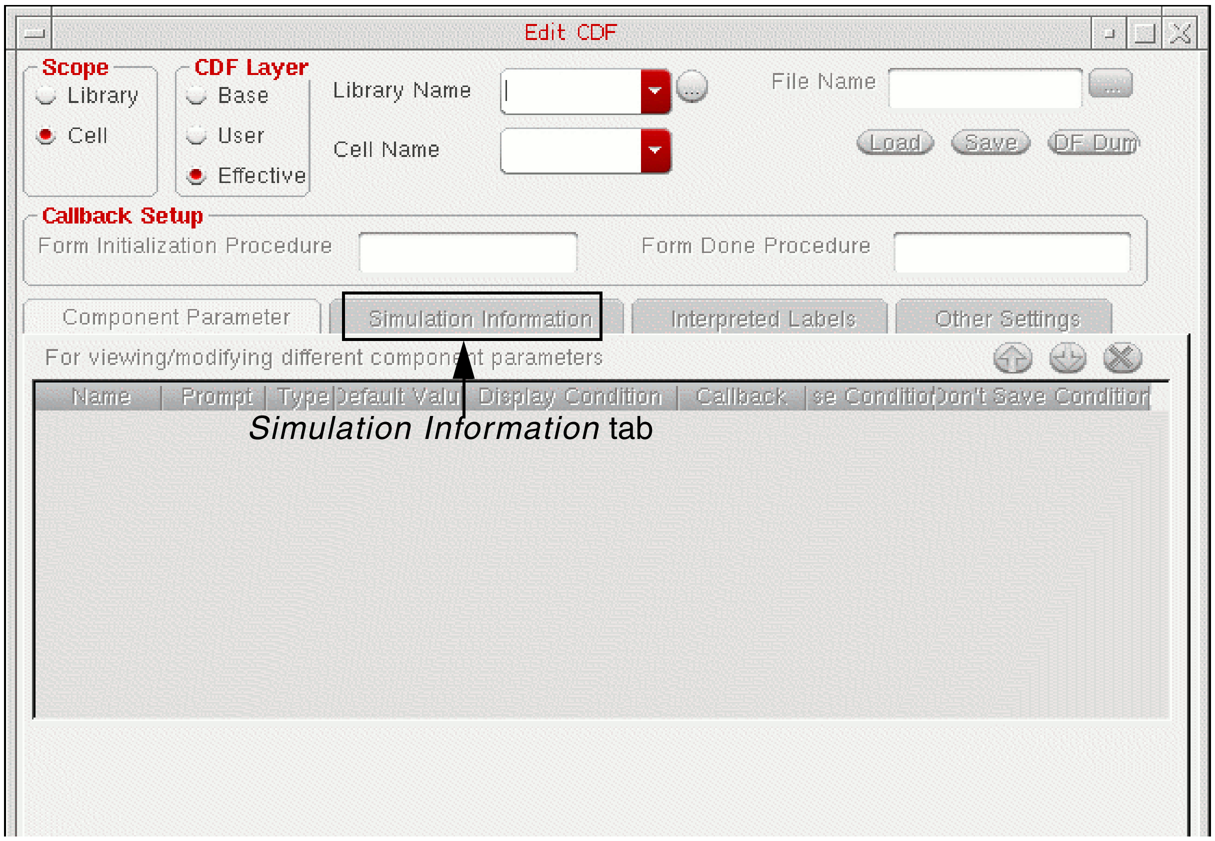
The AMS netlister ignores parameters whose names match those you typed in the *excludeParameters* field.

Excluding Parameters from Netlisting for a Cell

To specify parameters to exclude from netlisting at the cell level, do the following:

1. In the command interpreter window (CIW), choose *Tools – CDF – Edit*.

   The Edit CDF form appears.



   Note: For detailed information about this form, see the *Component Description Format User Guide*.

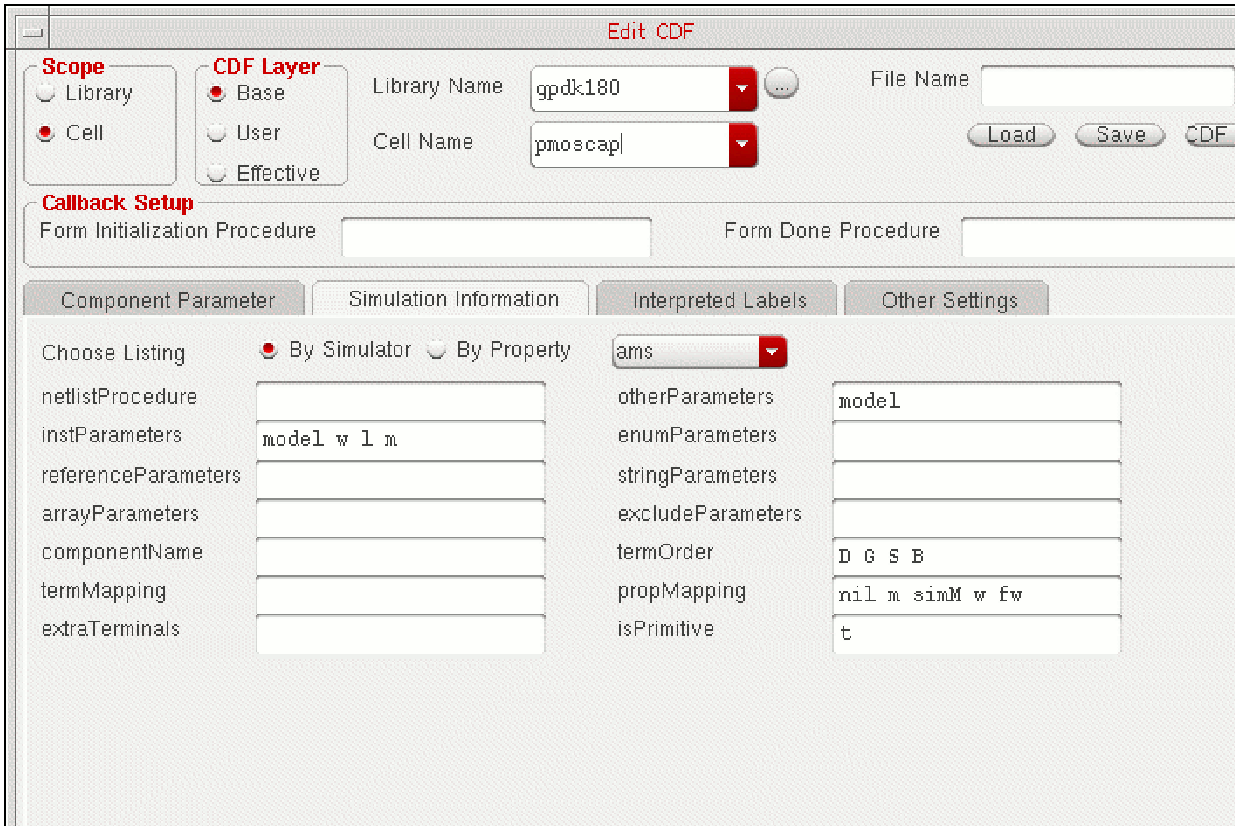
2. In the *Scope* group box, select *Cell*.
3. In the *CDF Layer* group box, select *Base*.
4. In the *Library Name* drop-down combo box, select the library that contains the cell whose parameters you want to exclude from netlisting.

   The *Cell Name* field becomes active.

5. In the *Cell Name* drop-down combo box, select the cell.
6. Select the *Simulation Information* tab.

7. In the *Choose Simulator* drop-down combo box, select *ams*.

   AMS simulation information fields appear on the form.



Scope: Library [✓ Cell]
CDF Layer: [✓ Base] User Effective
Library Name: gpdk180
Cell Name: pmoscap
File Name:
Form Initialization Procedure:
Form Done Procedure:

Choose Listing: [✓ By Simulator] By Property — ams

netlistProcedure		otherParameters	model
instParameters	model w l m	enumParameters	
referenceParameters		stringParameters	
arrayParameters		excludeParameters	
componentName		termOrder	D G S B
termMapping		propMapping	nil m simM w fw
extraTerminals		isPrimitive	t

8. In the *excludeParameters* field, type a list of one or more parameters that you want to exclude from netlisting.

9. Click *OK*.

   The AMS netlister ignores parameters whose names match those you typed in the *excludeParameters* field.

Viewing the AMS Netlister Log File

To view the AMS netlister log file, do the following:

➤ In the Virtuoso® Hierarchy Editor, choose *AMS – Results – Log Files – Netlister Log*.

The netlister log file (`runDirectory/netlist/netlister.log`) appears in a separate window. The log file contains information and messages about the netlisting process.

Tip

See also the *NCBrowse Message Browser User Guide* for information about using the NCBrowse message browser to view and analyze log files.

Understanding How the Cellview-Based Netlister Operates

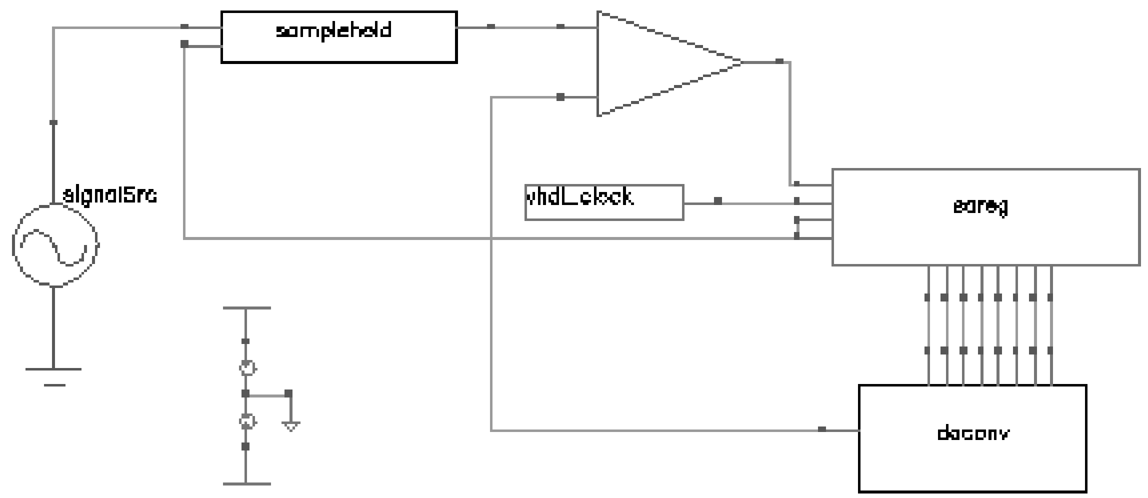
The following topics provide details about how the AMS cellview-based netlister performs certain tasks and handles certain situations:

- Passing Information to the Elaborator on page 209
- Netlisting Inherited Connections on page 210
- Netlisting Inherited Terminal Connections on page 211
- Netlisting netSet Properties on page 212
- Netlisting Aliased Signals on page 213
- Netlisting Multiplicity Factors on page 214
- Netlisting Iterated Instances on page 215
- Netlisting Model Names from Parameter Values on page 216
- Netlisting componentName Parameters on page 218
- Forcing Schematic Parameter Values to Netlist as Floating Point Values on page 219

Note: For information about OSS-based netlisting, see the *Open Simulation System Reference*. For important considerations, see “OSS-based AMS Netlister” and “Choosing the Netlister” in the *Virtuoso Analog Design Environment L User Guide*.

Passing Information to the Elaborator

AMS Designer uses Verilog-AMS attribute notation—`(* attribute *)`—to pass information to the elaborator. For example, consider the following simple schematic design where the resulting Verilog-AMS netlist contains attributes that declare library bindings for the elaborator:



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

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

module top (  );

vsource #(.type("dc"), .dc(3))
          (* integer library_binding = "analogLib";  *)
         V0  ( cds_globals.\vdd! , cds_globals.\gnd!  );
vsource #(.type("dc"), .dc(-3))
          (* integer library_binding = "analogLib";  *)
         V1  ( cds_globals.\vss! , cds_globals.\gnd!  );

vhdl_clock  (* integer library_binding = "diglib";  *) I5  ( .out1( clkSig ) );

sareg  (* integer library_binding = "diglib";  *) I3  ( .b2( b2 ),
 .endOfConv( endOfConv ), .b5( b5 ), .b6( b6 ), .b3( b3 ), .b7( b7 ),
 .b0( b0 ), .clkSig( clkSig ), .b4( b4 ), .b1( b1 ),
 .result( compOut ), .trigger( endOfConv ) );

daconv #(.refVolt(5.000000))  (* integer library_binding = "amslib"; *)
        I4  ( .b2( b2 ), .b5( b5 ), .b6( b6 ), .b3( b3 ), .compSig( dacOut ),
        .b7( b7 ), .b0( b0 ), .b4( b4 ), .b1( b1 ) );

signalSrc  (* integer library_binding = "amslib";  *) I0  ( .sig( inSig ) );

comparator  (* integer library_binding = "amslib";  *)
            I2  ( .inn( dacOut ), .inp( holdSig ), .out( compOut ) );
```

```
samplehold  (* integer library_binding = "amslib";  *)
            I1 ( .inSig( inSig ), .holdSig( holdSig ), .trigger( endOfConv ) );

endmodule
```

You can use the `view_binding` attribute similarly to specify a view binding:

```
    (* integer library_binding = "basic";
       integer view_binding = "functional"; *)
```

Note: This netlist also contains out-of-module references to objects in the `cds_globals` module.

See also

- Netlisting Inherited Connections on page 210
- Netlisting Inherited Terminal Connections on page 211
- Netlisting netSet Properties on page 212
- Netlisting Aliased Signals on page 213
- Netlisting Multiplicity Factors on page 214
- Netlisting Iterated Instances on page 215

Netlisting Inherited Connections

The AMS netlister uses Verilog-AMS attributes when translating inherited connections. The AMS elaborator uses these attributes to resolve inherited connections in the same way that other Cadence applications do.

Note: An inherited connection is a net expression associated with either a signal or a terminal. You use inherited connections to override specific global names in your design. For more information, see "Inherited Connections" in the *Virtuoso Schematic Editor L User Guide*.

The Virtuoso® Schematic Editor uses the following syntax to associate a net expression with a wire that represents a signal (for an inherited net expression):

```
[@property_name:%:default_net_name]*
```

property_name	Name of the property whose value can redefine the global signal name
default_net_name	Global signal name; must not be a nested netlist property expression (the software does not evaluate nested netlist property expressions)

The AMS netlister translates this syntax into a Verilog-AMS `wire` declaration and uses the Verilog-AMS attribute construct as follows:

```
wire (* integer inh_conn_prop_name = "property_name";
        integer inh_conn_def_value = "default_net_name"; *)
     signalName ;
```

For example, the AMS netlister translates the inherited connection net expression

```
[@xground:%:vdd!]*
```

into the following code in the Verilog-AMS netlist:

```
wire
 (* integer inh_conn_prop_name="xground";
    integer inh_conn_def_value="cds_globals.\\vdd! "; *)
\vdd! ;
```

See also the following topics:

- Netlisting Inherited Terminal Connections on page 211
- Netlisting netSet Properties on page 212

Netlisting Inherited Terminal Connections

In the Virtuoso® Schematic Editor, you create an inherited terminal connection by associating a net expression with the pin that physically represents the terminal. The pin can exist on a Schematic, layout, or schematicSymbol cellview. The AMS netlister translates inherited terminal connection expressions into port attributes that become part of the Verilog-AMS netlist. The port attributes use the same syntax as net attributes. The only difference is that the netlister attaches port attributes to port declarations rather than to net declarations.

See also the following topics:

- Netlisting Inherited Connections on page 210
- Netlisting netSet Properties on page 212

Netlisting netSet Properties

In the Virtuoso Schematic Editor, you can override the default name associated with an inherited connection by creating or modifying the appropriate `netSet` property name and value pair on the component instance that represents the affected branch of the hierarchy, thus creating an instance value for an inherited connection. (See "Adding netSet Properties to Create an Inherited Connection" in the *Virtuoso Schematic Editor L User Guide* for more information.) The AMS netlister translates `netSet` properties into `cds_net_set` attributes in the Verilog-AMS netlist using the following syntax:

```
module_identifier
     (* integer cds_net_set[0:n] = { property_nameList };
        property_list *)
     instance_identifier
     (port_list_connection);
```

module_identifier	Module identifier
n	Positive integer representing one less than the number of elements in the `cds_net_set` array that stores the names of the `netSet` properties; for example, for a two-element array, `n` is `1`
property_nameList	Comma-separated list of *n*+1 property name strings
property_list	One or more property declarations, each preceded by `integer`, where a property declaration is one of the following:

`integer` *property_name* `=` *property_value*`;`

Simple property declaration specifying the override connection such as

```
integer vdd = "cds_globals.\\3.3v! ";
```

`integer` *property_name*`[0:1] = {` *property_name*`,` *def_net_name* `};`

Inherited connection declaration consisting of a two-element array, where the first element is the new property name and the second element is the new default connection name, such as

```
integer xground[0:1] =
{"new_ground","cds_globals.\\gnd5! "};
```

instance_identifier

Instance identifier

port_list_connection

List of port identifiers and their connections

For example, the following `netSet` properties

Property	**Value**
vdd	3.3v!
xground	[@new_ground:%:gnd5!]

translate to the following `cds_net_set` attributes in the Verilog-AMS netlist:

```
comparator
 (* integer library_binding = "amslib";
    integer cds_net_set[0:1]= {"xground","vdd"};
    integer xground[0:1] = {"new_ground","cds_globals.\\gnd5! "};
    integer vdd = "cds_globals.\\3.3v! ";  *)
I2 ( .inn( dacOut ), .inp( holdSig ), .out( compOut ) );
```

Note: When the elaborator encounters a net or port inherited connection attribute, it searches the hierarchy for a `cds_net_set` attribute that lists the inherited connection property name. If it finds the property name, the elaborator connects the net or terminal to the signal name specified in the `cds_net_set` attribute as the value of the property. If the elaborator cannot find the property, it uses the default connection.

See also the following topics:

- Netlisting Inherited Connections on page 210
- Netlisting Inherited Terminal Connections on page 211

Netlisting Aliased Signals

Schematics often have nets of different names or different widths that carry the same signals. In a process called *aliasing*, the AMS netlister automatically uses instances of the `cds_alias` module to connect such nets while retaining their original names. (Retaining the original names facilitates cross-probing.) The netlister uses aliasing when connecting

- Differently named nets of a common width that all carry the same signal
- A terminal of one width to a net of the same name but of a different width

Important

> Modules that use aliased ports (as, for example, when the same name appears more than once in the list of port names) cannot instantiate other objects within themselves.

The AMS netlister instantiates the following `cds_alias` module in the `cds_globals` module or in individual cellview netlists as necessary:

```
// Verilog HDL for "basic", "cds_alias" "functional"

module cds_alias(a,a);
    parameter width = 1;
    inout [width-1:0] a;
endmodule
```

For example, the AMS netlister automatically writes the following `cds_alias` instantiation to the netlist to alias `signal2` and `signal3`:

```
cds_alias #(.width(1))
    (* integer library_binding = "basic";
       integer view_binding = "functional"; *)
    ams_alias_inst_0 (signal2, signal3);
```

See also “Passing Information to the Elaborator” on page 209.

Netlisting Multiplicity Factors

A multiplicity factor (m factor) is a value that can be inherited down a hierarchy of instances. Circuit designers use m factors to mimic parallel copies of identical devices without having to instantiate large sets of devices in parallel. The value of the inherited m factor in a particular module instance is the product of the m-factor values in the ancestors of the instance and of the m-factor value in the instance itself. If there are no passed m factors in the instance or in the ancestors of the instance, the value of the m factor is one.

To identify m factors, the AMS netlister notes the parameters required by each instance in the design and assumes that any such parameter with the name `m` is an m-factor. The m-factor parameter might be an instance property or a component parameter. To implement the m-factor capability, the netlister adds the `passed_mfactor` attribute to the corresponding instance statement in the netlist.

Note: For more information about the language attributes that AMS Designer provides to support m factors, see “Using an m factor (Multiplicity Factor)” in “Instantiating Modules and Primitives” in the *Cadence Verilog-AMS Language Reference*.

For example, you might have a `pmos4` model whose symbol has the following CDF parameters:

Field Label	**Parameter Value**
Model name	`pmos4`
Multiplier	`2`
Width	`20u M`
Length	`3u M`

The *Multiplier* field contains the m-factor value, `2`. The netlister creates a netlist that contains the following instance statement:

```
pmos4 #(.m(2), .region("triode"), .w(20u), .l(3u))
  (* integer library_binding = "amslib";
      integer cds_net_set[0:0]= {"bulk_n"};
      integer bulk_n = "cds_globals.\\vdd! ";
      integer passed_mfactor = "m"; *)
 M11 ( net92, cds_globals.„nd! , net79, cds_globals.\vdd!  );
```

The `.m(2)` on the first line passes the m-factor value to the `pmos4` model. The `passed_mfactor` attribute identifies the parameter `m` as the m factor. Each instance statement for the `pmos4` model contains a similar attribute.

Netlisting Iterated Instances

An iterated instance is a single instance that represents multiple logical copies. An iterated instance has a name of the form `I<3:1>`, which indicates the number of logical copies. Although the logical copies are not explicitly present in the design, they function as design elements with implicit connections to other elements of the design. The AMS netlister expands each iterated instance into a number of instances and uses the `elaboration_binding` attribute to pass information to the elaborator.

For example, perhaps you have a top-level schematic called `iter_top` that contains an instance of a cell called `iter_master` with an instance name of `I<3:1>`. This instance name indicates an iterated instance that represents three logical instances. The AMS netlister expands the iterated instance to generate the following netlist:

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

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

module iter_top (  );       // iter_top is the top-level schematic
```

```
     iter_master           // iter_master is the cell being instantiated
       (* integer library_binding = "amslib";
          integer elaboration_binding = "I[3:1]"; *)
        I_3 ( .a( cds_globals.\gnd!  ) ), // Iterated instances
        I_2 ( .a( cds_globals.\gnd!  ) ), // in their expanded form
        I_1 ( .a( cds_globals.\gnd!  ) );
endmodule
```

See also “Passing Information to the Elaborator” on page 209.

Netlisting Model Names from Parameter Values

You can use `modelname`, `model`, or `modelName` parameters (referred to here as `model*` parameters) to pass a model name through the design hierarchy. A `model*` parameter must be an AEL expression that has `parseAsCEL` set to `t` and `parseAsNumber` set to `nil`. The AMS netlister handles a `model*` parameter as follows:

If the value of the parameter is a...	Then the AMS netlister...
Literal string, for example, `model="modStr"`	Evaluates it as a literal (`modStr`)
pPar expression, for example, `model=pPar("modStr")`	Evaluates it to be the parameter (`modStr`); writes a parameter declaration (`parameter modStr=`*`defVal`*`;`) and instantiates an `analogmodel` construct
iPar expression that resolves to a literal, for example, `model=iPar("modStr"), modStr="myModel"`	Replaces the name of the cell on the instantiation to which the `modelname` parameter applies

Important

If the model name evaluates to a name that is not legal in the Verilog-AMS language (such as `4nmos`, because Verilog-AMS identifiers cannot begin with a number), the AMS netlister prepends the name with a backslash (for example, `\4nmos`) to *escape* the identifier and make it legal in the Verilog-AMS language.

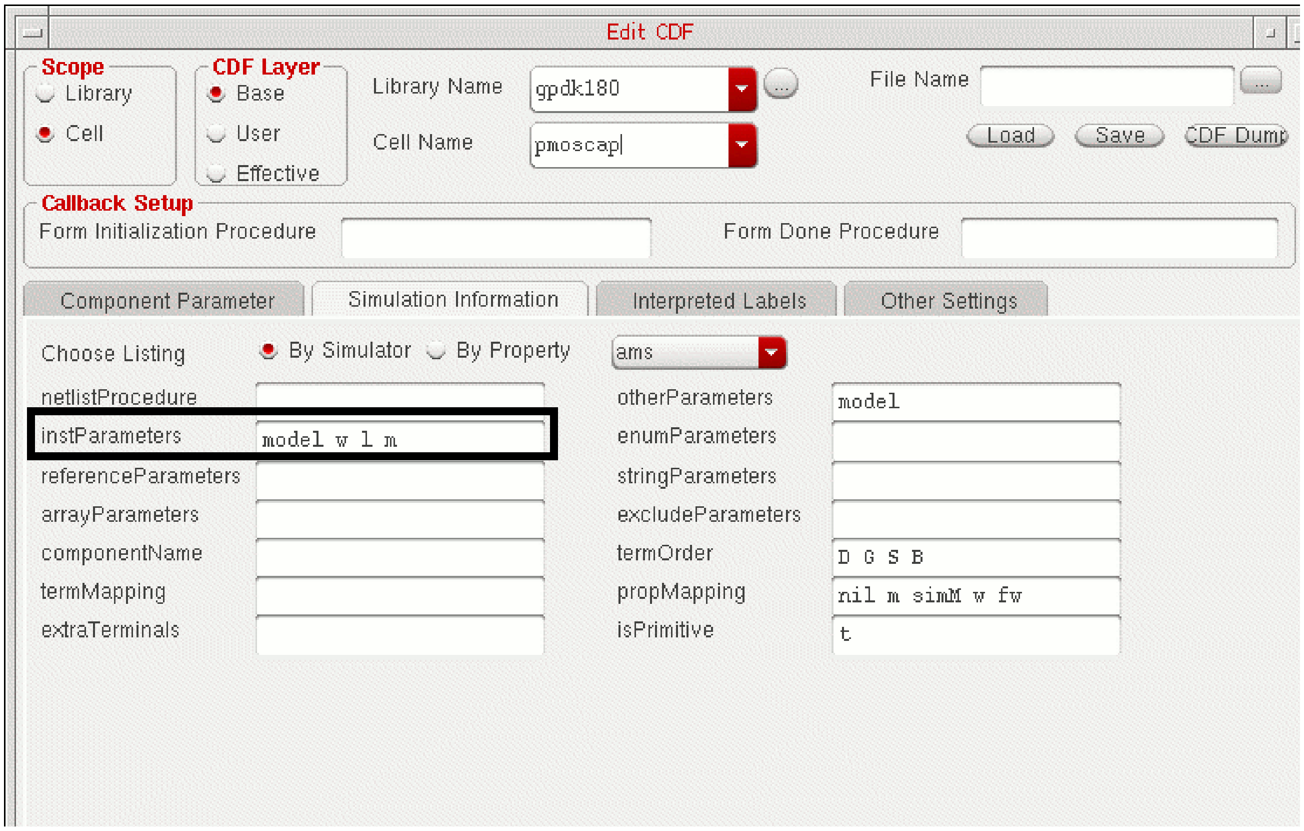
The examples below illustrate these scenarios. These examples use the `modelname` parameter to pass model names, but they could also use `model` or `modelName`.

- Example 1 — modelname is a Literal String on page 217
- Example 2 — modelname is a pPar Expression on page 218

- Example 3 — modelname is an iPar Expression on page 218

Note: If an instance has more than one `model*` or *componentName* parameter, the `modelname` parameter has the highest precedence, followed by `model`, `modelName`, and, finally, *componentName*.

Note: If the device is a primitive that supports model passing semantics, you must list the `model`, `modelname`, or `modelName` parameter in the *instParameters* field on the Edit CDF form.



Scope: Library [✓ Cell]
CDF Layer: [✓ Base] User Effective
Library Name: gpdk180
Cell Name: pmoscap
File Name:
Form Initialization Procedure:
Form Done Procedure:
Choose Listing: [✓ By Simulator] By Property — ams

netlistProcedure		otherParameters	model
instParameters	model w l m	enumParameters	
referenceParameters		stringParameters	
arrayParameters		excludeParameters	
componentName		termOrder	D G S B
termMapping		propMapping	nil m simM w fw
extraTerminals		isPrimitive	t

Example 1 — modelname is a Literal String

The model definition for instance `i0` of a `pmos` device is `pmos395`. The model name is a literal string: `modelname="pmos395"`. The AMS netlister writes the following Verilog-AMS instantiation from this information:

```
pmos395 #(.l(3u), .w(9.5u)) (*integer library_binding = "analogLib";*)
        i0 ( .D( vref3 ), .G( vref1 ), .S( cds_globals.\vdd!  ) );
```

The syntax for the Verilog-AMS instance in this case is

```
model_name #(prop_list) (* attr_list *) inst_name (port_connections);
```

Example 2 — modelname is a pPar Expression

The model definition for instance `i0` of a `pmos` device is `pmos395`. The `modelname` parameter is a pPar expression: `modelname=pPar("mod1")`. The parameter in the pPar expression (`mod1`) is a CDF parameter for the `pmos` instance whose default value is `"pmos395"`. The AMS netlister writes the following Verilog-AMS parameter declaration (for `mod1`) and `analogmodel` instantiation:

```
parameter mod1="pmos395";
...
analogmodel #(.l(3u), .w(9.5u)) , .modelname(mod1) )
        (*integer library_binding = "analogLib";*)
        i0 ( .D( vref3 ), .G( vref1 ), .S( cds_globals.\vdd!  ) );
```

The Verilog-AMS syntax in this case is

`analogmodel #(`_`prop_list`_**`) (*`** _`attr_list`_ **`*)`** _`inst_name`_ **`(`**_`port_connections`_**`);`**

Example 3 — modelname is an iPar Expression

The model definition for instance `i0` of a `pmos` device is `pmos395`. The `modelname` parameter is an iPar expression: `modelname=iPar("modelString")`. The iPar expression evaluates to the model name as follows:

> `modelname=iPar("modelString")` and `modelString="pmos395"`
> so `modelname="pmos395"`

The AMS netlister finds the `modelString` parameter and replaces the iPar expression with the value of that parameter and writes the following Verilog-AMS instantiation:

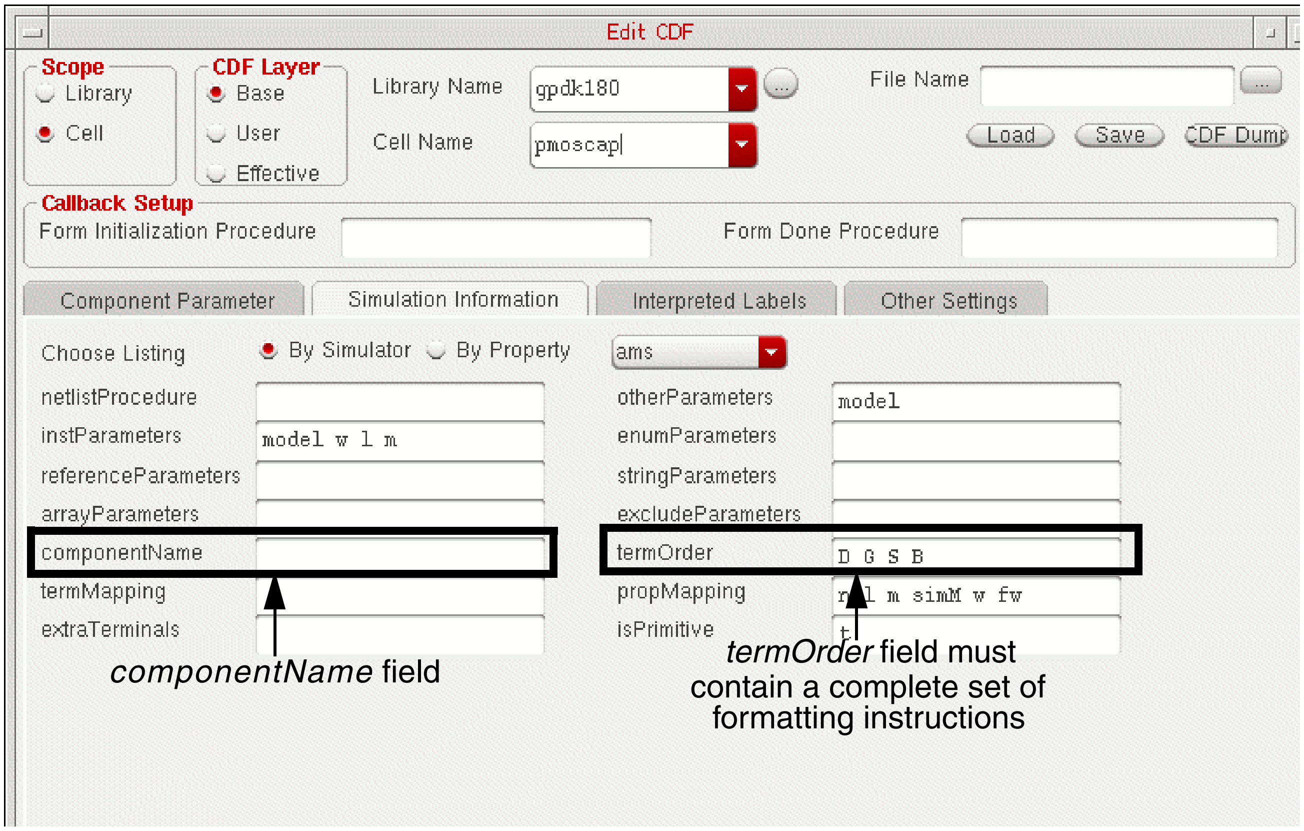
```
pmos395 #(.l(3u), .w(9.5u)) (*integer library_binding = "analogLib";*)
        i0 ( .D( vref3 ), .G( vref1 ), .S( cds_globals.\vdd!  ) );
```

which is the same result as when `modelname="pmos395"` directly (see "Example 1 — modelname is a Literal String" on page 217).

Netlisting componentName Parameters

You can use the *componentName* field on the *Simulation Information* tab to specify the name of a SPICE or Spectre primitive that has the same name as a Verilog-AMS built-in primitive. (You must provide a complete set of formatting instructions in the *termOrder* field.) The AMS netlister uses the value you type in this field verbatim (that is, it does not employ any name mapping). The value in the *componentName* field must be a literal string (unlike `model*` parameters which can also be pPar or iPar expressions—see "Netlisting Model Names from Parameter Values" on page 216 for more information). The AMS netlister always processes the value in the *componentName* field as in "Example 1 — modelname is a Literal String" on page 217.

Note: If the value of a *componentName* parameter is an AEL expression, the AMS netlister produces an error message and does not generate a netlist.



Library Name	gpdk180	File Name	
Cell Name	pmoscap		
Form Initialization Procedure		Form Done Procedure	
Choose Listing	[✓ By Simulator] By Property	ams	
netlistProcedure		otherParameters	model
instParameters	model w l m	enumParameters	
referenceParameters		stringParameters	
arrayParameters		excludeParameters	
componentName		termOrder	D G S B
termMapping		propMapping	nil m simM w fw
extraTerminals		isPrimitive	t

Note: If an instance has more than one `model*` or *componentName* parameter, the `modelname` parameter has the highest precedence, followed by `model`, `modelName`, and `componentName`.

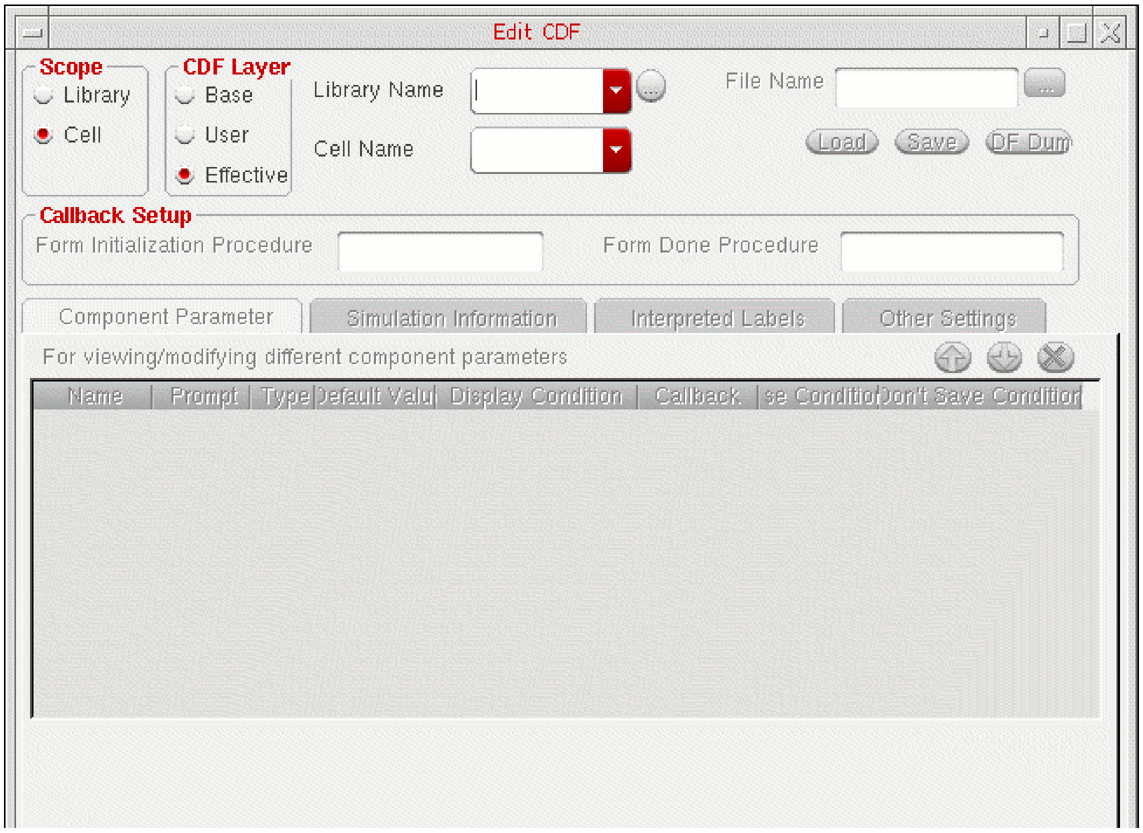
Forcing Schematic Parameter Values to Netlist as Floating Point Values

For cases where a whole number floating point parameter value appears as an integer in the netlist, you can edit the CDF for that cell's parameter to force floating point values. You can override the default value that you specify for a parameter (a floating point value) by editing the object properties of specific cell instances on the schematic.

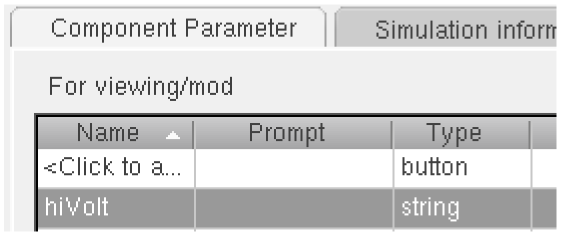
To specify a parameter for a cell so that the AMS netlister writes its value as a floating point value to the netlist, do the following:

1. In the CIW, choose *Tools – CDF – Edit*.

The Edit CDF form appears.

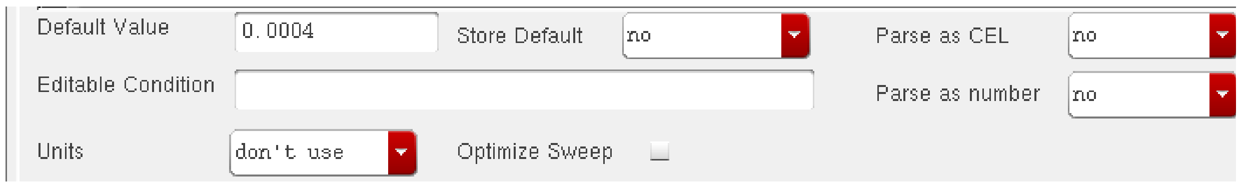


2. In the *Scope* group box, select *Cell*.
3. In the *CDF Layer* group box, select *Base*.
4. Use the drop-down combo boxes in the *Library Name* and *Cell Name* fields to specify the name of the component that has the parameter of interest.
5. (Optional) If the parameter does not already exist, you can click in the *Name* column on the *Component Parameters* tab and type a parameter name. For example, `hiVolt`.
6. In the *Type* column, select *string*.



Name	Prompt	Type
<Click to a...		button
hiVolt		string

Options related to string parameters appear at the bottom of the Edit CDF form:



Default Value	0.0004	Store Default	no	Parse as CEL	no
Editable Condition				Parse as number	no
Units	don't use	Optimize Sweep	☐		

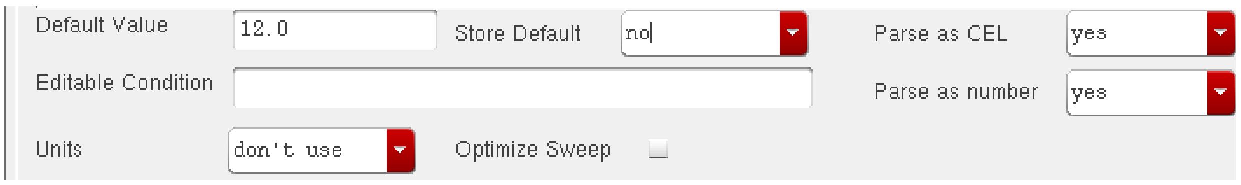
7. In the *Parse as CEL* field, select *yes*.

   This setting specifies that you want the program to process this string parameter value as a CDF Expression Language (CEL) expression. CEL is another name for the analog expression language (AEL). See the *Analog Expression Language Reference* for more information.

8. In the *Parse as number* field, select *yes*.

   For string parameters that contain numeric data, this setting tells the program to evaluate this string parameter value as a floating-point number.

9. In the *Default Value* field, type a default value for the parameter using the number format you want the netlister to use. For example, type `12.0` instead of `12` to force the AMS netlister to write the parameter value as a floating point number to the netlist.



Default Value	12.0	Store Default	no	Parse as CEL	yes
Editable Condition				Parse as number	yes
Units	don't use	Optimize Sweep	☐		

Important

For Verilog-AMS and VHDL-AMS (which is a strongly-typed language), you need to type a whole number as a real number (using a decimal point followed by a zero) to force the type for these text cellviews.

10. Click *OK*.

    The AMS netlister writes the cell instance statement to include specific declarations for parameters using a non-default value. For example, if you change the default value of the `hiVolt` parameter to `10.5` by editing the object properties for the cell instance on the schematic, the AMS netlister writes a cell instance statement something like this:

```
myCell #(.hiVolt(10.5))  (* integer library_binding = "myLib";  *)
I4 ( pinConnections );
```

9 Working with Schematic Designs

One of the AMS Designer flows allows you to create a Verilog®-AMS netlist for each cellview you save using the Virtuoso® Schematic Editor. If you are familiar with the schematic editor, you already know most of what is required to take advantage of the AMS Designer capabilities. See the following topics for more information:

Note: For detailed information about the schematic editor, see the *Virtuoso Schematic Editor L User Guide*. For specific information about inherited connections, see "Interited Connections".

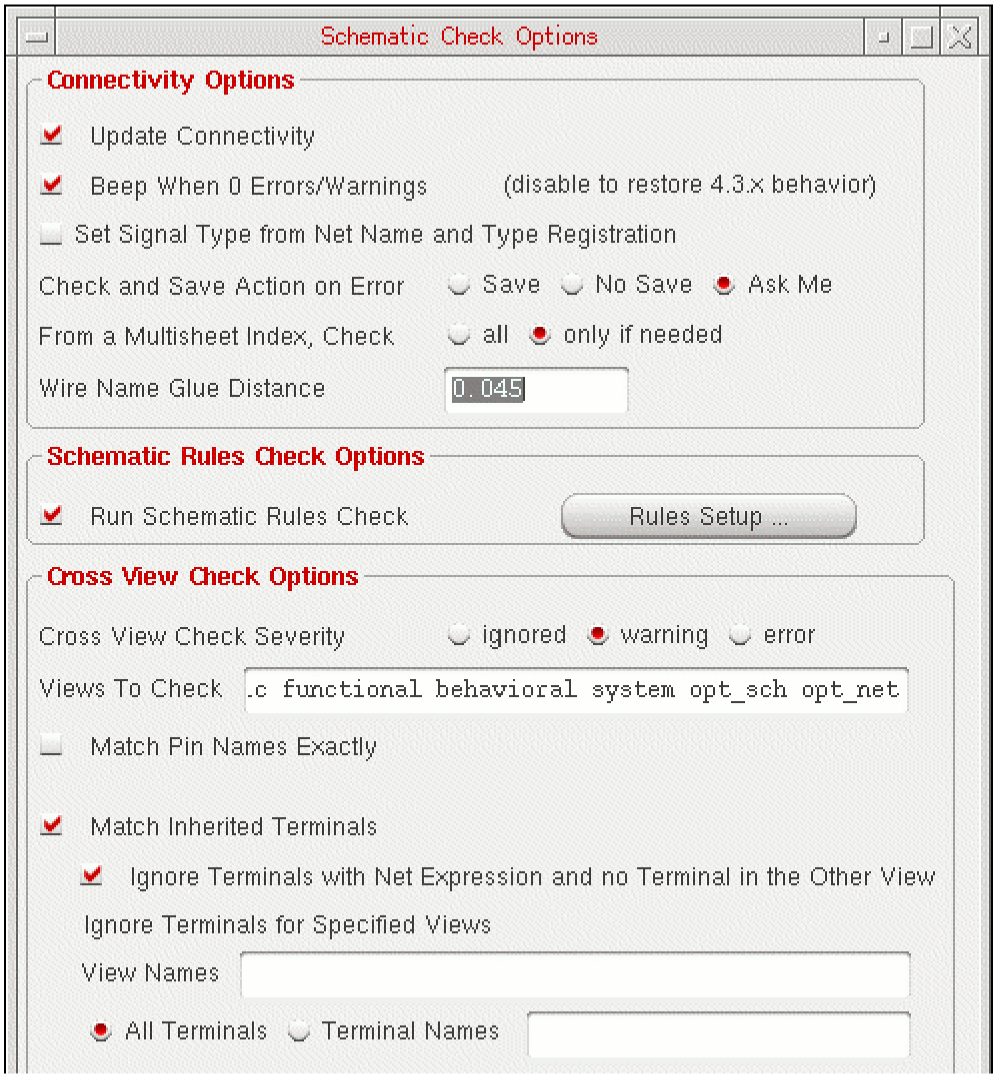
Specifying Schematic Rules Checking for AMS Designer

You can specify the checking that occurs whenever you check and save a schematic. For AMS Designer, these checks can give you feedback regarding Verilog-AMS translation without your having to generate a netlist.

To specify and turn on checking rules, do the following:

1. In the Virtuoso® Schematic Editor window, choose *Options – Check*.

   The Schematic Check Options form appears.



Schematic Check Options

Connectivity Options

✓ Update Connectivity
✓ Beep When 0 Errors/Warnings (disable to restore 4.3.x behavior)
☐ Set Signal Type from Net Name and Type Registration
Check and Save Action on Error: Save / No Save / [✓ Ask Me]
From a Multisheet Index, Check: all / [✓ only if needed]
Wire Name Glue Distance: 0.045

Schematic Rules Check Options

✓ Run Schematic Rules Check — Rules Setup ...

Cross View Check Options

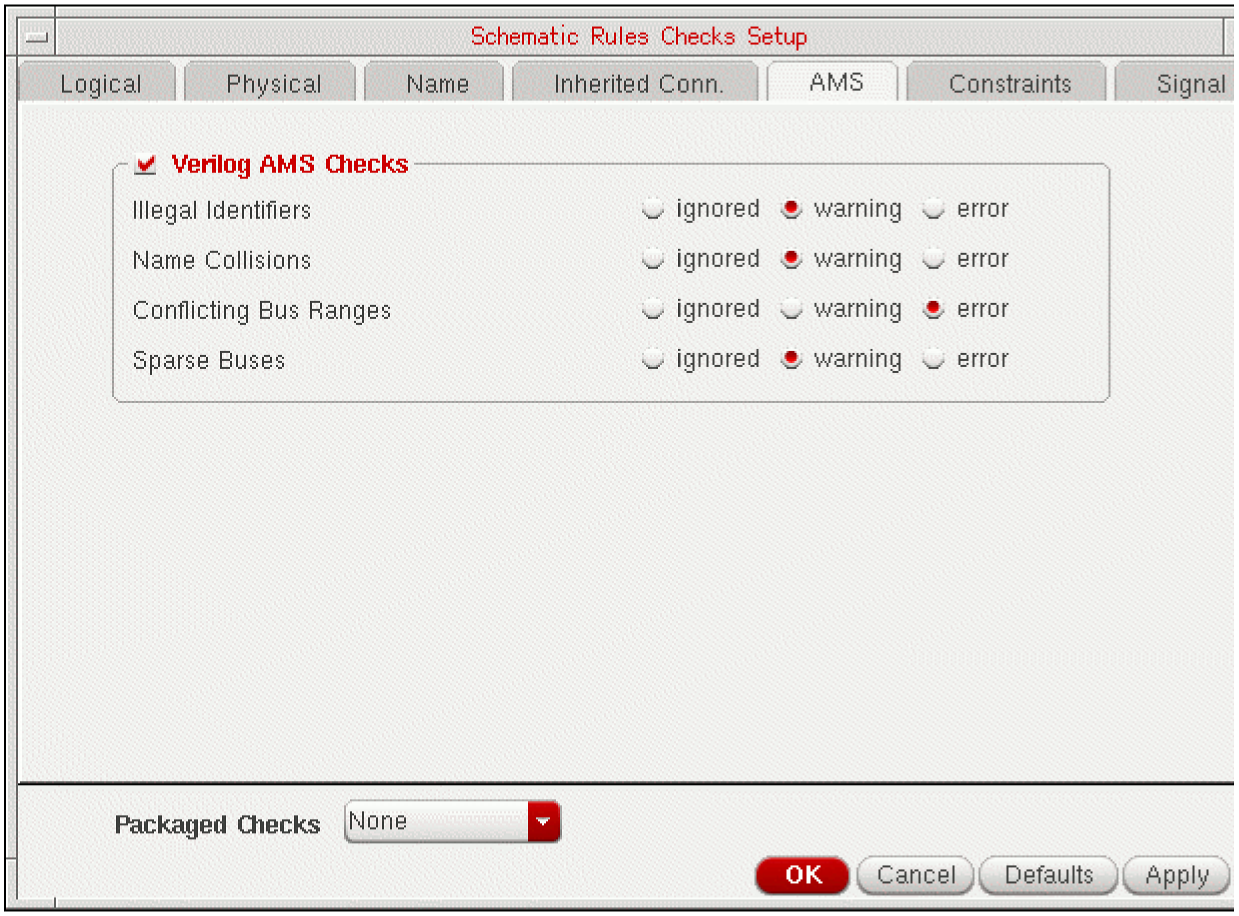
Cross View Check Severity: ignored / [✓ warning] / error
Views To Check: .c functional behavioral system opt_sch opt_net
☐ Match Pin Names Exactly
✓ Match Inherited Terminals
✓ Ignore Terminals with Net Expression and no Terminal in the Other View
Ignore Terminals for Specified Views
View Names:
[✓ All Terminals] / Terminal Names:

2. Click *Rules Setup*.

The Setup Schematic Rules Checks form appears.

3. Select the *AMS* tab.
4. Select *yes* for *Run AMS Checks*.

   The *Verilog-AMS Checks* group box becomes active.


Schematic Rules Checks Setup

Logical | Physical | Name | Inherited Conn. | AMS | Constraints | Signal

✓ Verilog AMS Checks

Illegal Identifiers	ignored	[✓ warning]	error
Name Collisions	ignored	[✓ warning]	error
Conflicting Bus Ranges	ignored	warning	[✓ error]
Sparse Buses	ignored	[✓ warning]	error

Packaged Checks: None

5. (Optional) Change the severity levels for one or more of these checks:

Verilog-AMS Check	Selection	Description
Illegal Identifiers	*ignored*	Maps noncompliant identifiers to names that are legal in the target language
	warning	Maps noncompliant identifiers to names that are legal in the target language and issues a warning telling you how the netlister mapped the name
	error	Netlisting halts immediately when the AMS netlister encounters a noncompliant identifier
Name Collisions	*ignored*	Maps names that are not unique to system-generated names that are legal in the target language without issuing a warning
	warning	Maps names that are not unique to system-generated names that are legal in the target language and issues a warning
	error	Netlisting halts immediately when the AMS netlister encounters a name that is not unique
Conflicting Bus Ranges	*ignored*	Netlisting continues when the AMS netlister encounters conflicting bus ranges, without issuing a warning, if it is possible to create a valid netlist
	warning	Netlisting continues when the AMS netlister encounters conflicting bus ranges if it is possible to create a valid netlist; the netlister indicates how it transforms noncompliant bus data **Note:** The generated netlist is likely to be less readable than one created from compliant bus data.
	error	Netlisting halts immediately when the AMS netlister encounters conflicting bus ranges

Verilog-AMS Check	Selection	Description
Sparse Buses	*ignored*	Overdeclares any sparse buses without issuing a warning
	warning	Overdeclares any sparse buses and issues a warning
	error	Netlisting halts immediately when the AMS netlister encounters a sparse bus

See also "Verilog-AMS Compatibility Exceptions" on page 625.

6. Click *OK*.

Whenever you choose any of the following menu commands to check and save your schematic, the program performs the checks you specified:

- *File – Check and Save*
- *Check – Current Cellview*
- *Check – Hierarchy*

Language Noncompliance

Identifiers are noncompliant if one or more of the following situations applies:

- Identifiers do not follow the syntax required by the netlist language you plan to use
- Identifiers are reserved words in the netlist language

  For a list of Verilog-AMS reserved words, see the "Verilog-AMS Keywords" appendix in the *Cadence Verilog-AMS Language Reference*.
- Identifiers do not map cleanly to the netlist language
- Identifiers are not unique within the design

Because the determination of noncompliance depends on the target netlist language, it is possible to have identifiers that are compliant for one target language and noncompliant for another. To ensure that identifiers are compliant for every target netlist language, use the following syntax.

```
basic_identifier ::=
      letter {[_] letter_or_digit}

letter_or_digit ::=
      a-z | 0-9
```

For example, the following identifiers are compliant for every target language:

```
an_identifier_name
a_2nd_name
a_name2
```

The following identifiers might be noncompliant for some target languages because they do not use the suggested syntax:

Noncompliant Identifier	Reason Why It Might Be Noncompliant
`2identifier`	Should begin with a letter
`My_identifer`	Should not use uppercase letters
`an_identifier_`	Should end with a letter or digit
`a&b`	Should not use characters other than a-z, 0-9, and underscore

System-Generated Names

To comply with AMS Designer environment guidelines, each instance, cell, terminal, parameter, and net in your design must have a unique name. If the names of these components are not unique, the AMS netlister acts as shown in the table below.

How Verilog-AMS Handles Non-Unique Identifiers

Objects sharing a name	AMS netlister action
module terminal, cell	No mapping occurs; netlisting proceeds normally
parameter, module terminal	Netlisting fails
instance terminal, parameter of the same instance	No mapping occurs, and a warning is issued
parameter, cell	No mapping occurs, and netlisting proceeds normally
net, parameter	Net identifier maps to `*netName*_netclash`
net, module terminal	Net identifier maps to `*netName*_netclash`
	Note: No mapping occurs when the net and module terminal are connected to each other.
net, cell	Net identifier maps to `*netName*_netclash`
instance, net	Instance identifier maps to `*instName*_instclash`

How Verilog-AMS Handles Non-Unique Identifiers, *continued*

Objects sharing a name	AMS netlister action
instance, parameter	Instance identifier maps to *`instName`*`_instclash`
instance, module terminal	Instance identifier maps to *`instName`*`_instclash`
instance, cell	Instance identifier maps to *`instName`*`_instclash`

Bus Range Conflicts

Bus ranges conflict when the indexes sometimes go from smaller to larger and other times go from larger to smaller in references to the same bus. For example:

```
a<0:7>
a<7:6>
a<5:0>
a<2:4>
```

The same example in Verilog-AMS is as follows:

```
a[0:7]
{a[7],a[6]}
{a[5],a[4],a[3],a[2],a[1],a[0]}
a{2:4}
```

Sparse Buses

Sparse buses do not comply with AMS Designer environment guidelines because you must declare buses as a contiguous vector of bits before using them in Verilog-AMS.

Here is an example of a sparse bus:

```
b<5:0:2>
```

which is the same as

```
b<5>, b<3>, b<1>
```

If you specify *warning* or *ignore* for the *Sparse Buses* check, the AMS netlister overdeclares sparse buses so it can continue netlisting. For example:

```
module XXX (.b({b[5],,b{3],,b[1]}), ...);
    input [5:1] b;
    ...
```

Creating Cellviews Using the AMS Designer Environment

You can create a symbol or a block to represent a Verilog-AMS or VHDL-AMS module so that you can place it on a schematic. You can create a new HDL cellview (Verilog-AMS or VHDL-AMS) and then create a symbol cellview to go with it. Whenever you create a Verilog-AMS or VHDL-AMS cellview, the environment opens a text editing window containing skeleton source code for the supported language type you select. You can write the source code for your HDL design unit and save it as a cellview.

Note: Before you create a cellview, you must have a library in which to place it. You can create and store Verilog-AMS and VHDL-AMS components in any Cadence component library. You can create a new library or use one that already exists.

See the following topics for more information:

- Creating a Schematic Symbol View for a Verilog-AMS Module on page 232
- Creating a Block to Represent a Verilog-AMS Module on page 234
- Creating a Verilog-AMS Cellview from an Existing Symbol or Block on page 235
- Creating a VHDL-AMS Cellview from an Existing Symbol or Block on page 237
- Creating HDL Source Files Outside the AMS Designer Environment on page 240
- Creating a New Verilog-AMS Module Cellview on page 242
- Creating a New VHDL-AMS Module Cellview on page 243
- Creating a Symbol Cellview from a Verilog-AMS Cellview on page 245

See also “Viewing Source Code for an HDL Cellview” on page 247.

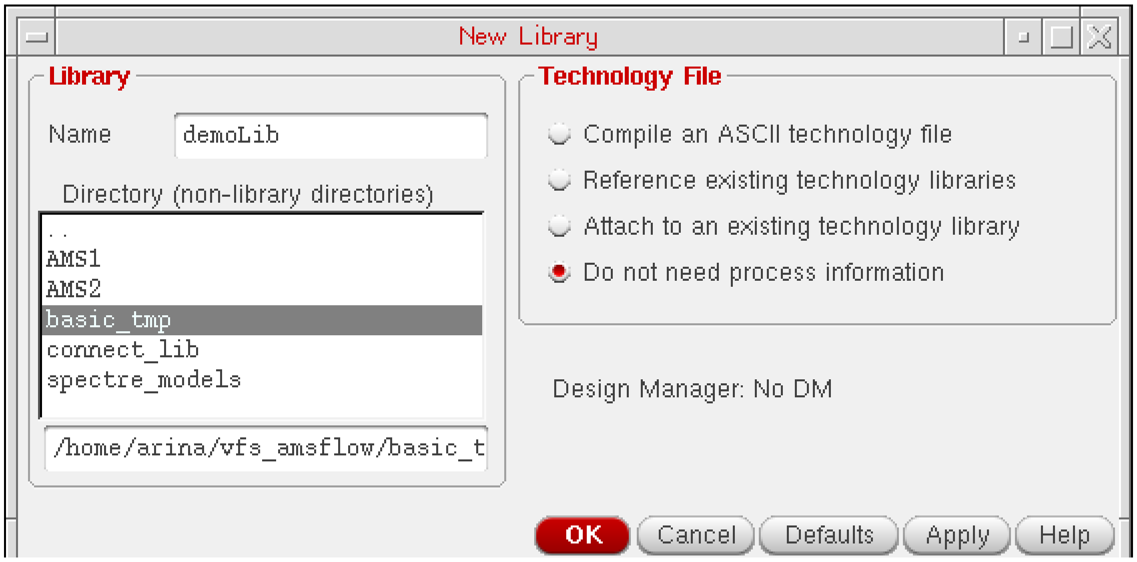
Creating a New Library

To create a new library, do the following:

1. From the command interpreter window (CIW), choose *File – New – Library*.

   The New Library form appears.

2. In the *Name* field, type the new library name.



Name: demoLib

Directory (non-library directories)

```
..
AMS1
AMS2
basic_tmp
connect_lib
spectre_models
```

/home/arina/vfs_amsflow/basic_t

Compile an ASCII technology file
Reference existing technology libraries
Attach to an existing technology library
[✓ Do not need process information]

Design Manager: No DM

3. In the *Directory* area, select a directory.

4. Turn on *Do not need process information*.

5. Click *OK*.

   A "*Created library...*" message appears in the CIW.

   For additional information about creating new libraries, see "Creating a Library in Library Manager" in the *Cadence Library Manager User Guide*.

Creating a Schematic Symbol View for a Verilog-AMS Module

To include a Verilog-AMS module on a schematic, you must create a symbol to represent the function described by the module. Use one of the following methods to create a symbol for your Verilog-AMS module:

- Creating a New Symbol Cellview from the Command Interpreter Window on page 232
- Copying an Existing Symbol on page 233
- Creating a New Symbol Cellview from a Pin List or from another Cellview on page 233

Place your new symbol in any existing library.

Important

The direction you assign to a symbol pin (Verilog-AMS defines pin direction) does not affect that terminal in the Verilog-AMS module. However, if you have multiple cellviews for a component, make sure that the name (which can be mapped), type, and location of pins you assign in a symbol cellview match what is specified in the other cellviews.

See also "Creating a Symbol Cellview from a Verilog-AMS Cellview" on page 245.

Creating a New Symbol Cellview from the Command Interpreter Window

To create a new symbol cellview from the command interpreter window (CIW), do the following:

1. In the CIW, choose *File – New – Cellview*.

   The New File form appears.

   See "Creating a New Cellview" in the *Cadence Library Manager User Guide* for detailed information about using this form.

2. In the *Type* field, be sure to select *schematicSymbol*.

   symbol appears in the *View* field.

3. When you are done specifying your symbol cellview, click *OK*.

   The Symbol Editor window appears.

Copying an Existing Symbol

To copy an existing symbol, do the following:

1. In the Library Manager, select the cellview you want to copy.
2. Choose *Edit – Copy*.

   The Copy View form appears.

   See "Copying a View" in the *Cadence Library Manager User Guide* for detailed information about using this form.

Tip

Look in `analogLib` for example cellviews to copy.

Creating a New Symbol Cellview from a Pin List or from another Cellview

Important

Before you begin, you must first have an existing cellview with defined input and output pins.

To create a new symbol cellview from another cellview, do the following:

1. In the schematic editor, select the cellview you want to copy.
2. Choose *Create – Cellview – From Cellview*.

   The Cellview from Cellview form appears.

   See "Automatically Creating a Cellview from another Cellview" in the *Virtuoso Schematic Editor L User Guide* for detailed information about using this form.

To create a new symbol cellview from the pin list of another cellview, do the following:

1. In the schematic editor, select the cellview whose pin list you want to use.
2. Choose *Create – Cellview – From Pin List*.

   The Cellview from Pin List form appears.

   See "Automatically Creating a Cellview from a Pin List" in the *Virtuoso Schematic Editor L User Guide* for detailed information about using this form.

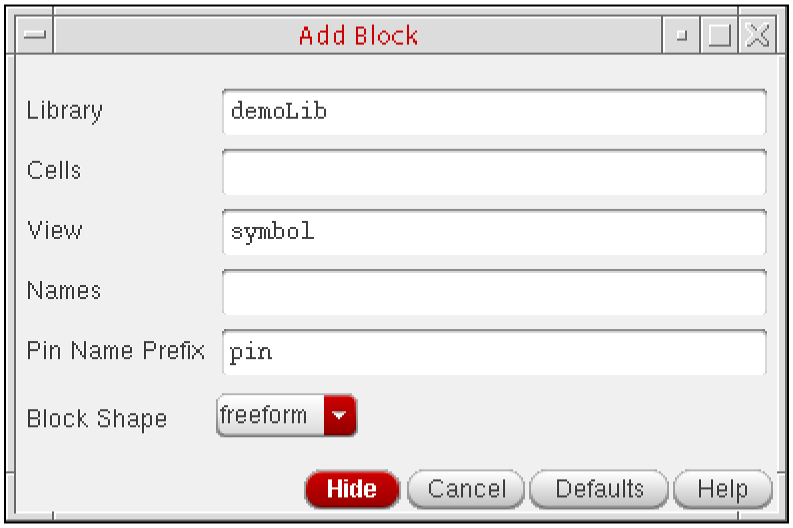
Creating a Block to Represent a Verilog-AMS Module

In top-down design practice, you can use blocks to represent Verilog-AMS components. You can create blocks at any level in your design, even before you know how the individual component symbols should look. The schematic editor automatically creates a symbol view for the block.

To create a block for a Verilog-AMS module and wire it up, do the following:

1. In the schematic editor, choose *Create – Block*.

   The Add Block form appears.

Library	demoLib
Cells	
View	symbol
Names	
Pin Name Prefix	pin
Block Shape	freeform

   See “Adding Blocks” in the *Virtuoso Schematic Editor L User Guide* for detailed information about this form.

2. In the *Library*, *Cells*, and *View* fields, type a library name, cell name, and view name.

   The default library name is the current library and the default view name is *symbol*.

   Specify a cell and view combination that does not already exist in that library. You can have schematic (*schematic*), Verilog-A (*VerilogA*), or Verilog-AMS (*VerilogAMSText*) views for that cell, but you cannot already have a symbol (*symbol*) view.

3. (Optional) In the *Names* field, type one or more instance names.

4. (Optional) In the *Pin Name Prefix* field, specify the pin name seed to use when you connect a wire to the block.

   If you specify a seed of `pin`, the schematic editor names the first pin that you add `pin1`, names the second pin `pin2`, and so on.

5. Using the *Block Shape* drop-down combo box, select a block shape.
6. Place the block:
   - ❑ For *freeform* shape, click where you want to place the first corner of the rectangle and drag to the opposite corner. Release the mouse button to complete the block.
   - ❑ For any other shape, drag the predefined block shape to the location where you want to place it and click.

     See the *Virtuoso Schematic Editor L User Guide* for details about modifying the block shapes using the `schBlockTemplate` variable in the `schConfig.il` file.

   As you place each block, the schematic editor labels it with an instance name.

   See also the following topics in the *Virtuoso Schematic Editor L User Guide*:

   - ❑ Adding Narrow or Wide Wires
   - ❑ Adding Wires and Pins to Blocks

Creating a Verilog-AMS Cellview from an Existing Symbol or Block

To create a Verilog-AMS cellview from an existing symbol or block, do the following:

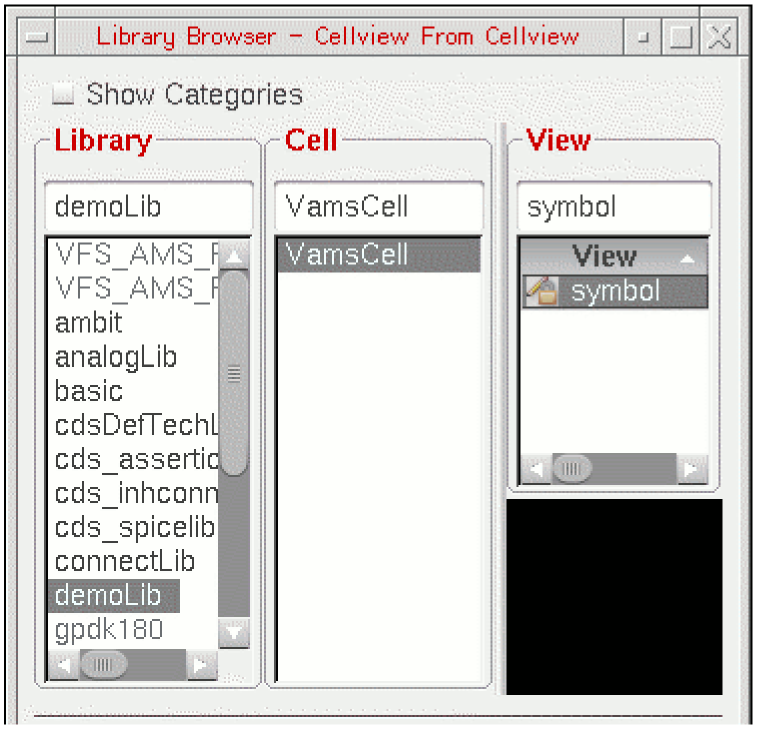
1. In the schematic editor, choose *Create – Cellview – From Cellview*.

   The Cellview from Cellview form appears.

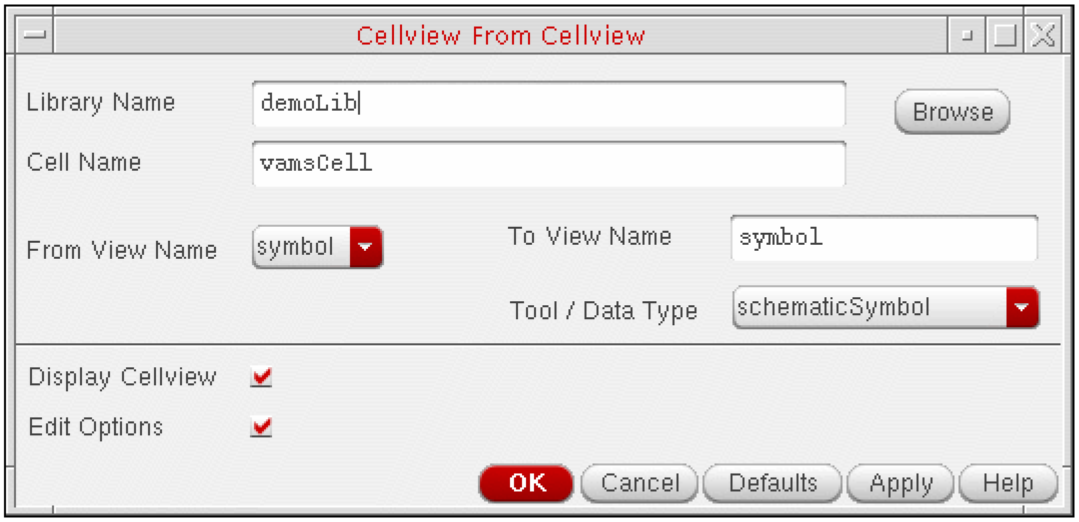
   See "Automatically Creating a Cellview from another Cellview" in the *Virtuoso Schematic Editor L User Guide* for detailed information about using this form.
2. Click *Browse*.

   The Library Browser form appears.

3. On the Library Browser form, select the existing symbol view you want to use.



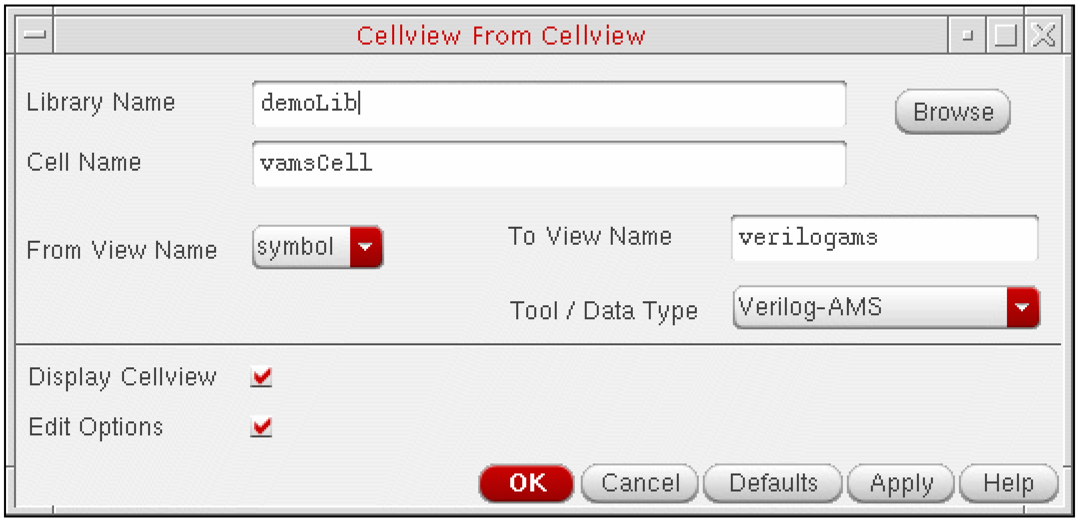
The library, cell, and view names appear in the corresponding fields on the Cellview from Cellview form.



Library Name: demoLib
Cell Name: vamsCell
From View Name: symbol
To View Name: symbol
Tool / Data Type: schematicSymbol
Display Cellview ✓
Edit Options ✓

4. Using the *Tool / Data Type* drop-down combo box, select *Verilog-AMS*.

verilogams appears in the *To View Name* field.



Library Name: demoLib
Cell Name: vamsCell
From View Name: symbol
To View Name: verilogams
Tool / Data Type: Verilog-AMS
Display Cellview: ✓
Edit Options: ✓

5. Click *OK*.

   A template for the Verilog-AMS module appears in a text editor window. The AMS Designer environment creates the module template based on the *From View Name* symbol data, including any pin and parameter information. For example:

```
//Verilog-AMS HDL for "libraryName", "cellName" "viewName"
`include "constants.vams"
`include "disciplines.vams"

module samplehold ( holdSig, inSig, trigger );

    input inSig;
    input trigger;
    output holdSig;
parameter model = ...;

parameter real m=...;
endmodule
```

6. Finish coding the module, then save and exit the text editor.

   The AMS Designer environment verifies correct syntax and creates the Verilog-AMS cellview.

   Note: If the syntax checker encounters any problems, error messages appear in another window.

Creating a VHDL-AMS Cellview from an Existing Symbol or Block

To create a VHDL-AMS cellview from an existing symbol or block, do the following:

1. In the schematic editor, choose *Create – Cellview – From Cellview*.

   The Cellview from Cellview form appears.

   See "Automatically Creating a Cellview from another Cellview" in the *Virtuoso Schematic Editor L User Guide* for detailed information about using this form.

2. Click *Browse*.

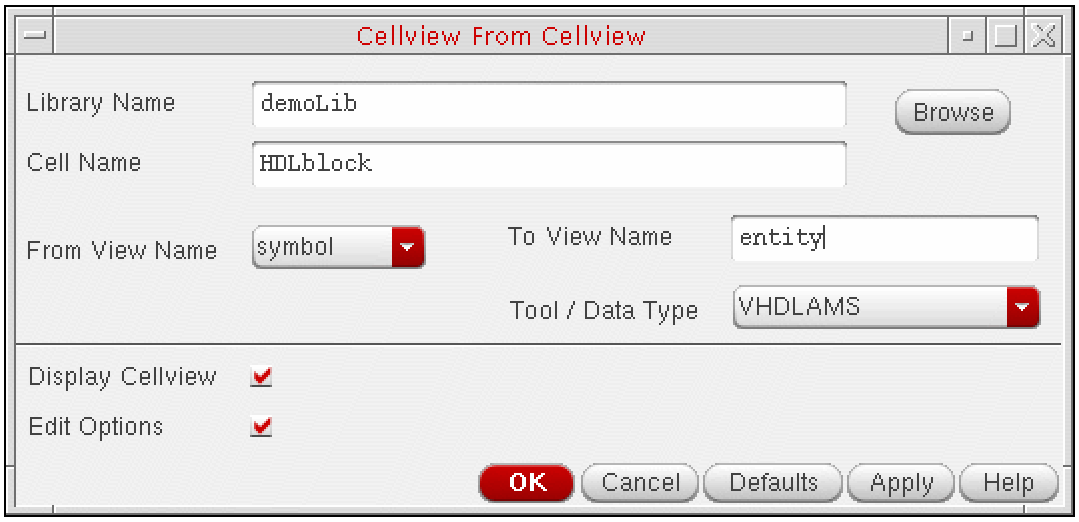
   The Library Browser form appears.

3. On the Library Browser form, select the existing symbol view you want to use.

   The library, cell, and view names appear in the corresponding fields on the Cellview from Cellview form.

4. Using the *Tool / Data Type* drop-down combo box, select *VHDLAMS*.

   entity appears in the *To View Name* field.



Cellview From Cellview

Library Name	demoLib
Cell Name	HDLblock
From View Name	symbol
To View Name	entity
Tool / Data Type	VHDLAMS
Display Cellview	✓
Edit Options	✓

   Note: If you want to create an architecture instead of an entity, type any other name in the *To View Name* field. The program will create a template for a VHDL-AMS architecture using the *To View Name* and the *Cell Name*, such as:

```
...
architecture viewName of cellName is ...
```

5. Click *OK*.

   A template for the VHDL-AMS entity (or architecture) appears in a text editor window. The AMS Designer environment creates the template based on the *From View Name* symbol data, including any pin and parameter information. For example:

```
library ieee, std;
use ieee.std_logic_1164.all;
use ieee.electrical_systems.all;
entity \cellName\ is
port ( terminal pin1 :  electrical;
terminal pin2 :  electrical;
terminal pin3 :  electrical );
end \cellName\;
```

6. Finish coding the entity (or architecture), then save and exit the text editor.

The AMS Designer environment verifies correct syntax and creates the VHDL-AMS cellview.

Note: If the syntax checker encounters any problems, error messages appear in another window.

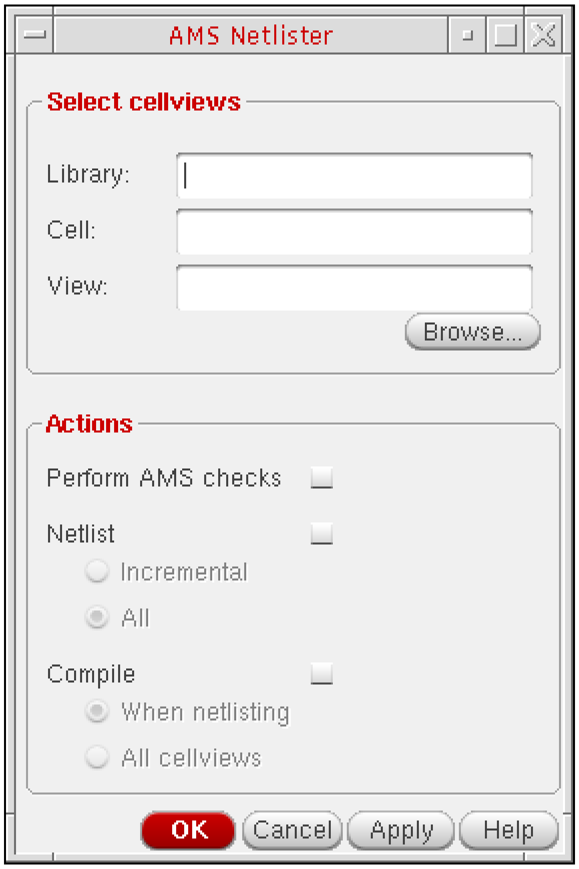
Creating HDL Source Files Outside the AMS Designer Environment

You can create Verilog (digital), Verilog-AMS, VHDL (digital), and VHDL-AMS source files either inside or outside the AMS Designer environment. However, if you use source files created outside the AMS Designer environment, you forfeit the automatic cross-checking among views that the environment performs. For example, if you edit the port list of a text view outside the environment, you must remember to update the corresponding symbol view. When you make a similar change in the environment, AMS Designer detects the change and prompts you to update the symbol view.

When you update an HDL view outside the environment, you must be sure to compile it by doing the following:

1. In the command interpreter window (CIW), choose *Tools – AMS – Netlist*.

The AMS Netlister form appears.



See “Library Netlisting from the CIW” on page 187 for details about using this form.

2. In the *Actions* group box, turn on *Compile* and select *All cellviews*.

3. Click *OK*.

Creating a New Verilog-AMS Module Cellview

To create a new cellview that is a Verilog-AMS module, do the following:

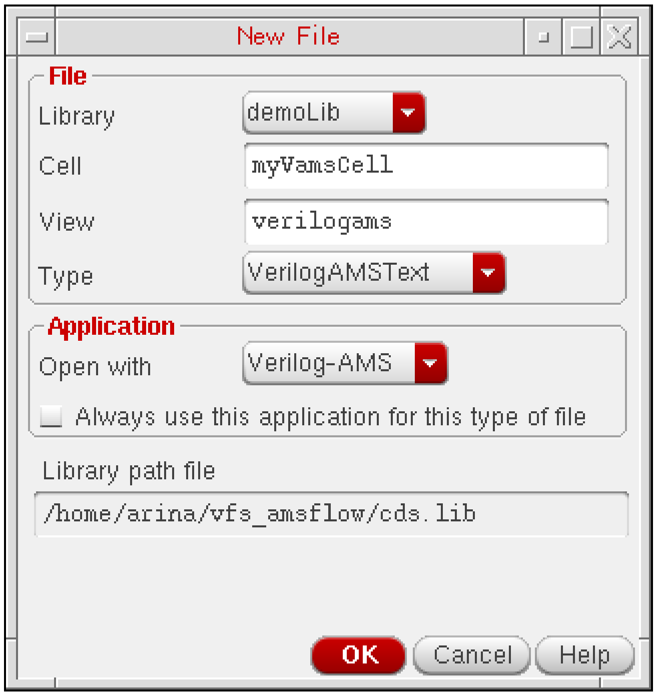
1. In the CIW, choose *File – New – Cellview*.

   The New File form appears.

   See "Creating a New Cellview" in the *Cadence Library Manager User Guide* for detailed information about using this form.

2. In the *Library* field, select a library.
3. In the *Cell* field, type a cell name.
4. Using the drop-down combo box in the *Type* field, select *VerilogAMSText*.

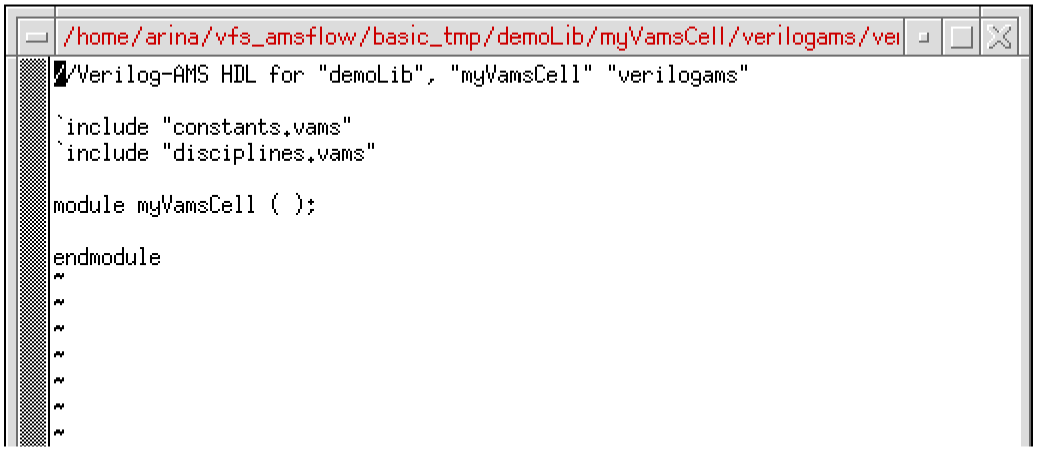
   verilogams appears in the *View* field.



Library: demoLib
Cell: myVamsCell
View: verilogams
Type: VerilogAMSText
Open with: Verilog-AMS
☐ Always use this application for this type of file
Library path file: /home/arina/vfs_amsflow/cds.lib

5. Click *OK*.

A skeleton (preliminary lines of code) for the Verilog-AMS module appears in a text editing window. The module name matches the cell name.



/home/arina/vfs_amsflow/basic_tmp/demoLib/myVamsCell/verilogams/ve

```
//Verilog-AMS HDL for "demoLib", "myVamsCell" "verilogams"

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

module myVamsCell ( );

endmodule
~
~
~
~
~
~
~
```

6. Finish coding the module, then save and exit the text editor.

Important

The module name must match the cell name. Do not change the module name.

Creating a New VHDL-AMS Module Cellview

To create a new cellview that is a VHDL-AMS module, do the following:

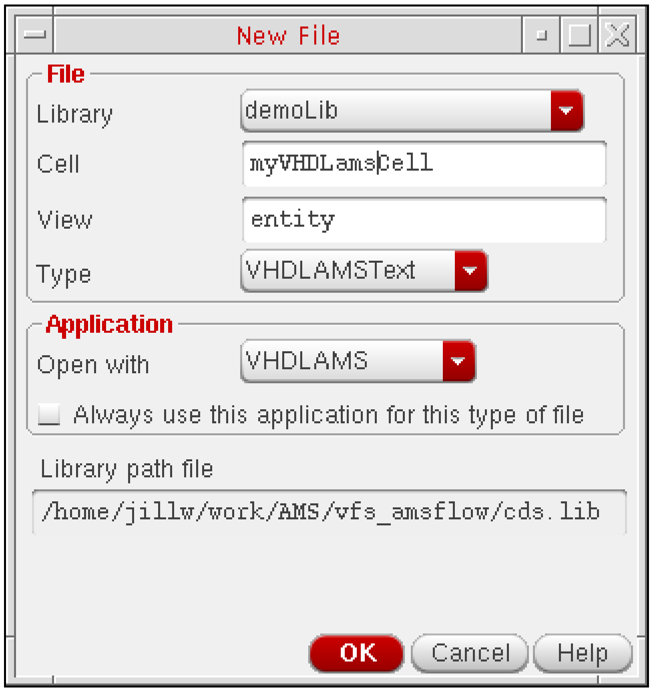
1. In the CIW, choose *File – New – Cellview*.

   The New File form appears.

   See "Creating a New Cellview" in the *Cadence Library Manager User Guide* for detailed information about using this form.

2. In the *Library* field, select a library.
3. In the *Cell* field, type a cell name.
4. Using the drop-down combo box in the *Type* field, select *VHDLAMSText*.

entity appears in the *View* field.



Library: demoLib
Cell: myVHDLamsCell
View: entity
Type: VHDLAMSText
Open with: VHDLAMS
Always use this application for this type of file
Library path file: /home/jillw/work/AMS/vfs_amsflow/cds.lib

To comply with AMS guidelines, the view name should be all lowercase. If you do not change the default view name (*entity*), the program creates a VHDL-AMS entity. If you type a view name other than `entity`, the program creates an architecture (which describes the behavior of an entity).

5. Click *OK*.

A skeleton (preliminary lines of code) for the VHDL-AMS design unit appears in a text editing window. The entity name matches the cell name. (The architecture name matches the view name.)

As an entity, the skeleton might look something like this:

```
library ieee, std;
use ieee.std_logic_1164.all;
use ieee.electrical_systems.all;
entity \cellName\ is
end \cellName\;
```

As an architecture, the skeleton might look something like this:

```
library ieee, std;
use ieee.std_logic_1164.all;
use ieee.electrical_systems.all;

architecture viewName of cellName is
begin
end viewName;
```

6. Finish coding the entity or architecture, then save and exit the text editor.

Important

The entity name must match the cell name. Do not change the entity name.

Creating a Symbol Cellview from a Verilog-AMS Cellview

If you created a Verilog-AMS cellview without creating a symbol for it, or if you have a component that has only a Verilog-AMS cellview, you can add a symbol view by following these steps:

1. In the schematic editor, choose *Create – Cellview – From Cellview*.

   The Cellview from Cellview form appears.

   See "Automatically Creating a Cellview from another Cellview" in the *Virtuoso Schematic Editor L User Guide* for detailed information about using this form.

2. Click *Browse*.

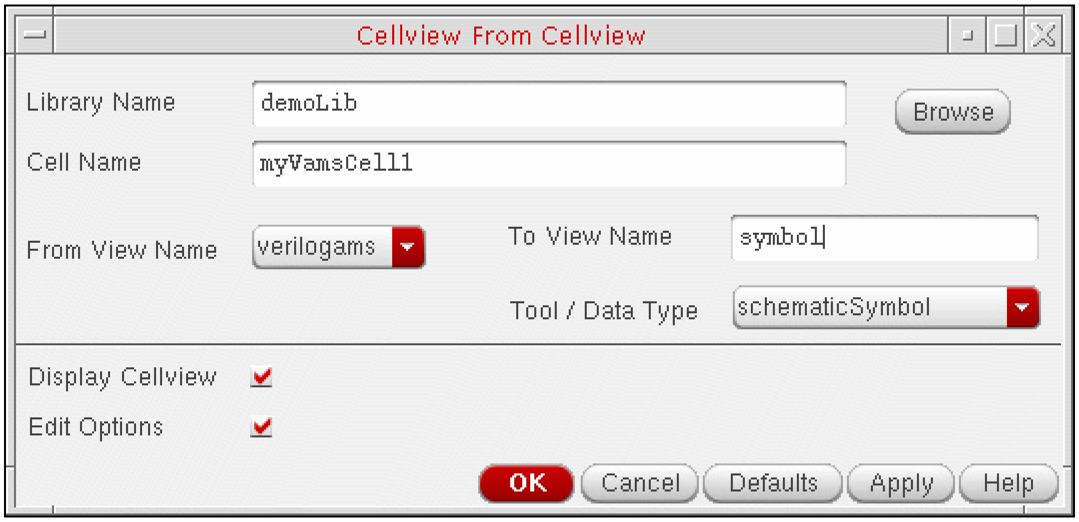
   The Library Browser form appears.

3. On the Library Browser form, select the existing *verilogams* view for which you want to create a symbol.

   The library, cell, and view names appear in the corresponding fields on the Cellview from Cellview form.

4. Using the *Tool / Data Type* drop-down combo box, select *schematicSymbol*.

symbol appears in the *To View Name* field.



Library Name: demoLib
Cell Name: myVamsCell1
From View Name: verilogams
To View Name: symbol
Tool / Data Type: schematicSymbol
Display Cellview: ✓
Edit Options: ✓

5. Click *OK*.

   The Symbol Generation Options form appears.

   See “Editing Symbol Generation Options” in the *Virtuoso Schematic Editor L User Guide* for details about using this form.

6. Click *OK*.

   A Virtuoso® Symbol Editor window appears.

7. Edit the symbol, save it, and close the Symbol Editor window.

Note: If you modify the pin directions in the Verilog AMS file and save the file, Virtuoso displays a message informing you about the mismatch in the pin directions and confirms if you want to update the symbol view. To update the symbol view, select Yes. Otherwise, select No. Alternatively, you can use the SKILL variable `vmsUpdateSymbolForDirectionMismatch`. This variable can have the following three values:

`query`: Displays a message informing you about the mismatch in the pin directions and confirms if you want to update the symbol view. This is the default value.

`t`: Updates the symbol automatically

`nil`: Does not update the symbol

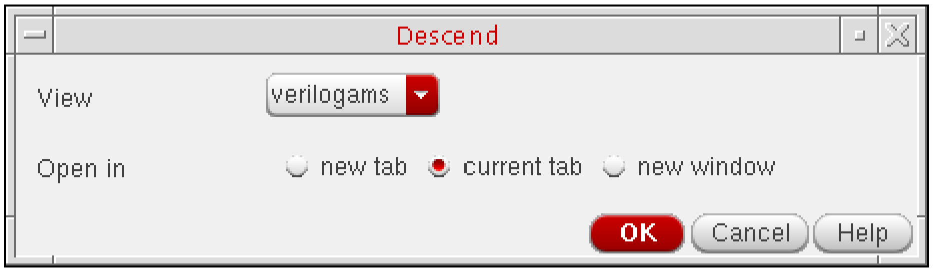
Viewing Source Code for an HDL Cellview

To view the source code (Verilog-AMS, VHDL-AMS) for an HDL cellview, do the following:

1. In the schematic editor, select an instance that has an HDL cellview.
2. Choose *Edit – Hierarchy – Descend Edit*.

   The Descend form appears.

   See "Descending Using the Descend Command" in the *Virtuoso Schematic Editor L User Guide* for details about using this form.
3. Using the drop-down combo box in the *View* field, select a Verilog-AMS or VHDL-AMS cellview. For example, select *verilogams*.

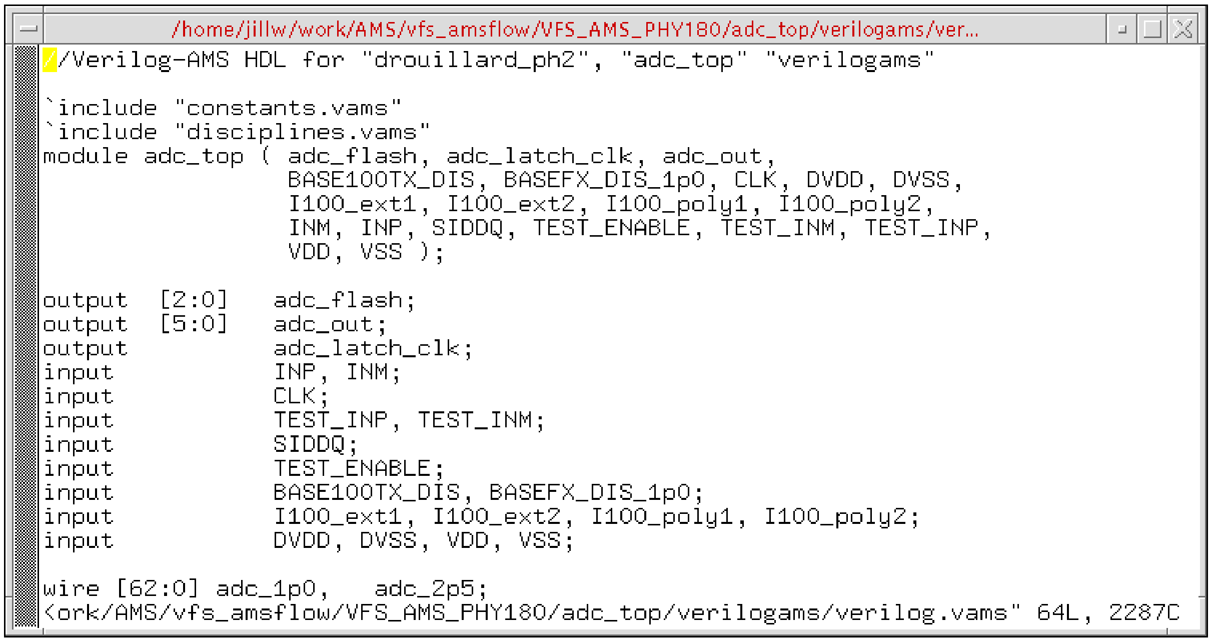

Descend

View: verilogams

Open in: new tab [✓ current tab] new window

4. Click *OK*.

The source code for the instance appears in a text editing window. For example:



/home/jillw/work/AMS/vfs_amsflow/VFS_AMS_PHY180/adc_top/verilogams/ver...

```
//Verilog-AMS HDL for "drouillard_ph2", "adc_top" "verilogams"

`include "constants.vams"
`include "disciplines.vams"
module adc_top ( adc_flash, adc_latch_clk, adc_out,
                 BASE100TX_DIS, BASEFX_DIS_1p0, CLK, DVDD, DVSS,
                 I100_ext1, I100_ext2, I100_poly1, I100_poly2,
                 INM, INP, SIDDQ, TEST_ENABLE, TEST_INM, TEST_INP,
                 VDD, VSS );

output  [2:0]   adc_flash;
output  [5:0]   adc_out;
output          adc_latch_clk;
input           INP, INM;
input           CLK;
input           TEST_INP, TEST_INM;
input           SIDDQ;
input           TEST_ENABLE;
input           BASE100TX_DIS, BASEFX_DIS_1p0;
input           I100_ext1, I100_ext2, I100_poly1, I100_poly2;
input           DVDD, DVSS, VDD, VSS;

wire [62:0] adc_1p0,   adc_2p5;
<ork/AMS/vfs_amsflow/VFS_AMS_PHY180/adc_top/verilogams/verilog.vams" 64L, 2287C
```

Using Net and Pin Properties

The AMS Designer environment supports the following properties:

`netType`	Specifies the type of a net. The type must be one of `supply0`, `supply1`, `tri`, `tri0`, `tri1`, `triand`, `trior`, `trireg`, `wand`, `wire`, `wor`, or `wreal`. For example, setting the property `netType=wand` on net `myNet` results in the netlister writing the net declaration as follows: `wand myNet;`
`netDiscipline`	Specifies the discipline for a net. For example, setting the property `netDiscipline=electrical` on net `myNet` results in the netlister writing the net declaration as follows: `electrical myNet;`

`supplySensitivity` `groundSensitivity`	Specifies names of signals, typically global signals, to which you want a connect module to be sensitive When you specify a value for either the `supplySensitivity` or the `groundSensitivity` property on a signal in a connect module, the declared signal (in the connect module) takes on the value of the `supplySensitivity` or `groundSensitivity` signal you specify. When you specify a value for the `supplySensitivity` or the `groundSensitivity` property (or both) on a signal in an ordinary module, the value of the `supplySensitivity` or `groundSensitivity` signal overrides the value of the signal of the same name in the connect module to which the ordinary module connects. For more information, see “Specifying Supply Sensitivity Attributes“ in the Cadence Verilog-AMS Language Reference.

groundSensitivity and supplySensitivity Properties

The `groundSensitivity` and `supplySensitivity` properties provide a way to make a connect module sensitive to supplies in the module to which the connect module is connected.

Typically, the port of the connected module is a digital port. It is possible to make a connect module sensitive to supplies in an analog port, but making the connect module sensitive to supplies in the connected digital port is much more likely to produce the behavior that you expect. This is so because

- When the connect module converts analog signals to digital values, the decision to output a one or a zero depends on the relationship between the analog signal and a threshold value. The threshold value is determined by the supply values in the component that includes the connected digital port.
- When the connect module converts digital values to analog signals, the connect module needs to determine what voltage to produce for each digital input value. Again, that voltage depends on the supplies in the component that includes the connected digital port.

Overview of the Sensitivity Properties

The `groundSensitivity` and `supplySensitivity` properties, which are added to a port or pin definition, have the following syntax.

```
sensitivity_properties ::=
    (* [ integer groundSensitivity = "sig1_sensitive_to" ; ]
       [ integer supplySensitivity = "sig2_sensitive_to" ; ] *)
```

sig1_sensitive_to, *sig2sensitive_to*

Names of signals, typically global signals, to which a connect module is made sensitive.

When the `groundSensitivity` property is included as part of a signal declaration in the connect module, the declared signal takes on, by default, the value of *sig1_sensitive_to*. When the `groundSensitivity` property is included as part of a signal declaration in an ordinary module, the *sig1_sensitive_to* value in that module overrides the *sig1_sensitive_to* value specified in the connect module. The `supplySensitivity` property works similarly.

For example, the connect module might be defined as follows.

```
connectmodule l_to_e(dval, aval);
    ...
    electrical (* integer groundSensitivity = "global_pwr.pow1" ; *) gnd ;
    ...
endmodule
```

This connect module is connected to the digital port `d` in an ordinary module that is defined as follows.

```
module sample(d);
    output (* integer groundSensitivity = "global_pwr.pow5" ; *) d ;
...
endmodule
```

In this example, `gnd` is defined in the connect module as taking on, by default, the value of `global_pwr.pow1`, but that value is overridden by the value `global_pwr.pow5` specified in the module `sample` when the connect module is inserted across the digital port `d`. To generalize, if the `groundSensitivity` property is not used in the ordinary module, the connect module uses the default value specified on the `groundSensitivity` property in the connect module.

Basic Principles for Using the Sensitivity Properties

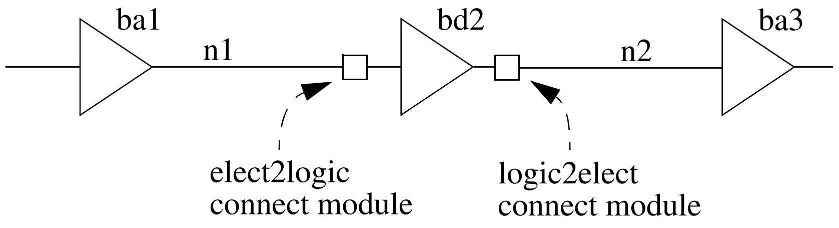
Some basic principles will help you use the `groundSensitivity` and `supplySensitivity` properties correctly.

- Connect modules are always inserted between a digital port and an analog net. When you use the `groundSensitivity` and `supplySensitivity` properties, you make the connect module sensitive to the signals on the digital port. That is true whatever the direction of the port might be.

- There are two steps involved in establishing ground or supply sensitivity: inserting the necessary properties in the connect module, and adding the corresponding properties to the connected digital port. If the connected digital port is part of a schematic, you define the properties on the connected pin in the schematic. If the connected digital port is defined in a text module, you add the properties to the port definition in the module.
- The default value associated with the `groundSensitivity` and `supplySensitivity` properties must be the name of a signal, not the name of a property.
- You must use detailed discipline resolution or the sensitivity properties have no effect.

Example: Using the Sensitivity Properties in a Chain of Buffers

Assume that you have the following schematic containing three buffers. Buffers `ba1` and `ba3` are instances of a module that is implemented as an analog block with analog input and output pins. Buffer `bd2` is implemented as a digital block, with logic input and output pins.



During elaboration, connect modules are inserted across net `n1` and the digital input port of buffer `bd2`, and across the digital output port of buffer `bd2` and net `n2`.

Assume that the string of buffers is designed to run at 5.0 volts. The connect module must then be written to work at that voltage. For example, an A2D connect module with hardcoded thresholds set for 5.0 volts might look like this.

```
‘include "disciplines.vams"
connectmodule elect2logic(aVal, dVal);
    output dVal;
    input aVal;
    logic dVal;
    electrical aVal;

    reg temp;

    always begin              // Digital, do this always.
    if(V(aVal) >  3.0)
        #1 temp = 1;          // Delay 1 time unit,drive output 1
    else if (V(aVal) < 2.0)
        #1 temp = 0;          // or drive output 0, depending on aVal.
    else
        #1 temp = 1’bx;
    end

    assign dVal = temp;        // Bind register to digital output.
```

```
endmodule
```

But assume now that the string of buffers can run at either 3.0 volts or 5.0 volts, depending on the supplies that are provided. To make the connect module sensitive to the supplies, you use the `groundSensitivity` and `supplySensitivity` properties, and rewrite the `always` statement so that the threshold is calculated from the supply and ground values.

```
‘include "disciplines.vams"

connectmodule elect2logic(aVal, dVal);
    output dVal;
    input aVal;
    logic dVal;
    electrical aVal;

    electrical (* integer supplySensitivity = "cds_globals.\\vdd! " ; *) \vdd! ;
    electrical (* integer groundSensitivity = "cds_globals.\\vss! " ; *) \vss! ;

    reg temp;

    always begin                  // Do this always.
    if(V(aVal) >  ((V(\vdd! ) - V(\vss! ))/2 + 0.5 ))
        #1 temp = 1;              // Delay 1 time unit,drive output 1.
    else if (V(aVal) < ((V(\vdd! ) - V(\vss! ))/2 -0.5))
        #1 temp = 0;              // or drive output 0, depending on aVal.
    else
        #1 temp = 1’bx;
    end

    assign dVal = temp;            // Bind register to digital output.
endmodule
```

The next step is to specify the digital ports to which the connect module is sensitive. To do that, you add the `groundSensitivity` and `supplySensitivity` properties to the connected digital port. In the buffer string example illustrated above, connect modules are connected to both the input and the output ports of buffer `bd2` and must therefore be sensitive to the supplies in those ports. In this case, the `groundSensitivity` and `supplySensitivity` properties must be added to both ports of the buffer, like this.

```
module bux2_5V (Z,A);

input
    (* integer supplySensitivity="\\vdd! ";
       integer groundSensitivity="\\vss! "; *)
A ;
output
    (* integer supplySensitivity="\\vdd! ";
       integer groundSensitivity="\\vss! "; *)
Z;

wire \vss! ;
wire \vdd! ;

analog begin
    V(\vss! ) <+ 0.0 ;
    V(\vdd! ) <+ 5.0 ;
end

buf #1 (Z,A);

specify
    specparam
    t_A_Z_rise = 0.1,
    t_A_Z_fall = 0.1;

    // Delays
    (A +=> Z) = (t_A_Z_rise,t_A_Z_fall);
endspecify

endmodule
```

Making Connect Modules Sensitive to Ground and Supply

This section describes how to use the `groundSensitivity` and `supplySensitivity` properties to make a connect module sensitive to supplies whose values are set by inherited connections. You might use this capability, for example, when you want to be able to switch between two different power supplies and have connect modules act differently depending on a value that is provided by inherited connection. This capability provides a powerful multiple supply capability but requires significant changes to the affected modules.

The primary change involved in making the connect module sensitive to values that are determined by inherited connections is to the declarations of the ports in the ordinary module to which the connect module is sensitive. You use inherited connections to set the values of the signals in the ports and use the sensitivity properties to make the connect module sensitive to those values.

The following example illustrates how to set up the inherited connections and sensitivities in the ordinary module. Sensitivities are specified for both the input and output ports, `A` and `Z`, so that connect modules can be inserted across and be sensitive to the supplies in either or both of those ports.

```
module bux2 (Z,A);

input
    (* integer supplySensitivity="\\vdd! ";
       integer groundSensitivity="\\vss! "; *)
A ;
output
    (* integer supplySensitivity="\\vdd! ";
       integer groundSensitivity="\\vss! "; *)
Z;

wire
    (* integer inh_conn_prop_name="lSup";
       integer inh_conn_def_value="cds_globals.\\vss! "; *)
\vss! ;
wire
    (* integer inh_conn_prop_name="hSup";
       integer inh_conn_def_value="cds_globals.\\vdd! "; *)
\vdd! ;

buf #1 (Z,A);

`ifdef functional
`else
specify
    specparam
    t_A_Z_rise = 0.1,
    t_A_Z_fall = 0.1;

    // Delays
    (A +=> Z) = (t_A_Z_rise,t_A_Z_fall);
endspecify
‘endif

endmodule
```

Notice how the input port `A` has a specified `supplySensitivity` signal name of `"\\vdd! "`. When a `supplySensitive` connect module is connected to this input port, the connect module becomes sensitive to the value of `"\\vdd! "`.

Next, consider how the value of `"\\vdd! "` is set. That value is set according to the inherited connections properties farther down in the module. For `"\\vdd! "` the relevant specification looks like this.

```
wire
    (* integer inh_conn_prop_name="hSup";
       integer inh_conn_def_value="cds_globals.\\vdd! "; *)
\vdd! ;
```

This statement establishes an inherited connection with the name `hSup` and the default value of `"cds_globals.\\vdd! "`. If the value of `hSup` is not set anywhere above this module, then the value of `\vdd!` is set to the value of `"cds_globals.\\vdd! "`. If inherited connections are used to set a different value for `hSup`, then `\vdd!` takes on the different value, which, because `supplySensitivity` is being used, can be passed on to a connected connect module.

The connect module is set up in the usual way to be sensitive to the value of `"cds_globals.\\vdd! "`. For example, you might prepare the following A2D module to connect to the digital input port `A` mentioned above.

```
‘include "disciplines.vams"

connectmodule elect2logic(aVal, dVal);
    output dVal;
    input aVal;
    logic dVal;
    electrical aVal;

    electrical (* integer supplySensitivity = "cds_globals.\\vdd! " ; *) \vdd! ;
    electrical (* integer groundSensitivity = "cds_globals.\\vss! " ; *) \vss! ;

    reg temp;

    always begin                   // Do this always.
    if(V(aVal) >  ((V(\vdd! ) - V(\vss! ))/2 + 0.5 ))
        #1 temp = 1;               // Delay 1 time unit,drive output 1
    else if (V(aVal) < ((V(\vdd! ) - V(\vss! ))/2 -0.5))
        #1 temp = 0;               // or drive output 0, depending on aVal.
    else
        #1 temp = 1’bx;
    end

    assign dVal = temp;             // Bind register to digital output.

endmodule
```

With this preparation, you can then change the value of `hSup` and that changed value is inherited through the design. The sensitivity properties then make the attached connect modules sensitive to that changed value.

10

Using External Text Designs

You can use the AMS Designer environment to develop and simulate your designs. You can also simulate external text designs: existing and new designs that you develop outside the AMS Designer environment.

Important

There are a few differences between using HDL modules and design units standalone and using them in the AMS Designer environment.

- ❑ To avoid problems reading and writing a file, always specify the full path when opening files inside a module using `$fopen`. The AMS Designer environment might use a run directory that is in a different location from what you expect.
- ❑ When you are using the AMS Designer environment, editing HDL files directly might cause problems. For more information, see "Creating HDL Source Files Outside the AMS Designer Environment" on page 240.

To be able to simulate an external text design, do the following:

1. Specify the working library.
2. Compile your HDL modules into a library (lib/cell/view).
3. Create symbols for text modules that you plan to place on schematics.
4. Create a configuration that uses the text modules.
5. Create and edit a `cds_globals` module to add information about global variables and design variables in external text modules.

You can now set up and run the simulation.

Specifying the Working Library

To specify the working library, use the following command in your `hdl.var` file:

```
DEFINE WORK libraryName
```

If your design contains components from more than one library, define the working library to be the one that contains the top level of your design.

For more information about the `hdl.var` file, see “The hdl.var File” in the *Virtuoso AMS Designer Simulator User Guide*.

Compiling a Module into a Library

To compile a module into a library, use the `ncvlog` or `ncvhdl` command with the `-use5x` option.

The AMS Designer environment works most efficiently with modules that are available in the Cadence library/cell/view structure (also known as “5x”). You can bring modules into Cadence libraries by establishing appropriate links from the cellview directory in the library to the original source information so that AMS Designer is aware of any changes to that source. You do not need to copy modules into libraries.

Note: For more information about the `ncvlog` and `ncvhdl` commands with the `-use5x` and other options, see “Compiling Verilog Source Files with ncvlog” in *NC-Verilog Simulator Help* and “Compiling VHDL Source Files with ncvhdl” in *NC-VHDL Simulator Help*, respectively.

Here are some examples:

Command(s)	Description

`ncvlog -ams -use5x master.vams`
`ncvlog -ams -use5x /mnt4/lgp/master.vams`
	Assumption: `master.vams` contains a module definition for `master`.
	
	Compiles a cell named `master` with the default view name `module` into the current working library.

`ncvlog -ams -use5x -specificunit ncvlog_lib.master1:behavioral master.vams`	Assumption: `master.vams` contains more than one module definition, one of which is `master1`. Compiles a cell named `master1` with the view name `behavioral` into the `ncvlog_lib` library.
`ncvhdl -v93 -ams -use5x daconv2.vhd`	Assumption: `daconv2.vhd` contains an entity named `daconv2` and an architecture named `daconv2_behav`. Compiles a cell named `daconv2` with default view names `entity` and `daconv2_behav` to the current working library.

Note: If you need to bring primitives into a 5X library system, compile them into a library that is used only by primitives. You need to use the Conversion Tool Box to convert primitive cells for use with AMS Designer, but you must avoid converting non-primitive cells. Because the Conversion Tool Box operates on entire libraries, the conversion process requires that the two kinds of cells be located in different libraries.

See also

- Compiling into Temporary Libraries on page 261
- Binding to a New Cellview in a Temporary Library on page 261
- Listing Compiled Modules on page 286

Compiling into Temporary Libraries

If the access permissions for a library do not allow writing, you can establish a corresponding temporary (TMP) library that does allow writing. The software can write only derived data to TMP libraries. If a non-writable library already contains a particular cellview, you can compile this cellview into a TMP library.

To set up a temporary library, add the following commands to your `cds.lib` file:

```
DEFINE masterLibraryName directoryPath
ASSIGN masterLibraryName TMP TMPdirPath
```

The first line defines the master library. The second line assigns a TMP library to the master. You can create new cells in the TMP library.

For example:

```
DEFINE mylib ./mylib
ASSIGN mylib TMP ./mylib_tmp
```

To create a new module view in the TMP library, you can type a command like the following:

```
ncvlog -ams -use 5x -work mylib -view module cellA.vams
```

AMS Designer compiles the `module` view for `cellA` into the TMP library for `mylib`. If you want to bind to the `module` cellview, you must set the `CDS_BIND_TMP_DD` environment variable.

Note: For more information about TMP libraries, see "Temporary Directory for a Library" in the "Cadence Library Structure" chapter of the *Cadence Application Infrastructure User Guide*.

Binding to a New Cellview in a Temporary Library

If you want to be able to bind to new cellviews you compile into a TMP library you must set the `CDS_BIND_TMP_DD` shell environment variable. When you set this variable, you can create cells and views in TMP libraries and create bindings to the new views.

The following table describes the effects of the `CDS_BIND_TMP_DD` variable values. Setting this variable to any other value has the same effect as not setting the variable at all. These values are case-insensitive.

Value	Effect
`both`, `cell`, `true`, or `yes`	Allows you to bind to cellviews in a TMP library.
`view`	Allows the creation of new views (only) in a TMP library, even when the corresponding master data does not exist in the master library.

For example,

1. To assign a TMP library for a read-only working library called `amslib` , add the following lines in your `cds.lib` file:

   ```
   DEFINE amslib ./AMS_lib/amsLib
   ASSIGN amslib TMP /tmp/amslib_tmp
   ```

   The first line defines the working library, `amslib`. The second line assigns a TMP library to `amslib`. You can create new cells in the TMP library.

2. If you set `CDS_BIND_TMP_DD` to `both`, `cell`, `true`, or `yes`, you can compile a new cellview into your TMP library as follows:

   ```
   setenv CDS_BIND_TMP_DD both
   ncvlog -ams -use5x -work amslib -cell compar6 -view verilogams compar6.vams
   ```

   This combination compiles a `verilogams` view for cell `compar6` into theTMP library for the read-only `amslib` master library here:

   ```
   amslib/compar6/verilogams/compar6.vams
   ```

3. If you set `CDS_BIND_TMP_DD` to `view`, you can compile the `compar6` module as a new `verilogams` view for a cell already in your TMP library as follows:

   ```
   setenv CDS_BIND_TMP_DD view
   ncvlog -ams -use5x -work amslib -cell myCell -view verilogams compar6.vams
   ```

   This combination compiles a `verilogams` view for module `compar6` into the already-existing `myCell` in theTMP library for the read-only `amslib` master library here:

   ```
   amslib/myCell/verilogams/compar6.vams
   ```

See also "Compiling a Module into a Library" on page 259.

Note: For more information about TMP libraries, see "Temporary Directory for a Library" in the "Cadence Library Structure" chapter of the *Cadence Application Infrastructure User Guide*.

Creating a Configuration with a View List for AMS

Once you have compiled your HDL modules into a library and created symbols for your text modules, you can use the Virtuoso® Hierarchy Editor to create a configuration cellview that has a view list for AMS. (See also "Creating a Config Cellview" on page 175.) You can create a new configuration cellview by opening the New Configuration form from the command interpreter window (CIW) or from the Library Manager. See the following topics for how to open the New Configuration form:

- Opening the New Configuration Form from the CIW on page 264
- Opening the New Configuration Form from the Library Manager on page 264

Once the New Configuration form appears, you can create a configuration cellview with a view list for AMS as follows:

1. On the New Configration form, click *Use Template*.

   The Use Template form appears.

2. Using the drop-down combo box in the *Name* field, select *AMS_Compatibility*.

   The resulting view list is

   ```
   stimulus dataflow behavioral behavior functional structure hdl verilogams
   veriloga verilogNetlist system spectre spice cmos.sch cmos_sch schematic symbol
   ```

   If you select the *AMS* template to create a configuration, the resulting view list is

   ```
   verilogams veriloga behavioral functional schematic symbol
   ```

   Note: Symbol views must have associated models that describe the represented device for AMS simulation.

   However, if you are working with an external text design, the *AMS_Compatibility* template might do a better job of selecting appropriate views.

3. Click *OK*.

   All of the design instances and their cell bindings appear in the Virtuoso® Hierarchy Editor window.

4. Choose *View – Update* to check and save the new configuration.

   An Update prompt appears.

5. Click *OK* to save the new config view.

Opening the New Configuration Form from the CIW

To open the New Configuration form from the CIW, do the following:

1. Choose *File – New – Cellview*.

   The New File form appears.

2. Select a library and specify a cell name.
3. In the *Type* drop-down combo box, select *config*.

   config appears in the *View* field and *Hierarchy Editor* appears in the *Open with* field in the *Application* group box.

4. Click *OK*.

   The New Configuration form appears.

Opening the New Configuration Form from the Library Manager

To open the New Configuration form from the Library Manager, do the following:

1. In the *Library* and *Cell* columns, select the cell for which you want to create the configuration cellview.
2. Choose *File – New – Cell View*.

   The New File form appears.

3. In the *Type* drop-down combo box, select *config*.

   config appears in the *View* field and *Hierarchy Editor* appears in the *Open with* field in the *Application* group box.

4. Click *OK*.

   The New Configuration form appears.

Creating a cds_globals Module for External Text Designs

After bringing text modules into Cadence libraries, you need to netlist your design to generate a `cds_globals` module. Initially, the `cds_globals` module contains information about global signals and design variables in translated schematics but not about signals and variables in external text modules, so you might need to edit the `cds_globals` module.

See the following topics for more information:

- Global Signals on page 199
- Global Design Data Module (cds_globals) on page 200

11

Using Existing Analog Design Units

You can use existing analog primitives and SPICE and Spectre® design units in the AMS Designer environment. For information about converting analog primitives for use with the cellview-based netlister, see “Converting an Existing Analog Primitive Library” on page 660. For information about using SPICE and Spectre netlists, models, and subcircuits, see the following topics:

- Preparing to Use SPICE and Spectre Design Units on page 268
- Placing SPICE and Spectre Design Units on a Schematic on page 269
- Editing AMS Simulation Information on page 269

For additional information, see “Using Subcircuits and Models Written in SPICE or Spectre” in the “Preparing the Design: Using Analog Primitives and Subcircuits” chapter of the *Virtuoso AMS Designer Simulator User Guide*.

Preparing to Use SPICE and Spectre Design Units

To prepare to use a SPICE or Spectre model or subcircuit, do the following:

1. Include the netlist or subcircuit in a model file.

   The contents of the model file must be in Spectre or SPICE syntax.

   Spectre files must start with

   ```
   simulator lang=spectre
   ```

   SPICE files must start with

   ```
   simulator lang=spice
   ```

   and end with

   ```
   simulator lang=spectre
   ```

2. Give the elaborator the location of the model file using one of the following methods:

   ❑ Define the `MODELPATH` variable in the `hdl.var` file:

   ```
   define MODELPATH model_filename
   ```

   ❑ Use the `-modelpath` option for `ncelab`.

   ❑ Use the Model Library Setup form.

Placing SPICE and Spectre Design Units on a Schematic

To place a SPICE or Spectre design unit on a schematic, do the following:

1. Prepare it as described in “Preparing to Use SPICE and Spectre Design Units” on page 268.
2. Create a symbol to represent it on the schematic.

Tip

You can use the Library Manager to copy an existing symbol, then modify the new symbol as necessary.

3. If you plan to use the cellview-based netlister, you will need to edit the ams simInfo.
4. Place the symbol on your schematic.

   For more information about this step, see the “Creating Schematics” chapter of the *Virtuoso Schematic Editor User Guide*.

Editing AMS Simulation Information

To edit AMS simulation information in the component description format (CDF) for the cell, do the following:

1. In the command interpreter window (CIW), choose *Tools – CDF – Edit*.

   The Edit CDF form appears.
2. In the *CDF Layer* group box, select *Base*.
3. In the *Library Name* field, select the library containing the new symbol.
4. In the *Cell Name* field, select the cell name of the new symbol.
5. Select the *Simulation Information* tab.
6. In the *Choose Simulator* field, select *ams*.

   Fields appropriate for the AMS simulator appear on the tab.
7. In the *termOrder* field, type the terminal names in netlisting order.

8. If your SPICE or Spectre design unit supports models (as, for example, the `cap` and `diode` primitives in `analogLib` do), do the following:

   a. In the *otherParameters* field, type `model`.

   b. In the *isPrimitive* field, type `t`.

   The netlister will translate this cell to a Spectre primitive that supports models.

9. If you are preparing a Spectre built-in primitive that does not support models (such as, for example, `vdc` or `vsin` in `analogLib`), type the name of the primitive in the *componentName* field.

10. If you need to add any additional component parameters, see the "Defining Parameters" chapter of the *Component Description Format User Guide*.

11. Click *OK*.

For more information about editing simInfo, see the "Modifying Simulation Information" chapter in the *Component Description Format User Guide*.

12

Creating and Using a Test Fixture Module

You can create and use a behavioral module for a test fixture that instantiates a component, applies stimuli, and checks the resulting outputs. The top level of your design can be a VHDL or Verilog®-AMS module (behavioral) or it can be a schematic (structural). Test fixtures can contain both structural and behavioral elements. (Schematic test fixtures are structural only: They contain an instance of the top-level module, analog stimuli, and analog sources that deliver analog stimuli.) Because AMS Designer netlists schematics as Verilog-AMS modules, creating and using a test fixture ultimately involves instantiating a text module (the unit under test or design under test) in the test fixture.

You can use any behavioral language that AMS Designer supports to write the test fixture. Differences in the way AMS Designer supports these languages might make you prefer to use one language over another. You can create the test fixture module either in or outside of the AMS Designer environment. See the following topics for details:

- Creating a Verilog-AMS Test Fixture in the AMS Designer Environment on page 273
- Creating a Verilog-AMS Test Fixture Outside the AMS Designer Environment on page 274

Because the test fixture module is at the highest level, it has no ports. The content of the test fixture module depends on what inputs and outputs you need to provide and examine. A typical Verilog-AMS structure might look like this:

```
module test_fixture_name () ;          // There are no ports.
    signal_declarations_for_stimuli
    instantiation_of_top_level_module
    instantiations_of_behavioral_testbench_modules_or_primitives
    digital_behavioral_constructs_like_initial_and_always_blocks
    analog_blocks_to_generate_analog_stimuli
endmodule
```

The test fixture becomes the highest module in the design hierarchy. For AMS Designer, you must specify a config cellview for the top level of the design (see "Creating a Config Cellview" on page 175).

Test fixtures can be simple, perhaps providing only a stimulus, or they can be very complex, testing complete cycles of the top-level module. Test fixtures like the latter might provide stimuli to the inputs, read the outputs, and react by providing new stimuli that depend on the outputs.

Creating a Verilog-AMS Test Fixture in the AMS Designer Environment

To create a Verilog®-AMS test fixture in the AMS Designer environment, do the following:

1. Choose *File – New – Cellview*.

   The New File form appears.

2. Select a library.
3. In the *Type* drop-down combo box, select *VerilogAMSText*.

   `verilogams` appears in the *View* field. *Verilog-AMS* appears in the *Open with* field.

4. In the *Cell* field, type the module name you plan to use.

   The cell name must match the module name.

5. Click *OK*.

   Skeleton code for your Verilog-AMS module appears in a text editor window.
   The module name matches the cell name you typed in the *Cell* field.

```
//Verilog-AMS HDL for "libraryName", "cellName" "verilogams"

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

module cellName ( );

endmodule
```

6. Create content for the test fixture module.

   Note: A test fixture module has no ports.

7. Save the file and exit the text editor.

Creating a Verilog-AMS Test Fixture Outside the AMS Designer Environment

To create a Verilog®-AMS test fixture outside the AMS Designer environment, do the following:

1. Use a text editor to create the module.
2. Save the file that contains the module.
3. Compile the module using the `-use5x` option, which creates a new cell and view for the test fixture in the working library.

See “Using External Text Designs” on page 257 for more information.

Creating and Testing a Verilog-AMS Switch Module Using a Verilog-AMS Test Fixture

The following example illustrates how you can verify the operation of a switch by instantiating its module in a test fixture and running a simulation.

Both modules in this example are Verilog-AMS modules. The `myswitch` module describes a basic on/off switch controlled by a digital control signal. When the control signal is high, the switch passes the current from its input port to its output port. The `testfixture` module instantiates the `myswitch` module.

A file called `switchcomps.vams` contains the `myswitch` module, along with other modules. The `myswitch` module looks like this:

```
//Verilog-AMS HDL for "amslib", "myswitch" "verilogams"

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

module myswitch (analogin, analogout, logicsignal );
input analogin, logicsignal ;
output analogout ;
electrical analogin, analogout ;

analog
    begin
        if (logicsignal == 1) V(analogout) <+ V(analogin) ;
        else I(analogin, analogout) <+ 0.0 ;
    end
endmodule
```

To instantiate the switch module in a test fixture, do the following:

1. Compile the `myswitch` module into the `amslib/myswitch/verilogams` lib/cell/view using the following `ncvlog` command:

   ```
   ncvlog -ams -use5x -specificunit amslib.myswitch:verilogams switchcomps.vams
   ```

2. Create the test fixture in the AMS Designer environment and put it in the `amslib` library. For the cell name, use `testfixture`.

   The following Verilog-AMS module code provides inputs for the instantiated `myswitch` module and reads the outputs to verify that the module operates as it should.

   ```
   module testfixture ( );
   electrical ain, aout ;
   reg logsig ;
   ground gnd ;
   electrical gnd ;

   myswitch mys(ain, aout, logsig) ; // Instantiate the component

   resistor #(.r(1000)) r1 (aout, gnd) ;

   analog
       begin
       V(ain) <+ 0.5 ;                  // Generate the analog stimuli.
       @(cross(V(aout)-0.25, +1))$strobe ("Turns on") ; // Read the output.
       @(cross(V(aout)-0.25, -1))$strobe ("Turns off") ;
   end

   initial begin
       logsig = 'b1 ;
       $strobe ("Switch on") ;
   end

   always begin
       #200 logsig = ~logsig ;          // Generate the digital stimuli.
       #200 if (logsig == 1) $strobe ("Switch on") ;
           else $strobe ("Switch off");
   end

   endmodule
   ```

   At this point, `amslib` contains both modules (`myswitch` and `testfixture`).

3. Create a config cellview for the test fixture.

4. In the Virtuoso Hierarchy Editor, use the *AMS* menu to prepare the design and simulate.

5. (Optional) Use SimVision to examine waveforms and use the information from the test fixture to determine whether your instantiated component works as desired.

13

Specifying Compiler Options

Virtuoso® AMS Designer provides access to Verilog-AMS and VHDL-AMS compiler options so that you can tailor behavior to your needs.

Note: If you are using an `hdl.var` file to define variables and settings for the compiler, elaborator, and simulator, see "Using Quick Setup" on page 83 and "Specifying an hdl.var File" on page 134.

See the following topics for more information:

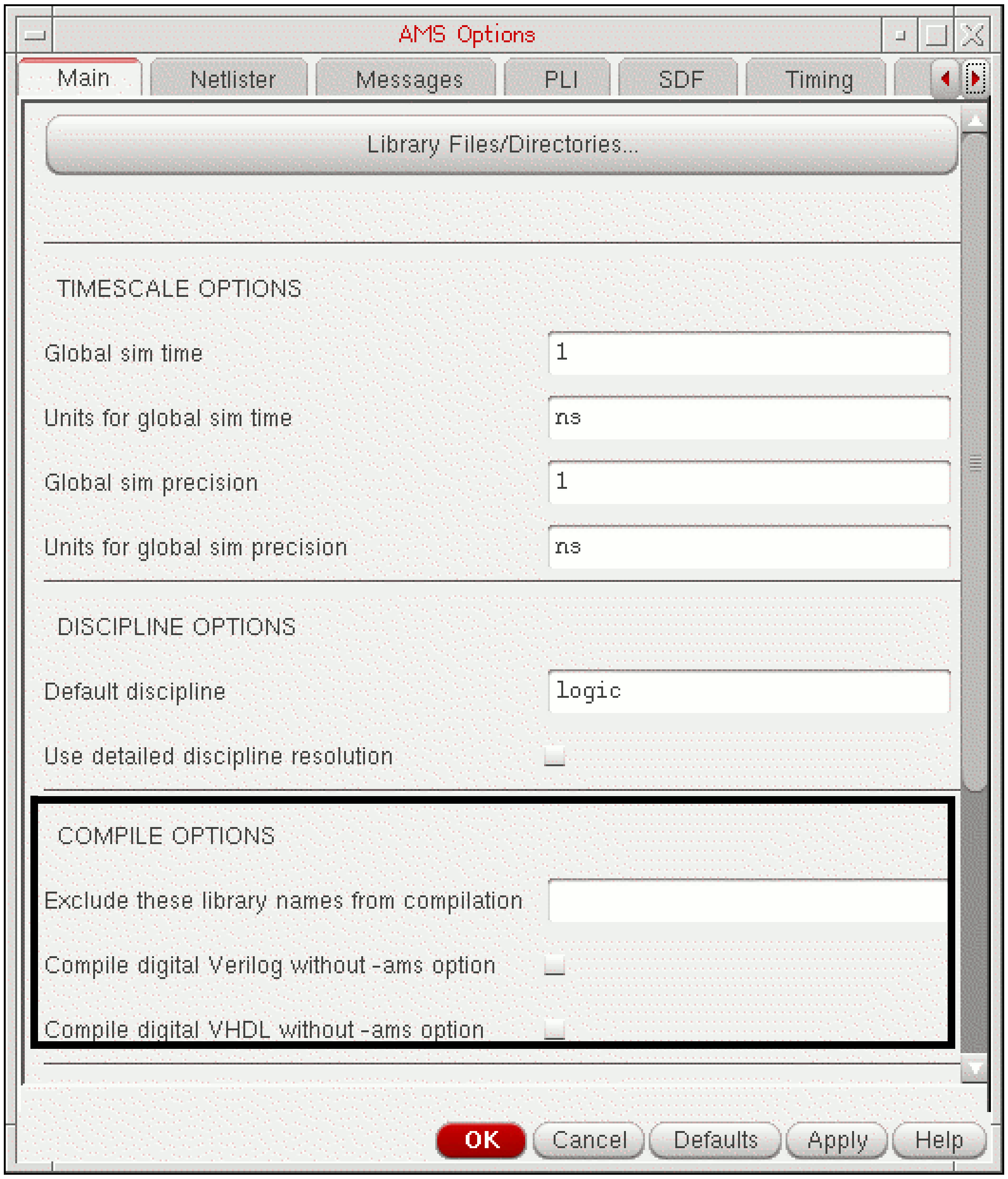
Specifying Libraries to Exclude during Compilation

To specify libraries you want to exclude during compilation, do the following:

1. In the Virtuoso® Hierarchy Editor, choose *AMS – Detailed Setup — AMS Options*.

   The AMS Options form appears.

2. On the *Main* tab, scroll down to the *COMPILE OPTIONS* group box.



TIMESCALE OPTIONS

Global sim time	1
Units for global sim time	ns
Global sim precision	1
Units for global sim precision	ns

DISCIPLINE OPTIONS

Default discipline	logic
Use detailed discipline resolution	☐

COMPILE OPTIONS

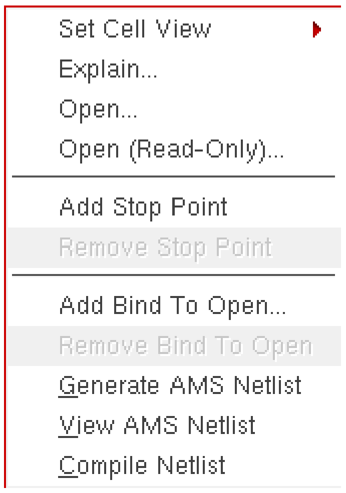
Exclude these library names from compilation	
Compile digital Verilog without -ams option	☐
Compile digital VHDL without -ams option	☐

3. In the *Exclude these library names from compilation* field, type a list of library names.

4. Click *OK*.

   When you perform the compile action, the compiler will not compile these libraries.

   Note: You can override this exclusion individually for any module by right-clicking the module on the *Table View* tab in the hierarchy editor and selecting *Compile Netlist* from the pop-up menu that appears.



For information about the table and tree views in the hierarchy editor, see "Table View" and "Tree View" in the "Cadence Hierarchy Editor Overview" of the *Virtuoso Hierarchy Editor User Guide*.

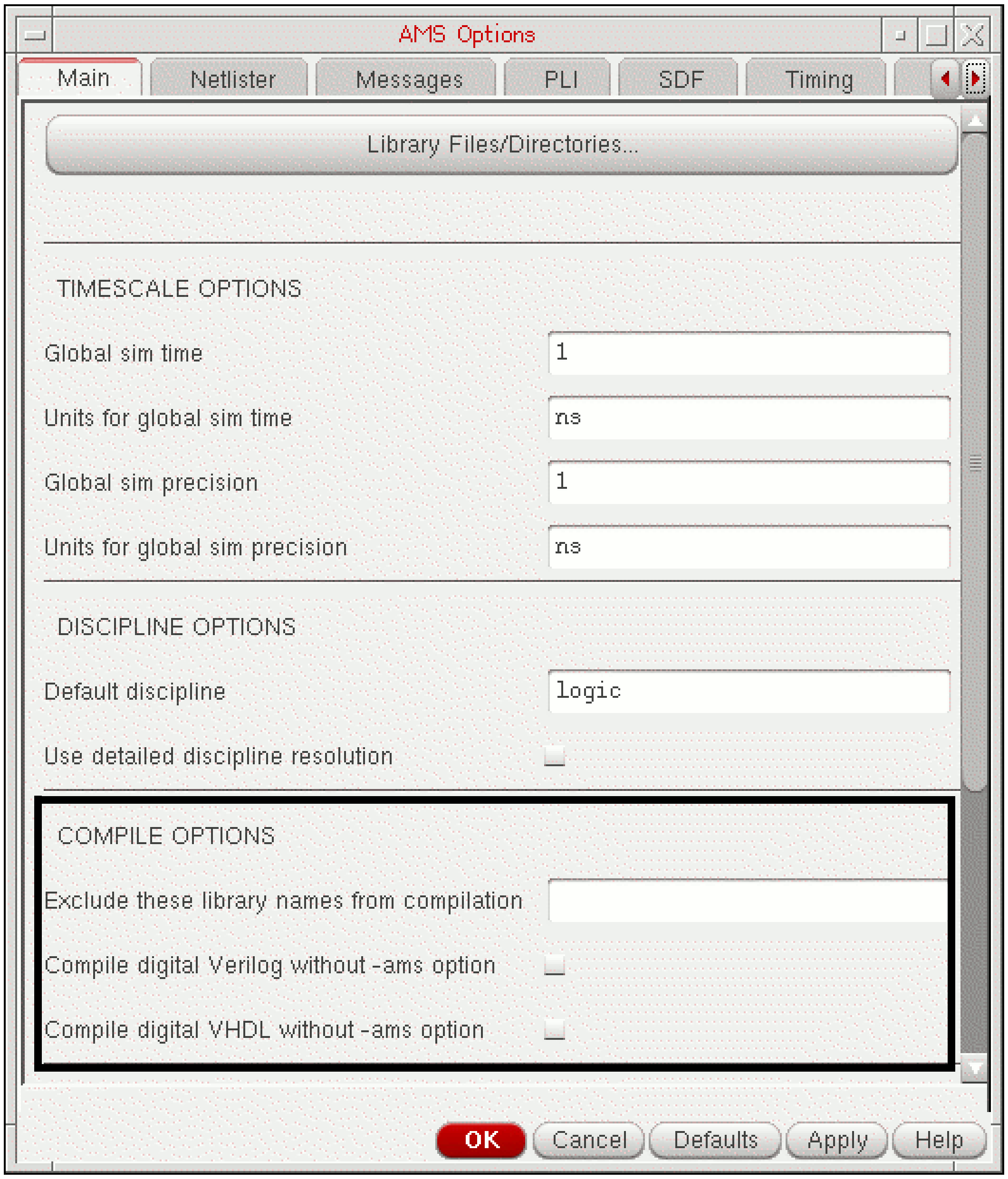
Compiling Digital Verilog without the -ams Option

To compile digital Verilog design units without the `-ams` compiler option, do the following:

1. In the Virtuoso® Hierarchy Editor, choose *AMS – Detailed Setup — AMS Options*.

   The AMS Options form appears.

2. On the *Main* tab, scroll down to the *COMPILE OPTIONS* group box.



TIMESCALE OPTIONS

Global sim time	1
Units for global sim time	ns
Global sim precision	1
Units for global sim precision	ns

DISCIPLINE OPTIONS

Default discipline	logic
Use detailed discipline resolution	☐

COMPILE OPTIONS

Exclude these library names from compilation	
Compile digital Verilog without -ams option	☐
Compile digital VHDL without -ams option	☐

3. Click to mark the *Compile digital Verilog without -ams option* check box.

   The program omits the `-ams` command-line option when compiling files named `verilog.v`. For information about the `-ams` option to the `ncvlog` (compiler) command, see “-AMs Option” under “ncvlog Command Syntax and Options” in the “Compiling” chapter of the *Virtuoso AMS Designer Simulator User Guide*.

   Note: If the program encounters a `verilog.v` file that is actually a link, the decision to use or omit the `-ams` option depends on the extension of the name of the physical file that is the target of the link.

4. Click *OK*.

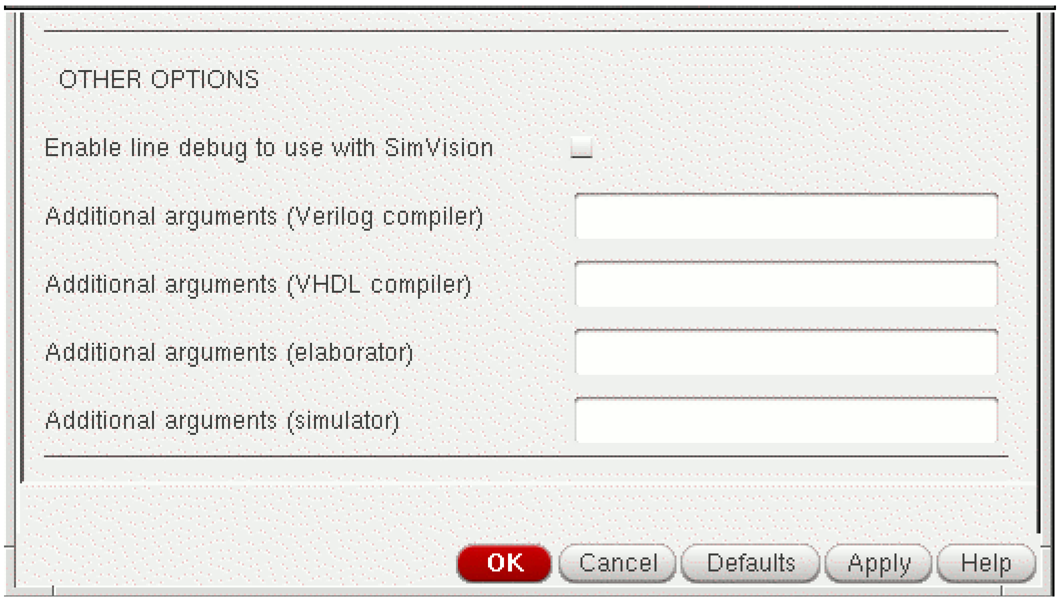
Turning On Line Debug for SimVision

To turn on line breakpoints and the ability to single-step through code in SimVision, do the following:

1. In the Virtuoso® Hierarchy Editor, choose *AMS – Detailed Setup — AMS Options*.

   The AMS Options form appears.

2. On the *Main* tab, scroll down to the *OTHER OPTIONS* group box.



OTHER OPTIONS

Enable line debug to use with SimVision	☐
Additional arguments (Verilog compiler)	
Additional arguments (VHDL compiler)	
Additional arguments (elaborator)	
Additional arguments (simulator)	

3. Click to mark the *Enable line debug to use with SimVision* check box.

4. Click *OK*.

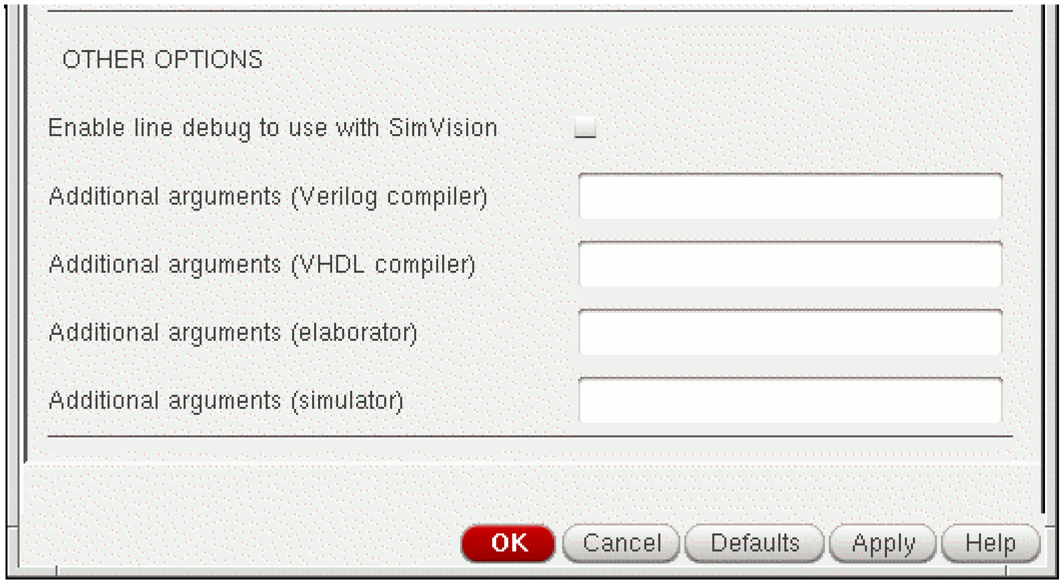
Specifying Additional Verilog Compiler Arguments

To specify additional arguments for the Verilog compiler (`ncvlog`), do the following:

1. In the Virtuoso® Hierarchy Editor, choose *AMS – Detailed Setup — AMS Options*.

   The AMS Options form appears.

2. On the *Main* tab, scroll down to the *OTHER OPTIONS* group box.



OTHER OPTIONS

Enable line debug to use with SimVision	☐
Additional arguments (Verilog compiler)	
Additional arguments (VHDL compiler)	
Additional arguments (elaborator)	
Additional arguments (simulator)	

3. In the *Additional arguments (Verilog compiler)* field, type any additional arguments you want the Verilog compiler to use.

   For information about Verilog compiler options, see “Compiling Verilog Source Files with ncvlog” in *Cadence NC-Verilog Simulator Help*.

Important

You must not specify a `-log` argument because the compiler automatically writes the default log file, `ncvlog.log`, to the run directory (unless you select *No log file*).

4. Click *OK*.

   The Verilog compiler uses the options you specified.

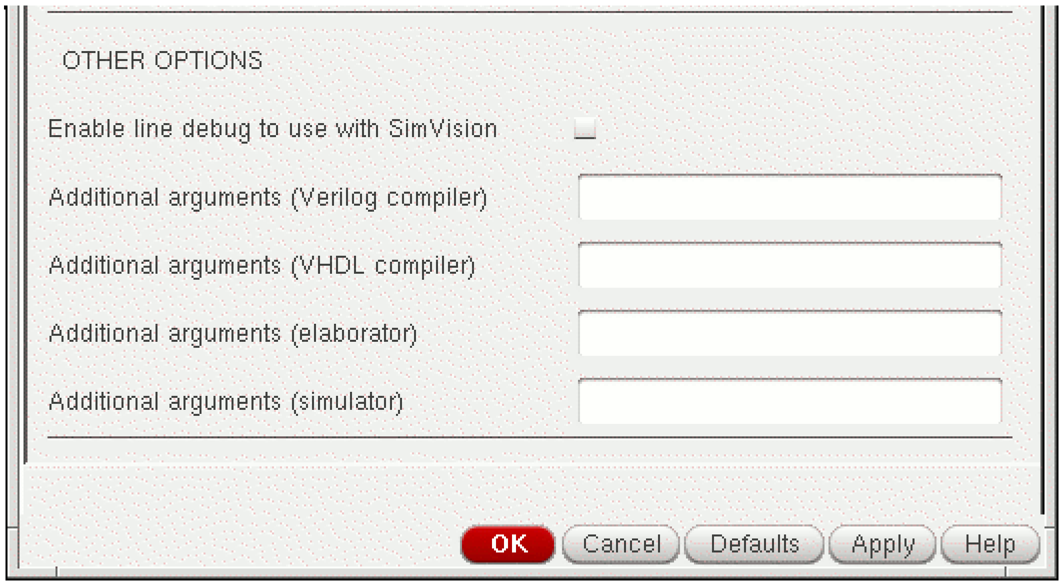
Specifying Additional VHDL Compiler Arguments

To specify additional arguments for the VHDL compiler (`ncvhdl`), do the following:

1. In the Virtuoso® Hierarchy Editor, choose *AMS – Detailed Setup — AMS Options*.

   The AMS Options form appears.

2. On the *Main* tab, scroll down to the *OTHER OPTIONS* group box.



OTHER OPTIONS

Enable line debug to use with SimVision	☐
Additional arguments (Verilog compiler)	
Additional arguments (VHDL compiler)	
Additional arguments (elaborator)	
Additional arguments (simulator)	

3. In the *Additional arguments (VHDL compiler)* field, type any additional arguments you want the VHDL compiler to use.

   For information about VHDL compiler options, see “Compiling VHDL Source Files with ncvhdl” in *Cadence NC-VHDL Simulator Help*.

Important

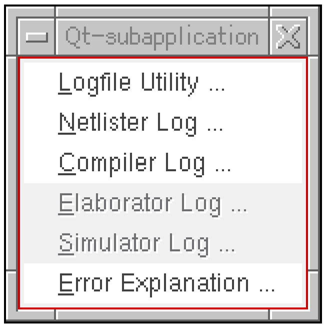
You must not specify a `-log` argument because the compiler automatically writes the default log file, `ncvhdl.log`, to the run directory (unless you select *No log file*).

4. Click *OK*.

   The VHDL compiler uses the options you specified.

14 Viewing Simulation Output

Output from the Virtuoso® AMS Designer simulator consists of simulation log files and plot data. You can view simulation log files and plot data using the *AMS – Plot Results* and *AMS – Log Files* menus.



See the following topics for more information:

- Plotting Results on page 287
- Using the Log File Utility on page 289
- Viewing the Netlister Log File on page 289
- Viewing the Compiler Log File on page 289
- Viewing the Elaborator Log File on page 289
- Viewing the Simulator Log File on page 290
- Viewing Error Explanations on page 290

See also “Listing Compiled Modules” on page 286.

Listing Compiled Modules

You can use the `ncls` utility to query the list of compiled objects:

```
ncls -library library_name
```

If you are using implicit TMP directories—as you are when you run the AMS simulator in the Virtuoso® analog design environment (ADE)—you can use a command like

```
ncls -cds_implicit_tmpdir path_to_dir_with_pak -lib library_name
```

Note: When you run the AMS simulator in ADE, *`path_to_dir_with_pak`* is the path to the *`runDirectory`*`/netlist/ihnl` directory.

For example, if your working library is `amslib`, the command

```
ncls -library amslib
```

might produce a list like the following:

```
ncls: 06.20-s001: ... (c) Copyright 1995-2007 Cadence Design Systems, Inc.
    module amslib.comparator:module (VST)
    module amslib.comparator:module (SIG) <0x5d152f61>
    module amslib.comparator:module (SAM) <0x00000001>
    module amslib.comparator:module (SDB)
    architecture AMSLIB.DACONV:DACONV_BEHAV (AST)
    architecture AMSLIB.DACONV:DACONV_BEHAV (SIG) <0x6210a27d>
    architecture AMSLIB.DACONV:DACONV_BEHAV (COD) <0x6210a27d>
    architecture AMSLIB.DACONV:DACONV_BEHAV (COD)
    module amslib.elect_to_logic:module (VST)
    module amslib.elect_to_logic:module (SIG) <0x3954d83b>
    module amslib.elect_to_logic:module (COD) <0x3954d83b>
    module amslib.elect_to_logic:module (SAM) <0x00000001>
    module amslib.elect_to_logic:module (SDB)
    package AMSLIB.ELECTRICALSYSTEM (AST)
    package AMSLIB.ELECTRICALSYSTEM (COD)
    connect amslib.mixedsignal:connect (VST)
```

Note: For more information on the `ncls` command, see “ncls” in the “Utilities” chapter of *NC-Verilog Simulator Help*.

Plotting Results

You can plot simulation results from the AMS Designer environment either by plotting only the items you selected for plotting or by plotting voltages and currents for nets and terminal instances you select on the schematic.

Note: You specify the waveform viewer on the Quick Setup form or the General Setup form.

To plot the items you selected for plotting in the waveform viewer you specified, do the following:

➤ In the Virtuoso® Hierarchy Editor, choose *AMS – Plot Results – Plot Outputs*.

The items you selected for plotting appear in the waveform viewer you specified.

Refer to the waveform viewer documentation for information about how to use it.

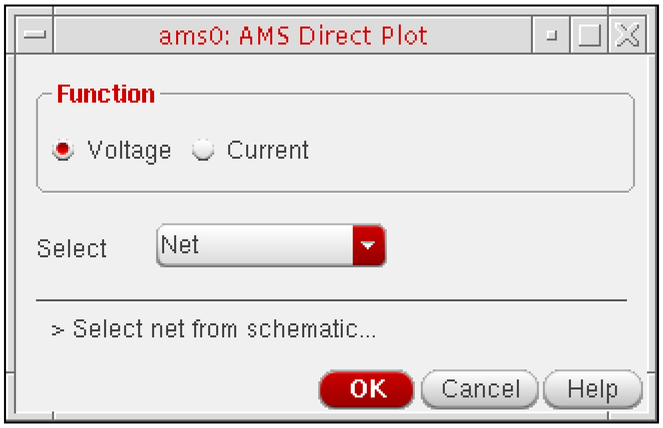
To plot voltages and currents for nets and terminal instances you select on the schematic, in the waveform viewer you specified, do the following:

1. In the Virtuoso® Hierarchy Editor, choose *AMS – Plot Results – Direct Plot*.

   The AMS Direct Plot form appears as well as the schematic and waveform windows.

2. On the AMS Direct Plot form, select either *Voltage* or *Current* from the *Function* group box.

   If you select *Voltage*, *Select net from schematic* appears on the form.


ams0: AMS Direct Plot

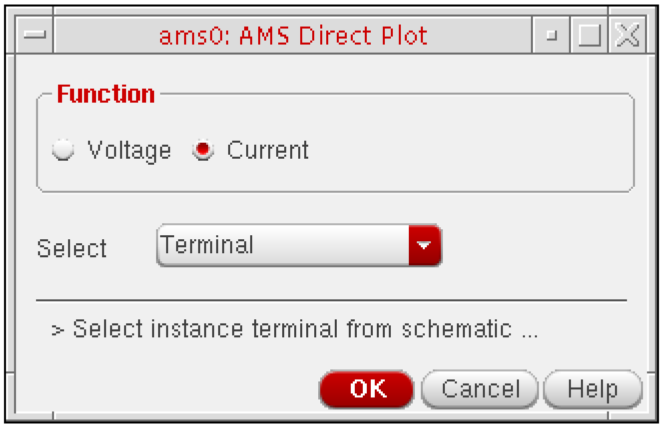
Function

[✓ Voltage] Current

Select: Net

> Select net from schematic...

If you select *Current*, *Select instance terminal from schematic* appears on the form.



Voltage [✓ Current]

Select Terminal

> Select instance terminal from schematic ...

3. Use the *Select* drop-down combo box to specify one of the following:

For *Voltage*:	Select *Net* for the voltage on a single net.
	Select *Differential Nets* for a differential voltage.
	Select positive net from schematic appears on the form.
For *Current*:	Select *Terminal* for the current into a single terminal.
	Select *Differential Terminals* for a differential current.
	Select positive instance terminal from schematic appears on the form.

4. For a single value, simply select either a net (for *Voltage*) or terminal (for *Current*).

   The waveform for the selected item appears in the waveform window.

5. For differential voltage or current, select the two items in succession on the schematic.

   The differential waveform appears in the waveform window.

6. When you are finished using the AMS Direct Plot form, click *OK*.

Using the Log File Utility

Using the log file utility (NCBrowse), you can view and analyze log files interactively.

To use the log file utility, do the following:

➤ In the Virtuoso® Hierarchy Editor, choose *AMS – Log Files – Logfile Utility*.

The NCBrowse window appears.

For information about NCBrowse and how to use it, see the *NCBrowse Message Browser User Guide*.

Viewing the Netlister Log File

To view the netlister log file, do the following:

➤ In the Virtuoso® Hierarchy Editor, choose *AMS – Log Files – Netlister Log*.

The netlister log file (`*runDirectory*/netlist/netlister.log`) appears in a text window.

Viewing the Compiler Log File

To view the compiler log file, do the following:

➤ In the Virtuoso® Hierarchy Editor, choose *AMS – Log Files – Compiler Log*.

The compiler log file (`*runDirectory*/psf/ncvlog.log` and/or `*runDirectory*/psf/ncvhdl.log`) appears in a text window.

Viewing the Elaborator Log File

To view the elaborator log file, do the following:

➤ In the Virtuoso® Hierarchy Editor, choose *AMS – Log Files – Elaborator Log*.

The elaborator log file (`*runDirectory*/psf/ncelab.log`) appears in a text window.

Viewing the Simulator Log File

To view the simulator log file for the AMS simulation, do the following:

➤ In the Virtuoso® Hierarchy Editor, choose *AMS – Log Files – Simulator Log*.

The simulator log file (`runDirectory/psf/ncsim.log`) appears in a text window.

Viewing Error Explanations

To view error explanations for AMS, do the following:

➤ In the Virtuoso® Hierarchy Editor, choose *AMS – Log Files – Error Explanation*.

The Error Explanation form appears.

For more information, see "Viewing the Error Explanation for AMS" in the *Virtuoso Analog Design Environment L User Guide*.

15 Using the amsdesigner Command

The `amsdesigner` command allows you to run AMS Designer from the command line or from a script, using the same default values used by the AMS Designer environment. The command includes options for modifying configurations, netlisting, compiling, elaborating, and simulating.

```
amsdesigner_command ::=
            amsdesigner [-help | -version]
      |     amsdesigner -lib libName -cell cellName -view viewName
                 action_option {action_option} {setup_option}

action_option ::=
      |     -compile all | incremental | whenNetlist
      |     -elaborate
      |     -hier_info [hierFileName]
      |     -netlist all | incremental
      |     -saveconfig '[[libName.]cellName:]viewName'
      |     -savenetlistfiles filePath
      |     -simulate [batch | gui | tcl]

setup_option ::=
            -analogcontrol filePath
      |     -append_log
      |     -cdsglobals overwriteEdits | retainEdits
      |     -cdslib filePath
      |     -globalsignals filePath | signalNames
      |     -hdlvar filePath
      |     -input filePath
      |     -liblist 'libName {libName}'
      |     -log logFileName
      |     -modelincdir modelIncDirs
      |     -modelpath modelPaths
      |     -ncelabOpts options
      |     -ncsimOpts options
      |     -ncvhdlOpts options
      |     -ncvlogOpts options
      |     -netlistToRunDir
      |     -rundir runDirPath
      |     -snapshot snapshotName
      |     -solver spectre | ultrasim
      |     -sourcefile libName cellName viewName 'SPICEorSpectreFilePath'
      |     -useRunDirNetlistsOnly
      |     -verilogfile libName cellName 'VerilogAMSFilePath'
      |     -viewlist 'viewName {viewName}'
```

The following table describes the `amsdesigner` command options and values.

amsdesigner Option and Value	Effect
`-HElp`	Returns a description of the `amsdesigner` command and its options.
`-VERSion`	Returns version information, including the versions of the `amsdesigner`, Virtuoso® Hierarchy Editor, `amsdirect`, and `ncvlog` programs and the versions of input and output files that the Virtuoso Hierarchy Editor uses.
`-LIb` *`libName`*	Specifies the library containing the configuration that you want to process.
`-CEll` *`cellName`*	Specifies the cell containing the configuration that you want to process.
`-VIEW` *`viewName`*	Specifies the cellview name of the configuration that you want to process. The `amsdesigner` command opens this configuration in read-only mode.
`-COmpile`	
`All`	Compiles all cellviews in the configuration, whether netlisted in this run or not.
`Incremental`	Compiles only new or revised netlists.
`Whennetlist`	Compiles only cellviews netlisted in this run.
`-Elaborate`	Elaborates the design.
`-HIer_info` [*`hierFileName`*]	Writes hierarchy information to *`hierFileName`*. Default: `./amsdesigner.hier_info` Relative paths are resolved with respect to the invocation directory. The hierarchy information gives the lib.cell:view of each view used in the design configuration, arranged from lowest-level to highest-level. Primitive instances are not included.

amsdesigner Option and Value	Effect
`-NETLIST`	
`All`	Netlists all cellviews in the configuration, whether they have changed since the previous netlisting or not.
`Incremental`	Generates Verilog®-AMS netlists for new or revised cellviews only.
`-SAVEConfig` `[[`*`libName`*`.]`*`cellName`*`:]`*`viewName`*	
	Specifies that the modified configuration information is to be written to the specified library, cell, and view. If *`libName`* or *`cellName`* are omitted, the values are assumed to be the same as the values for the cellview being opened.
`-SAVENetlistfiles` `'`*`filePath`*`'`	Specifies that the `ncvlog` options used in the run and the list of netlist files used in the configuration are to be written to the specified file. The resulting file can be used as the argument for the `ncverilog -f` option.
	Relative paths are resolved with respect to the invocation directory.
`-SImulate`	
`Batch`	Runs the simulation in the background. This mode, which does not allow you to interact with the simulator, usually simulates more quickly than the other modes.
`Gui`	Opens a graphical interface that allows you to interact with the simulator by using buttons, menus, and Tcl commands.
`Tcl`	Opens a text-based window where you can use the Cadence-supported Tcl commands to interact with the simulator.

amsdesigner Option and Value	Effect
`-ANalogcontrol` *`filePath`*	Specifies the analog simulation control file to be used. Using single quotes for the *`filePath`* argument prevents having included shell variables evaluated by the shell from which the `amsdesigner` command is run. The *`filePath`* argument is passed verbatim to the simulator. If specified, the `-analogcontrol` option can be specified only once. Relative paths are resolved with respect to the run directory.
`-APpend_log`	Appends NC program and AMS netlister log files to the `amsdesigner` log file.
`-CDSGlobals`	You can omit the `-CDSGlobals` option if you have not edited the `cds_globals` module by hand. If you have edited the `cds_globals` module by hand, you must use the `-CDSGlobals` option and you must specify a value for the option.
`Overwriteedits`	Regenerates and overwrites the `cds_globals` module as necessary, even if you have edited the module.
`Retainedits`	Does not overwrite an edited `cds_globals` module. However, the `retainEdits` value allows `amsdesigner` to overwrite the `cds_globals` module if you have *not* edited the module.
`-CDSLib` *`filePath`*	Specifies a cdslib file to load. Default: `./cds.lib` Relative paths are resolved with respect to the invocation directory.
`-Globalsignals`	Specifies a set of global signals, either in a file or as a list of signal names immediately following the option.
`filePath`	Specifies the file containing global signal names, each name on its own line in the file. If you specify a relative path to the file, that path is relative to the run directory.
`signalNames`	Specifies a space-separated list of one or more global signal names. For example: `-globalsignals VSS50! VDD50!`

amsdesigner Option and Value	Effect
`-HDlvar` *`filePath`*	Specifies the name of the `hdl.var` file to be used for compilation, elaboration, and simulation. Using single quotes for the *`filePath`* argument prevents having included shell variables evaluated by the shell from which the `amsdesigner` command is run. The *`filePath`* argument is passed verbatim to the NC software. Relative paths are resolved with respect to the invocation directory.
`-Input` *`filePath`*	Specifies a script file to run at the beginning of the simulation. The `-input` option can be specified more than once. Relative paths are resolved with respect to the run directory.
`-Liblist '`*`libName`* **`{`***`libName`***`}'`**	Specifies the global library list to be used for the configuration.
`-LOg` *`logFileName`*	Writew messages to *`logFileName`*. Default: `./amsdesigner.log` The *`logFileName`* that you specify with this variable interacts with the `CDS_LOG_PATH` environment variable to determine the actual log file name that is used. ■ If *`logFileName`* is an absolute path, the log file is written to *`logFileName`*. ■ If *`logFileName`* is a relative path and ❑ `CDS_LOG_PATH` is null, *`logFileName`* is placed in the current directory. ❑ `CDS_LOG_PATH` is non-null, the value of `CDS_LOG_PATH` is prepended to the *`logFileName`*. The `CDS_LOG_VERSION` environment variable also affects the final name of the log file.

amsdesigner Option and Value	Effect
-MODELIncdir *modelIncDirs*	Specifies the model include directories to be used. The *modelIncDirs* argument is a colon-separated list of directories. The *modelIncDirs* argument is passed verbatim to the simulator. If specified, the `-modelincdir` option can be specified only once. Using single quotes for the *modelIncDirs* argument prevents having included shell variables evaluated by the shell from which the `amsdesigner` command is run.
-MODELPath *modelPaths*	Specifies the model files to be used. The *modelPaths* argument is a colon-separated list of files with library section names enclosed in parentheses. Using single quotes for the modelPaths argument prevents having included shell variables evaluated by the shell from which the amsdesigner command is run. Single quotes should also be used when library sections are specified. Relative paths are resolved with respect to the run directory.
-NCElabOpts '*options***'**	Specifies ncelab options to be passed to the elaborator. The single quotes are required. This option can be specified only once.
-NCSimOpts '*options***'**	Specifies ncsim options to be passed to the simulator. The single quotes are required. This option can be specified only once.
-NCVHdlOpts '*options***'**	Specifies ncvhdl options to be passed to the compiler. The single quotes are required. This option can be specified only once.
-NCVLogOpts '*options***'**	Specifies ncvlog options to be passed to the compiler. The single quotes are required. This option can be specified only once.
-NEtlisttorundir	Specifies that netlist files are to be written to the current run directory.

amsdesigner Option and Value	Effect
-Rundir *runDirPath*	Specifies an existing run directory to use or a non-existing run directory to be created. The `-rundir` option cannot be specified more than once. For more information, see "Using Existing or Creating New Run Directories" on page 298. Relative paths are resolved with respect to the invocation directory.
-SNapshot *snapshotName*	Specifies the lib.cell:view of the snapshot to be created by the elaborator (if the elaborator runs) or used by the simulator. The *snapshotName* argument is passed verbatim. If specified, the `-snapshot` option can be specified only once.
-SOLver	
Spectre	Specifies that the Spectre solver is to be used for simulation.
Ultrasim	Specifies that the UltraSim solver is to be used for simulation.
-SOUrcefile *libName cellName* '*SPICEorSpectreFilePath*'	
	Specifies an HSPICE, SPICE, Spectre, or Verilog-A design block that is to be bound in the configuration to the specified library and cell. For more information, see "Sourcefile Property" in the *Virtuoso AMS Designer Simulator User Guide*.
-Userundirnetlistsonly	Specifies that the netlister is to consider, and, if necessary, update, netlists only from the run directory. Netlists in master libraries are ignored as the netlister determines whether incremental netlisting is needed for any particular object.
-VERIlogfile *libName cellName viewName* '*VerilogAMSFilePath*'	

amsdesigner Option and Value	Effect
	Specifies a Verilog-AMS or Verilog (digital) source file that is to be bound in the configuration to the specified library, cell, and view. Using `verilogfile` to override a view works only when your netlists are written to the run directory. For more information, see "verilogfile Property" in the *Virtuoso AMS Designer Simulator User Guide*.
`-VIEWList '`*`viewName`* **`{`***`viewName`***`}'`**	
	Specifies the global view list to be used for the configuration.

Note that it is possible to specify combinations of action options for the `amsdesigner` command that do not produce usable results. For example, if you specify that netlists are to be generated but not compiled, elaboration fails because the expected new netlists are not found.

Using Existing or Creating New Run Directories

There are three ways to specify the run directory that is required for each run of the `amsdesigner` command.

- Set the `defaultRunDir` `ams.env` variable.
- Associate a run directory with the design configuration, by turning on *Always use this run directory for this configuration* in the AMS Run Directory form of the AMS Designer graphical user interface. For more information, see "Initializing the AMS Designer Environment" on page 77.
- Use the `amsdesigner -rundir` option. This way of specifying the run directory has the highest precedence.

When you specify an existing run directory for the `amsdesigner` command, the settings stored in the run directory are used for the command (unless you override those settings by using `amsdesigner` options). Any command line overrides are in effect only for this run of the `amsdesigner` command and are not stored.

When you specify a run directory that does not exist (or do not specify a run directory), the `amsdesigner` command uses AMS Designer default settings (unless you override those defaults by using `amsdesigner` options) to create a run directory. In this case, any command line overrides are used for this run of the command and are stored in the run directory that the `amsdesigner` command creates.

When the `amsdesigner` command creates a new run directory and you want to run a simulation with that new run directory, be sure that you either pass an analog control file that specifies the transient analysis stop time or that the site-wide transient analysis stop time is set to a value greater than 0.0. You can run the `amsdesigner` command for other purposes without specifying a transient analysis stop time that is greater than 0.0, but simulations do not run.

Examples

The following command netlists, compiles, elaborates, and simulates the whole design.

```
amsdesigner -lib mylib -cell top -view config
    -netlist all -compile all -elaborate -simulate
```

The following command netlists the cellviews in the design that have been revised, then compiles just those newly netlisted cellviews. The design is neither elaborated nor simulated.

```
amsdesigner -lib amsLib -cell top -view config
    -netlist incremental -compile whenNetlist
```

The following command returns the versions of the programs and files that the `amsdesigner` command uses, and then exits. If you need to communicate with Cadence, you might use a command like this to obtain useful background information.

```
amsdesigner -version
```

The returned information includes information about the programs and files that the `amsdesigner` command uses.

```
@(#)$CDS: amsdesigner 5.0.0 07/09/2003 22:25 (cds12107) $
Tool:   cdsHierEditor   05.01.000-b005
Input:  expand.cfg      04.04.003
Input:  expand.cfg      05.00.000
Input:  pc.db   01.00
Output: expand.cfg      05.00.000
Output: Verilog 1364-1995
Output: VHDL    1076-1993
@(#)$CDS: amsdirect version 5.0.0 07/10/2003 15:11 (cds12107) $
ncvlog: v04.00.(s019)
```

The following command specifies a new, non-existing run directory. The new run directory is created and the AMS Designer default settings are used and then stored.

```
amsdesigner -lib VFS_AMS_PHY180_sims -cell aeq_ac_sim -view config_ams -netlist all
-compile all -elaborate -simulate -rundir newrundir
```

The following command specifies that simulation be done in the batch mode. The command also specifies a modelpath.

```
amsdesigner -lib VFS_AMS_PHY180_sims -cell aeq_ac_sim -view config_ams -netlist all
-compile all -elaborate -simulate batch -rundir existingrundir -modelpath '/home/
Usim_vfs_tutorial/spectre_models/gpdk.scs(NN)'
```

The following command overrides the solver that is used in `existingrundir`.

```
amsdesigner -lib VFS_AMS_PHY180_sims -cell aeq_ac_sim -view config_ams -netlist all
-compile all -elaborate -simulate batch -rundir existingrundir -solver ultrasim
```

The following command, among other things, specifies a run directory, specifies a couple of sourcefiles and a verilogfile, saves the configuration, and compiles, elaborates, and simulates the design.

```
amsdesigner -lib training -cell PLL1 -view config_AMS -netlist all -rundir '$CDIR/
test_run' -log amsd11.log -hdlvar '$CDIR/hdl.var' -ncvlogOpts "-nocopy" -
append_log -sourcefile analogLib pmos '$CDIR/models/tranModels.scs' -sourcefile
analogLib nmos '$CDIR/models/tranModels.scs' -saveConfig
'training.PLL1:config_AMS' -compile all -elaborate -simulate batch -verilogfile
'training digital_stimuli module $CDIR/training_cdba_lib/digital_stimuli/verilog/
verilog.v' -netlistToRunDir
```

16

Producing Customized Netlists

You can customize both the format and the content of a netlist that the AMS Designer netlister generates. Cadence provides the following customization aids:

- Procedures that replicate the default behaviors of the netlister
- Procedures that provide lower-level help, such as for printing warnings
- Functions that access the internal data structures that the netlister uses, so you can read and, in some cases, modify the information it stores there

These capabilities provide a range of options for tailoring netlists to meet your needs.

The discussion in this chapter points frequently to the descriptions of the functions in Appendix G, "SKILL Functions Supported for Netlisting."

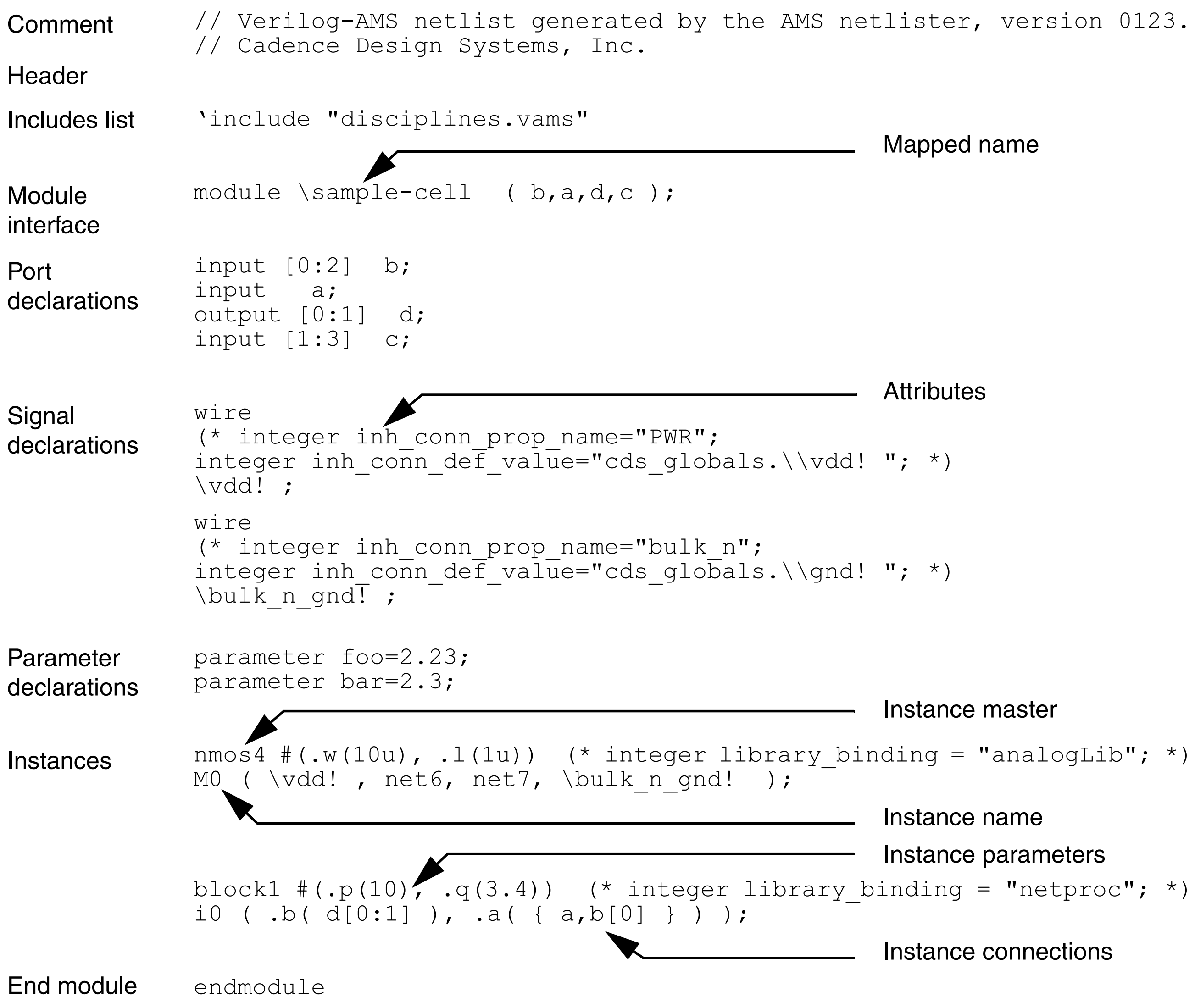
See the following topics for more information:

- Identifying the Sections of a Netlist on page 302
- Using Netlisting Procedures to Customize Netlists on page 303
- Addressing Problems using Customized Netlists on page 313
- Data Objects Supported for Netlisting on page 339

Identifying the Sections of a Netlist

Some of the customizations described in this chapter affect particular sections of the netlist. To establish a common vocabulary, the following illustration points out some of the relevant parts.

Figure 16-1 Sections of a Netlist



```
// Verilog-AMS netlist generated by the AMS netlister, version 0123.
// Cadence Design Systems, Inc.

`include "disciplines.vams"

module \sample-cell  ( b,a,d,c );

input [0:2]  b;
input   a;
output [0:1]  d;
input [1:3]  c;

wire
(* integer inh_conn_prop_name="PWR";
integer inh_conn_def_value="cds_globals.\\vdd! "; *)
\vdd! ;

wire
(* integer inh_conn_prop_name="bulk_n";
integer inh_conn_def_value="cds_globals.\\gnd! "; *)
\bulk_n_gnd! ;

parameter foo=2.23;
parameter bar=2.3;

nmos4 #(.w(10u), .l(1u))  (* integer library_binding = "analogLib"; *)
M0  ( \vdd! , net6, net7, \bulk_n_gnd!  );

block1 #(.p(10), .q(3.4))  (* integer library_binding = "netproc"; *)
i0 ( .b( d[0:1] ), .a( { a,b[0] } ) );

endmodule
```

For a table that lists the netlisting procedures responsible for generating these labeled sections of the netlist, see Table 16-2 on page 341.

Using Netlisting Procedures to Customize Netlists

You can use customized netlist procedures to change the netlists that the AMS netlister generates. However, as you override the default netlisting procedures, you become responsible for tracking items that the default netlisting procedures track automatically. You also become responsible for ensuring that the netlists generated by your procedures compile, elaborate, and simulate correctly.

Writing and Loading Netlisting Procedures

To use netlisting procedures, you define new functions for the AMS netlister, have the new functions recognized by the netlister, and access and change the data used to construct netlists. The next sections describe the operators you use and the steps to follow to accomplish those tasks.

- SKILL Operators on page 303
- Deciding What Kind of Override File to Use on page 305
- Replacing Default Procedures with Custom Procedures on page 307
- Loading an Override File on page 307
- Using Netlisting Procedures for Particular Instance Masters on page 308
- Using Netlisting Procedures to Customize the simInfo Values of Instances on page 309

SKILL Operators

Netlisting procedures are written in SKILL. The SKILL language provides an extensive set of operators and functions, including some functions designed specifically for writing netlisting procedures. These are described in Appendix G, "SKILL Functions Supported for Netlisting.". You can find additional information about the SKILL language in the *SKILL Language Reference* and in the *SKILL Language User Guide*.

For convenience, the table below summarizes the operation of the SKILL arrow operators. These operators are used to access the data used by the AMS netlister and to replace the default netlisting procedures with custom procedures.

`->`

Accesses the value of a property.

This operator is also used to override netlisting procedures. For more information, see "Replacing Default Procedures with Custom Procedures" on page 307.

For example,

```
formatterId->wiresProc
```

returns the name of the wire printing procedure:

```
amsPrintWires
```

`->?`

Returns a list of the fields included in an object.

For example,

```
A_parameterID->?
```

produces the following list:

```
name      string  - name of the parameter (VerilogAMS
                    namespace)
cdfName   string  - CDF name of the parameter
type      string  - type of parameter from ams.env (should
                    be used for parameter decl)
dbType    symbol  - type of parameter from DB/CDF -
                    'int | 'float | 'string | 'ael |
                    'aelNoNum | 'aelNum
value               value of the parameter, actual type
                    is listed in type
isDefault boolean - whether value is the default
                    value specified in base cell
                    CDF (used in instance parameters)
ignore    boolean - whether the parameter is
                    to be ignored for printing
owner   amsobject - the owner (cellview
                   or instance) of the parameter
```

`->??`

Returns a list of the fields included in an object and gives the current value for each of the fields. For example,

```
formatterId ->??
```

returns a list of values that begins like this:

```
(comments "// Verilog-AMS netlist generated by the AMS
netlister, version 5.0.33.118.\n// Cadence Design
Systems, Inc.\n" headers nil ifdefLanguageExtensions nil
useDefparam nil includeFiles nil paramDefVals nil
paramGlobalDefVal nil)
```

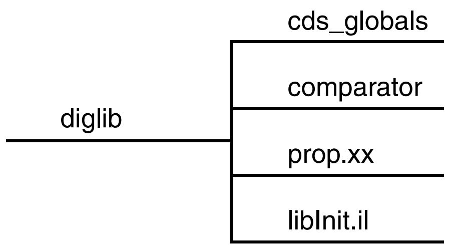
Deciding What Kind of Override File to Use

Typically, the netlisting procedures you write (and, often, the statements that override the default netlisting procedures with your custom procedures) are contained in override files. There are two different kinds of override files; which kind is most appropriate depends on how the netlisting procedures are used.

- Use `libInit.il` files to make netlisting procedures available for the cells in one or more libraries. A file named `libInit.il` is read automatically when the library containing it is used.

  - ❑ If you are defining a procedure that is specific to the cells in a particular library, place the procedure in a `libInit.il` file in the library. This way, the procedure is always available when the library is used, even if the library is copied.

    For example, you have a custom netlisting procedure called `MyCommentsProc`, defined in the file `libInit.il`. You place the `libInit.il` file directly in the library that holds the cells to which the procedure is applied. The library structure might look like this:

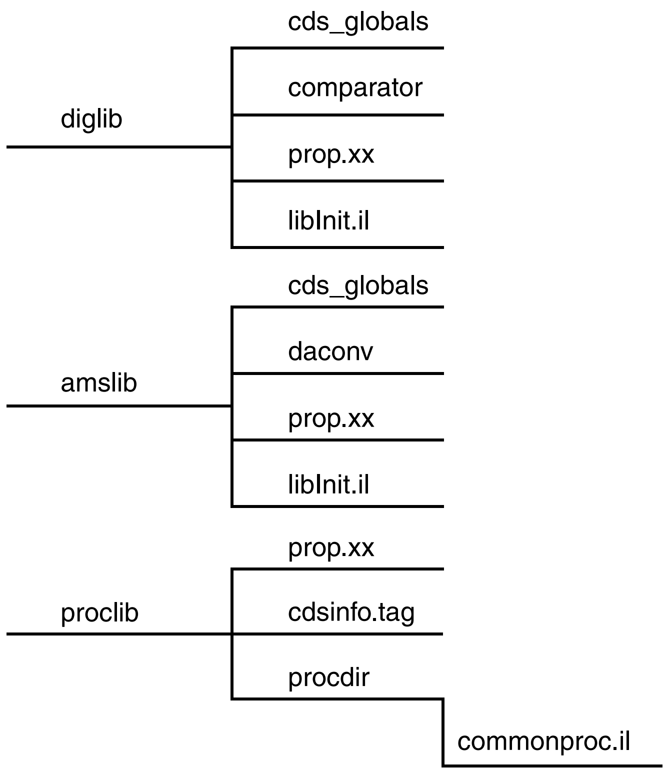
    The `libInit.il` file contains the specification of the custom procedure. For example, to change the default header, the code in the `libInit.il` file might look like this:

```
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 2, 2004.\n")
);defun
```

    When you netlist the `comparator`, the `MyCommentsProc` procedure is always available.

  - ❑ If you are defining a procedure that is applied to cells from different libraries, use a `libInit.il` file in each library but keep the actual procedure code in a separate file that is loaded by each `libInit.il` file.

For example, you have a custom netlisting procedure called `MYcustproc`, defined in the file `commonproc.il`. You place `commonproc.il` in a subdirectory called `procdir` of the library `proclib` and you place a `libInit.il` file in every library that uses the `MYcustproc` procedure. The library structure might look like this:



Each `libInit.il` file contains code like the following to load the custom procedure.

```
load(
    strcat(
        ddGetObjReadPath( ddGetObj( "proclib" ))
        "/procdir/commonproc.il"
    )
)
```

With this approach, when you netlist the `comparator` (from the `diglib` library) or the `daconv` (from the `amslib` library) the `MYcustproc` is available for each.

- If you are defining a procedure that is to be called for every instance in the netlist, use an initFile, as described in "Loading an Override File" on page 307.

Do not use `.cdsinit` files for override procedures because these files are not read when the netlister runs by itself or from the Virtuoso Hierarchy Editor.

It is also possible to type netlisting procedures directly into simulation information (simInfo) fields, in the form of *lambda* functions. With this approach, override files are not needed. For more information, see “Using Netlisting Procedures to Customize the simInfo Values of Instances” on page 309.

Replacing Default Procedures with Custom Procedures

The code in the override file typically does two things:

- It defines one or more custom netlisting procedures.
- It indicates which default procedure is to be replaced by the custom procedure.

For example, the following code first defines a customized replacement for the default procedure that prints comments in netlists. Then the code tells the netlister which default netlisting procedure is to be overridden by the custom procedure.

The `->` operator in this example is used to change the value of a field, and, in this case, has the effect of overriding the default `commentsProc` netlisting procedure with the customized procedure, `MYCommentNetProc`.

```
;; All of this code goes in the initFile.
(defun MYCommentNetProc ( formatterId cellViewId)
   (amsPrintComments formatterId cellViewId)
   (amsPrint formatterId “// Printing my own comments\n”)
) ; defun
;; Get the ID of the netlister
netId = amsGetNetlister()
;; Get the ID of the formatter
formatterID = netId->vlog
;; Override the default procedure to print comments
formatterID->commentsProc = ‘MYCommentNetProc
```

Loading an Override File

Before the AMS netlister can use a customized netlisting procedure that is stored in an override file, you must make the new procedure available to the netlister. There are three ways to do that:

- Use `libInit.il` files, as described in “Deciding What Kind of Override File to Use” on page 305. These files are loaded automatically when the design uses a component from the library.
- Load the override file from the CIW, using a statement like

  ```
  load( "custom.il" )
  ```

  As you are preparing the netlisting procedures override file, you might need to make more than one attempt before the file loads and runs without errors. If you use the `load` command to load a changed override file, you can avoid restarting AMS Designer after each change. This way of loading the override file works only if you are netlisting from the CIW with the AMS Netlister form.

Specifying When Netlisting Procedures Are Used

Normally, custom netlisting procedures are in effect globally, affecting every netlist and every netlisted instance. However, with the approaches described below, you can apply custom netlisting procedures to particular instance masters or to particular instances.

Using Netlisting Procedures for Particular Instance Masters

You can specify a custom procedure in the simInfo *netlistProcedure* field of instance masters. With this approach, you can use different custom netlisting procedures when netlisting instances of different master cells. All instances of that master then use the custom procedure for netlisting, but instances of a different master can use a different custom procedure or continue to use the default netlisting procedure. This gives you the flexibility to use, for example, a custom procedure called `MYPrintResistorInstance` for resistors and a different custom procedure called `MYPrintCapacitorInstance` for capacitors.

To use this capability,

1. Define and load the customized netlisting procedures to be used for netlisting the instance master. As discussed in "Deciding What Kind of Override File to Use" on page 305, the appropriate file is a `libInit.il` file.
2. From the CIW, choose *Tools – CDF – Edit*.

   The Edit Component CDF form appears.
3. Fill in the *Library Name* and *Cell Name* of the instance master.
4. In the *Simulation Information* section, click *Edit*.

   The Edit Simulation Information form appears.
5. In the *CDF Type* field, choose *Base*.
6. From the *Choose Simulator* pulldown, choose *ams*.
7. In the *netlistProcedure* field, specify the custom procedure or procedures.

   The function information for the *netlistProcedure* field can take either of two forms:

   ❑ The name of an instance function that overrides the `instanceProc` procedure for this instance master

- ❑ A SKILL disembodied property list (DPL) with the following syntax:

  ```
  nil
  [masterName instancemasterNameFunc]
  [params instanceparamsFunc]
  [ports instanceportsFunc]
  ```

  The keywords in the syntax indicate which netlisting procedures are overridden when this instance master is netlisted. The keywords correspond to, in order, the following procedures: `instanceMasterNameProc`, `instanceParametersProc`, `instancePortsProc`.

  For example,

  ```
  MYPrintInstance
  ```

  in the *netlistProcedure* field is equivalent, for this instance master only, to

  ```
  vlog->instanceProc = 'MYPrintInstance
  ```

  To use the DPL syntax,

  ```
  nil masterName MYPrintInstMasterName params MYPrintInstParams
  ```

  in the *netlistProcedure* field is equivalent, for this instance master only, to

  ```
  vlog->instanceMasterNameProc = 'MYPrintInstMasterName
  vlog->instanceParametersProc = 'MYPrintInstParams
  ```

8. In the Edit Simulation Information form, click *OK*.
9. In the Edit Component CDF form, click *OK*.

The AMS netlister uses the custom netlist procedure or procedures that you specify in step 7 whenever an instance of the master cell is written to the netlist. If there is a global override of the same netlisting procedure, the custom procedures specified in the *netlistProcedure* field take precedence for this instance master.

Using Netlisting Procedures to Customize the simInfo Values of Instances

You can specify a custom netlisting procedure in any of the AMS simInfo fields of instance masters. Each time the master is instantiated, the netlister uses the passed-in instance ID to evaluate the custom netlisting procedure. The output of the procedure is used as the value of that simInfo field for that instance. This capability allows you, for example, to change the number of inputs for a nand gate depending on the value of an instance property set by a pcell.

There are two ways to specify a custom netlisting procedure in a simInfo field: enter the procedure directly into the field, or enter the name of a function defined elsewhere.

To use this capability

1. From the CIW, choose *Tools – CDF – Edit*.

   The Edit Component CDF form appears.

2. Fill in the *Library Name* and *Cell Name* of the instance master.
3. In the *Simulation Information* section of the Edit Component CDF form, click *Edit*.

   The Edit Simulation Information form appears.

4. Select *Base* in the *CDF Type* field.
5. From the *Choose Simulator* pulldown, select *ams*.
6. Either enter the procedure directly into the simInfo field, using a lambda function, or enter the name of a function defined elsewhere.

   ❑ To use a lambda function, type into the field of interest a list of values where the first element is `lambda` and the second element is the procedure. The procedure must be entered without comments and without using new-line characters. The instance ID is the only argument that is passed to the lambda function.

   For example, you might enter the following into the *componentName* field of an instance master, all on one line.

   ```
   (lambda (inst) (if (null inst->nmos4var2) inst->cellName inst->nmos4var2))
   ```

   This function checks each instance of the instance master, looking for a property called `nmos4var2`. If the property does not exist on that instance, the instantiation statement for the instance refers to the normal instance master. If the property exists, the instantiation is written so that the instance master normally referenced by the instantiation statement is replaced by an instance master named `nmos4var2`.

   ❑ To enter the name of a function, type in a list of values where the first element is `FUNCTION` and the second element is the name of a procedure already defined in a loaded override file. For example, you might enter the following into the *termOrder* field.

   ```
   FUNCTION MYNumNodeFunction arg1
   ```

   The instance ID is always passed to the function. The third and following elements in the list are optional. If present, they constitute additional arguments to be passed to the function.

7. Click *OK* in the Edit Simulation Information form.
8. Click *OK* in the Edit Component CDF form.

The AMS netlister uses the custom netlist procedure that you specify in step 6 to calculate the effective value of the simInfo field whenever an instance of the master cell is written to the netlist.

Choosing the Best Customization Approach

See the following topics:

- Simpler Approach: Changing Object Values and Using Default Procedures on page 311
- More Powerful Approach: Using Fully Customized Netlisting Procedures on page 312

Simpler Approach: Changing Object Values and Using Default Procedures

AMS Designer uses internal databases to store the information needed to create netlists. The information in the databases is organized into objects, each of which has an associated ID that provides a convenient way of referring to the object. The objects contain fields, each holding a specific kind of information. For more information about the supported objects, see "Netlister Object" on page 339.

You can modify the netlists that the AMS netlister produces by

1. Changing the values of fields in the appropriate objects and then
2. Using the default netlisting procedures to create a netlist that incorporates those changed values.

For example, you can use the following approach to change the value assigned to the `width` parameter:

Without any netlisting procedure overrides, the AMS netlister generates the following netlist. Notice the value of the `width` parameter.

```
// Verilog-AMS netlist generated by the AMS netlister, version none.
// Cadence Design Systems, Inc.
`include "disciplines.vams"
module pwr_supply ( in1,out1 );
input   in1;
output   out1;
parameter width=5u;
parameter length=3.4u;
inductor #(.l(100.0m))  (* integer library_binding = "analogLib";  *)
L0 ( net36, out1 );
resistor #(.r(5))  (* integer library_binding = "analogLib";  *) R0 (
net31, in1 );
```

```
endmodule
```

To change the value assigned to the `width` parameter, you enter the following code in your netlist procedures override file. This code iterates through each of the parameters of the cellview, searching for a parameter named `width`. When it finds such a parameter, setting `cellview_params = nil` ends the search.

```
;; Changes value of "width" and calls default amsPrintParameters function.
(defun MYPrintParameters (formatterId cvId)
cellview_params = cvId->parameters
;; Change the value for the parameter named "width"
   (foreach param cellview_params
       (if param->name == "width" then
          param->value = "5 * sin(10)"
          cellview_params = nil
       )
   );; foreach
;; call the default print parameters function.
   amsPrintParameters(formatterId cvId)
);; defun
;; Set up the custom netlist procedure
   netId = amsGetNetlister()
   vlog = netId->vlog
   vlog->parametersProc = `MYPrintParameters
```

Using this override, the AMS netlister generates the following netlist. The value of the `width` parameter is now set to `5 * sin(10)`.

```
// Verilog-AMS netlist generated by the AMS netlister, version none.
// Cadence Design Systems, Inc.
`include "disciplines.vams"
module pwr_supply ( in1,out1 );
input   in1;
output   out1;
parameter width=5 * sin(10);
parameter length=3.4u;
inductor #(.l(100.0m))  (* integer library_binding = "analogLib";  *)
L0 ( net36, out1 );
resistor #(.r(5))  (* integer library_binding = "analogLib";  *)
R0 ( net31, in1 );
endmodule
```

More Powerful Approach: Using Fully Customized Netlisting Procedures

The full-custom use model builds on the capability described in the previous section, expanding beyond resetting the value of fields to completely rewriting the netlisting procedures. The major difference in the fully customized approach is that the helper netlisting functions are used much more extensively to fine tune the operation of the netlister. This is the most powerful form of custom netlisting procedures because it allows you to add new components, ports, or wires as necessary. While taking advantage of this flexibility, you must account for the various environment variables and format the netlist accordingly. And, of course, you must ensure that the resulting netlist is syntactically correct.

Much of the information you need for full-blown custom netlisting is described earlier in the chapter, including the SKILL operators used to access and change the fields in the data objects and the mechanism for overriding the default netlisting procedures. Detailed information about the data objects can be found in "Netlister Object" on page 339.

Addressing Problems using Customized Netlists

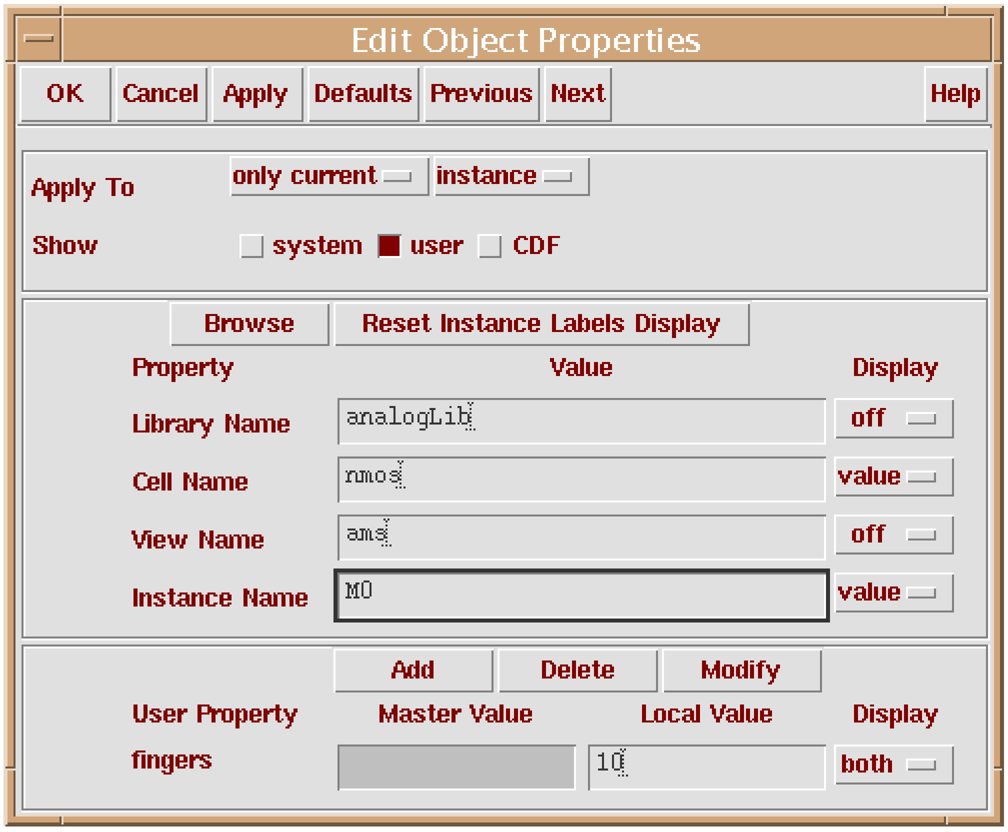
The flexibility of customized netlisting procedures lends itself to addressing a wide range of problems. The extended examples in the following sections illustrate how you can resolve various problems using netlisting procedures.

- Adjusting Parameter Values to Account for Number of Fingers on page 313
- Using Symbols that Represent Verilog Test Code on page 317
- Using CDF Instance Parameters to Define Inherited Connections on page 321
- Netlisting Schematic Parameterized Cells (Pcells) on page 326

Adjusting Parameter Values to Account for Number of Fingers

This example passes the number of fingers on a mosfet to the simulator so that the narrow width effect can be accounted for during simulation. The netlist has two nmos transistors with the same width and multiplier but the `M0` instance has 10 fingers, while the `M1` instance has 1 finger. The same models are used for both pre- and post-layout, so the scaling needed to adjust for the differing numbers of fingers cannot be done in the models; it must be done during netlisting.

The number of fingers for each instance is specified as a user property in the Edit Object Properties form. For instance, the form for instance `M0` looks like this.



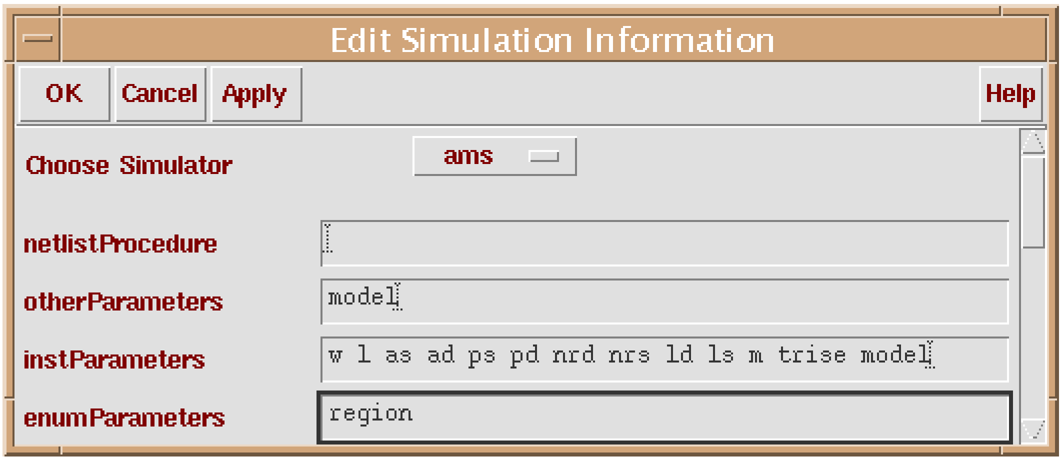
Apply To: only current, instance

Show: ☐ system ■ user ☐ CDF

Property	Value	Display
Library Name	analogLib	off
Cell Name	nmos	value
View Name	ams	off
Instance Name	M0	value

User Property	Master Value	Local Value	Display
fingers		10	both

The parameters that must be scaled are `w`, `as`, `ad`, `ps`, `pd` and `m`. For the approach taken in this example to work, these parameters have to appear in the *instParameters* field when *ams* appears in the *Choose Simulator* field on the Edit Simulation Information form.



Choose Simulator	ams
netlistProcedure	
otherParameters	model
instParameters	w l as ad ps pd nrd nrs ld ls m trise model
enumParameters	region

Notice, however, that the `fingers` property does not appear in the *instParameters* field. There is no need for it there because, although it is used as data for the netlist, it does not appear in the netlist.

The netlisting procedure looks at each parameter. If the current parameter is `w`, `as`, `ad`, `ps`, `pd`, or `m`, the parameter is scaled, either by dividing by the number of fingers or by multiplying by the number of fingers. After the new parameter values are set, the netlisting procedure calls the default instance parameter printing function, `amsPrintInstanceParameters`, to finish the work.

```
;; ======================================================================
;; A custom netlist procedure to compute instance parameters based
;; on number of fingers on the instance
;; ======================================================================
(defun MYPrintInstanceParameters (formatterId cvId instanceId)

  ;; Modify the parameters for "nmos" in a customized way -
  ;; based on number of fingers.
  ;;
  (if (instanceId->masterName == "nmos") then
      ;; We want to change the value of parameters w, as, ad, ps, pd and m
      ;; depending upon the value of property <fingers>.

      ;; A few notes regarding the simInfo:
      ;;
      ;; Please make sure that w, as, ad, ps, pd and m are in the
      ;; instParameters section of ams simInfo. Otherwise, amsdirect does
      ;; not pick them up for writing to the netlist.
      ;;
      ;; Please note that, "fingerwidth" should NOT be an instParameter, as
      ;; it is not required to be printed as a parameter on the instance.
      ;; Because it is not in the include list, amsdirect does not pick
      ;; it up. But it is there on the instance as a normal cellview
      ;; property. Grab it from the instance.
      numfingers = instanceId->id->fingers
      (if (numfingers != nil) then
        ;; Go through the list of parameters for nmos and modify
        ;; the value for the following parameters:
        ;; w, as, ad, ps, pd, m.
        ;;
        (foreach param instanceId->parameters
             (if ((param->name == "w")  ||
                  (param->name == "as") ||
                  (param->name == "ad") ||
                  (param->name == "ps") ||
                  (param->name == "pd") ) then
                param->value = strcat(param->value "/" numfingers)
              ); if
             (if (param->name == "m") then
                param->value = strcat(param->value "*" numfingers)
             ); if
        );foreach
      );if
  ); if
  /* Call the default instance parameters print function */
  amsPrintInstanceParameters(formatterId cvId instanceId)
); defun
;; ==========================================================================
;; Set up area
;; ==========================================================================
netlisterId = amsGetNetlister()
formatterId = netlisterId->vlog
;; Override the printing of instance parameters netlist procedure
formatterId->instanceParametersProc = 'MYPrintInstanceParameters
```

Running this netlist procedure results in a netlist that includes the following instantiation statements.

```
nmos #(.ps(1.268u), .as(2.04E-12), .l(130.0n), .pd(12.68u), .ad(2.04E-12),
 .w(6u), .m(2))
 (* integer library_binding = "analogLib";
    integer passed_mfactor = "m"; *)
  M1 ( out_y, net20, cds_globals.„nd! , •ulk_n_gnd!  );
nmos #(.ps(11.6u/10), .as(1.32E-12/10), .l(130.0n), .pd(9.8u/10), .ad(1.14E-12/10
),
 .w(6u/10), .m(2*10))
 (* integer library_binding = "analogLib";
    integer passed_mfactor = "m"; *)
  M0 ( net20, net9, cds_globals.„nd! , •ulk_n_gnd!  );
```

Notice how in the second instantiation (for `M0`) the values for `ps`, `as`, `pd`, `ad`, and `w` are divided by 10, while the value for `m` is multiplied by 10.

Using Symbols that Represent Verilog Test Code

This example places a symbol representing a piece of Verilog test code on a schematic, and inserts the test code into the netlist. The symbol, which has no pins, is just a vehicle for the test code.

To set up for this approach,

1. Prepare a cellview that contains the test code.

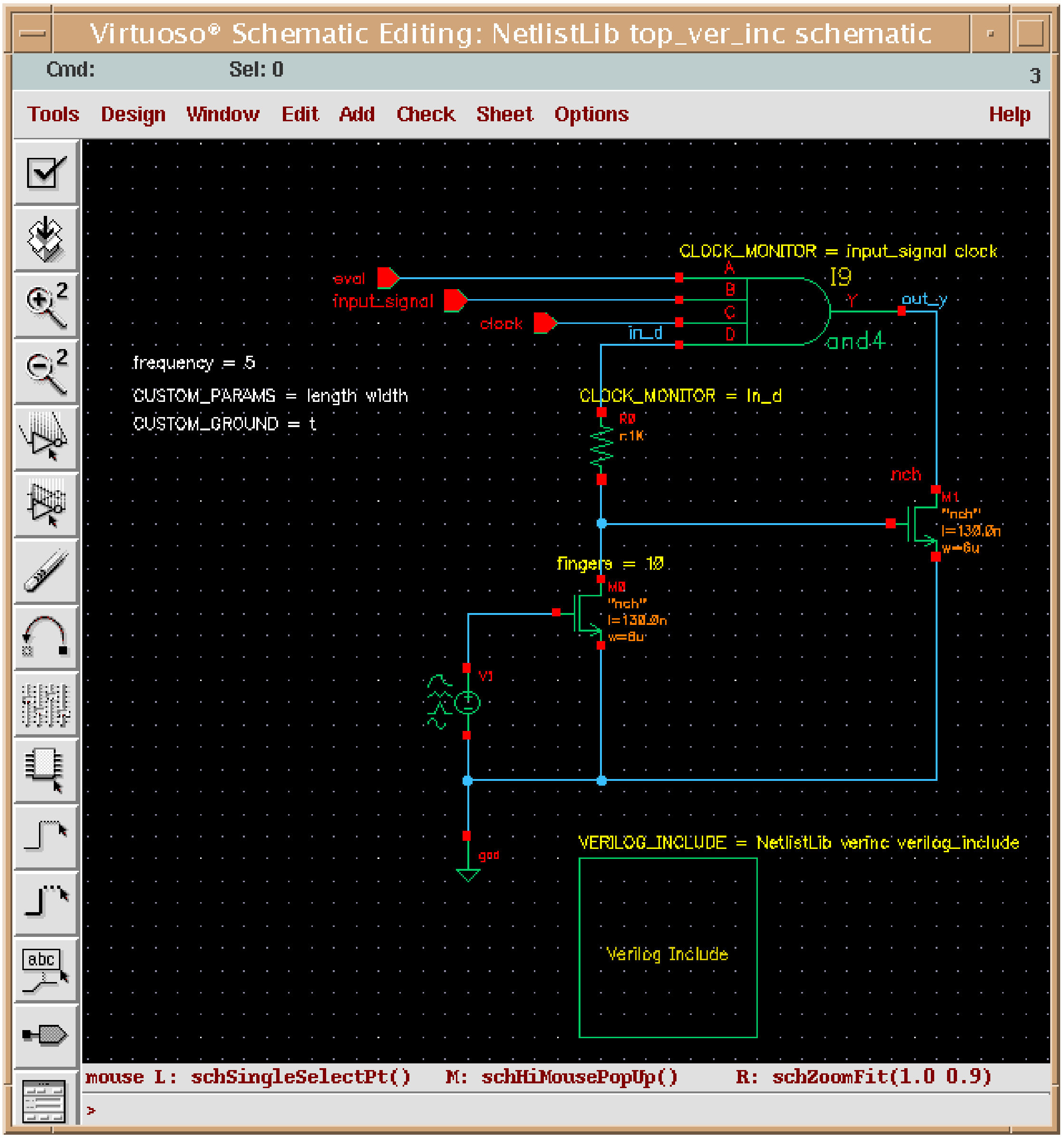
   For this example, assume that the full name of the cellview is `NetlistLib.verinc:verilog_include` and that the following test code is in a file called `verilog.v` in that cellview.

   ```
   // --- begin included file ---
   // Design debugger/monitor
   parameter TCOff=0;

   `ifdef CHECK_INPUT_TRANSITIONS

   always @(posedge(TCOff||in_d) or negedge(TCOff||in_d))
      if (eval==1'b1)
         $display($stime," WARNING: %m in_d transition (evaluate is active)");

   always @(posedge(TCOff||out_y) or negedge(TCOff||out_y))
      if (eval==1'b1)
         $display($stime," INFO: %m out_y transition (evaluate is active)");

   `endif
   // ---- end of included file ----
   ```

   Ultimately, this code is written into the netlist so that if the `CHECK_INPUT_TRANSITION` variable is set, the code checks the transitions.

2. Create a symbol and place it in the schematic whose netlist is to contain the test code.

In the following schematic, for example, notice the square symbol labeled `Verilog Include`.

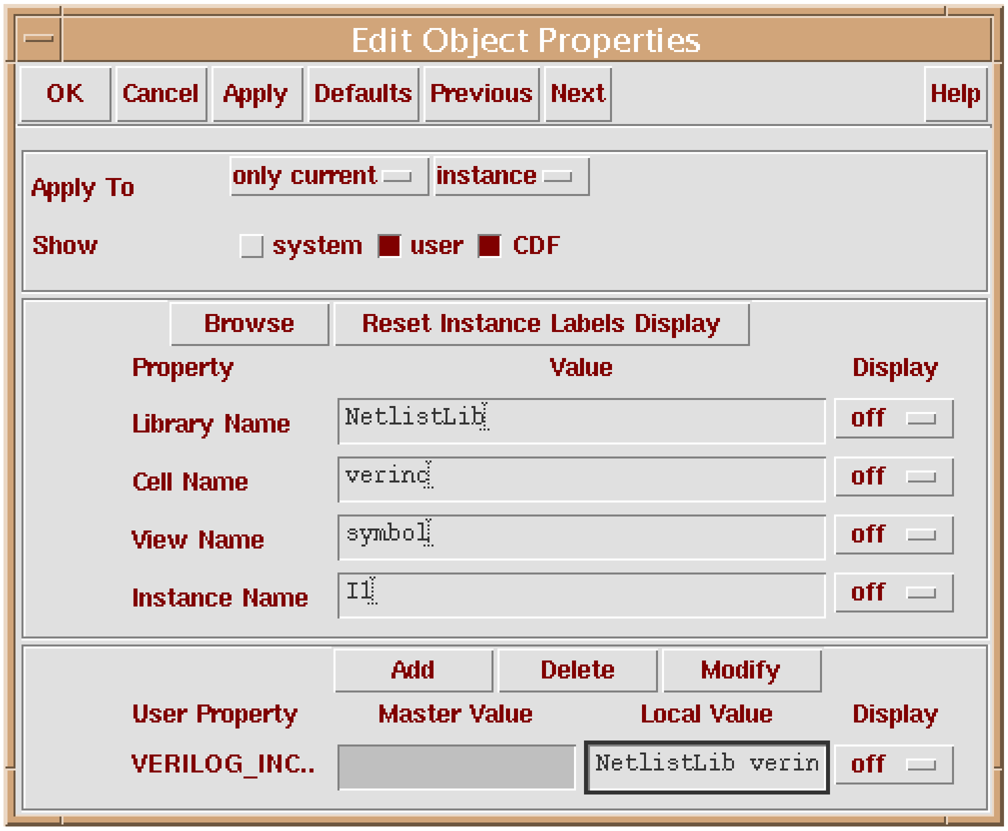

Virtuoso® Schematic Editing: NetlistLib top_ver_inc schematic
Cmd:
Sel: 0
3
Tools
Design
Window
Edit
Add
Check
Sheet
Options
Help
CLOCK_MONITOR = input_signal clock
I9
eval
input_signal
clock
in_d
out_y
and4
frequency = 5
CUSTOM_PARAMS = length width
CUSTOM_GROUND = t
CLOCK_MONITOR = in_d
R0
r:1K
nch
M1
fingers = 10
M0
V1
gnd
VERILOG_INCLUDE = NetlistLib verinc verilog_include
Verilog Include
mouse L: schSingleSelectPt()   M: schHiMousePopUp()   R: schZoomFit(1.0 0.9)

3. Select the placed symbol and open the Edit Object Properties form for it.
4. Add a User Property called `VERILOG_INCLUDE` with a value that indicates the full name of the cellview that contains the test code.

   In step 1, the code was placed in the cellview `NetlistLib.verinc:verilog_include`, so in this step enter the corresponding value `NetlistLib verinc verilog_include`, leaving out the punctuation.

5. Click *Apply*.

The form looks like this.



Apply To: only current, instance

Show: [ ] system [✓ user] [✓ CDF]

Property	Value	Display
Library Name	NetlistLib	off
Cell Name	verinc	off
View Name	symbol	off
Instance Name	I1	off

User Property	Master Value	Local Value	Display
VERILOG_INC..		NetlistLib verin	off

6. Create the override file.

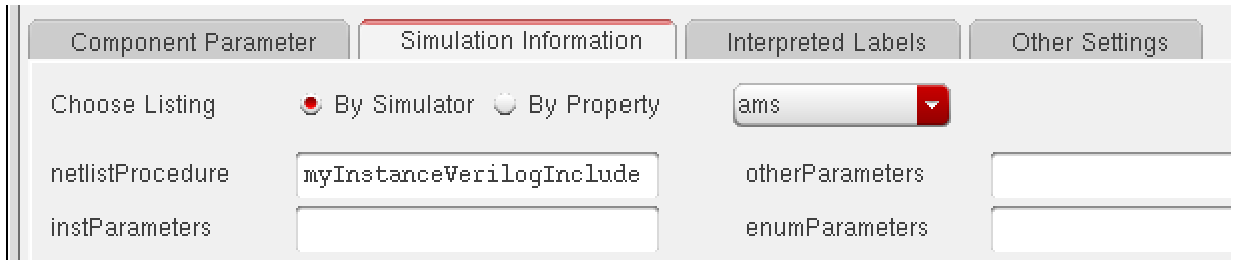
The override file is associated with only the `verinc` cell, so the appropriate override file to use is a `libInit.il` in the `NetlistLib` library.

```
;; =====================================================================
;; A custom netlist procedure for macro substitution.
;; =====================================================================
(defun MYInstanceVerilogInclude (formatter cellview inst)
  (let (lcv file)
     ;; Check for property VERILOG_INCLUDE on the instance
     ;;
     (when inst->id->VERILOG_INCLUDE
         (setq lcv (parseString inst->id->VERILOG_INCLUDE))
         ;; Get the the default file name verilog.v
         (setq file (ddGetObj (car lcv) (cadr lcv) (caddr lcv) "verilog.v"))
         ;; Read in the file and print the contents
         (if (null file)
             (progn
               sprintf(errmsg "Expected a verilog.v in %s:%s.%s\n"
                        (car lcv) (cadr lcv) (caddr lcv) )
```

```
                    (amsError formatter errmsg)
                  )
                  (prog (filePort lineBuffer)
                      ;; Open the file, and start printing contents of the file
                      (setq filePort (infile file->readPath))
                      (while (gets lineBuffer filePort)
                        (amsPrint formatter lineBuffer)
                      ) ; while
                      (close filePort)
                  ) ; prog
              ) ; if
         ) ; when
      ) ; let
    ) ; defun
```

7. Open the Edit Component CDF form for the `netlistLib.verinc` cell.
8. Open the Edit Simulation Information form.
9. Choose the *ams* simulator.
10. In the *netlistProcedure* field, add the name of the overriding netlisting procedure.

    The form looks like this:



Choose Listing [✓ By Simulator] By Property ams
netlistProcedure myInstanceVerilogInclude
otherParameters
instParameters
enumParameters

11. In the Edit Component CDF form, click *OK*.
12. Netlist the schematic.

    The generated netlist includes the checking code that you specified in step 1. An excerpt from the netlist looks like this:

```
nmos #(.ps(1.268u), .as(2.04E-12), .l(130.0n), .pd(12.68u), .ad(2.04E-12),
 .w(6u), .m(2))
 (* integer library_binding = "analogLib";
    integer passed_mfactor = "m"; *)
```

```
  M1 ( out_y, net20, cds_globals.„nd! , •ulk_n_gnd!  );
nmos #(.ps(11.6u/10), .as(1.32E-12/10), .l(130.0n), .pd(9.8u/10), .ad(1.14E-
12/10),
 .w(6u/10), .m(2*10))
 (* integer library_binding = "analogLib";
    integer passed_mfactor = "m"; *)
  M0 ( net20, net9, cds_globals.„nd! , •ulk_n_gnd!  );
// --- begin included file ---
// Design debugger/monitor

parameter TCOff=0;
‘ifdef CHECK_INPUT_TRANSITIONS

always @(posedge(TCOff||in_d) or negedge(TCOff||in_d))

   if (eval==1’b1)

      $display($stime," WARNING: %m in_d transition (evaluate is active)");

always @(posedge(TCOff||out_y) or negedge(TCOff||out_y))

   if (eval==1’b1)

      $display($stime," INFO: %m out_y transition (evaluate is active)");

‘endif
// ---- end of included file ----
```

Using CDF Instance Parameters to Define Inherited Connections

This example establishes inherited connections on programmable nodes so that the names and values of the inherited connections are calculated from object properties and can vary for each instance. This allows each of the instances within a single schematic to have different inherited connections.

For example, assume that you have an instance, `M1`, and that the *Substrate connection* field of the Edit Object Properties form for that instance contains the value `sub!`. That value needs to result in an inherited connection definition, like this.

```
wire
   (* integer inh_conn_prop_name="\\sub! ";
      integer inh_conn_def_value="cds_globals.\\sub! "; *)
  \sub!_sub! ;
```

The inherited connection is to be used to instantiate the instance, like this.

```
ns3v025d #(.as(4.64E-12), .ps(1.664E-05), .m(3), .ad(4.64E-12),
       .pd(1.664E-05),.l(3.2E-07), .w(1E-05))
 (* integer library_binding = "cmos025";
    integer passed_mfactor = "m"; *)
 M1  ( D1, G, S, Bulk, Dnw, \sub!_sub!  );
```

These results can be achieved by using a netlisting function to build, from the instance information contained in the Edit Object Properties form, an appropriate list for the *extraTerminals* field of the Edit Simulation Information form. The netlister then recalculates and uses that list to construct the necessary instantiation statements for each programmable node of each instance.

To implement this technique,

1. Determine what object properties are to be used as input to the netlisting function.

   The component being netlisted for this example is a mosfet, which has three programmable nodes. By examining the Edit Component CDF form for the cell, you find that the programmable nodes are: `bulkNode`, `sub_node`, and `dnw_node`. These are the names you use in the netlisting function, as described in the next step.

2. Prepare the netlisting function.

   The function must generate lists that are appropriate for the *extraTerminals* field of the cell CDF, using as input the object property values associated with each instance. If no object property values are specified, the function needs to create default output.

   The following function is one example that meets these requirements.

```
(defun AMSnmos_dnw_inhExtraTerminals (inst "g")
 (let
   ;; Default values
   ( (term1 '(nil name "B" direction "inputOutput" netExpr))
     (netExpr1 "[@vbulk_n:%:vssa!]")
     (term2 '(nil name "DNW" direction "inputOutput" netExpr))
     (netExpr2 "[@vdnw:%:not_set!]")
     (term3 '(nil name "SUB" direction "inputOutput" netExpr))
     (netExpr3 "[@vsub:%:not_set!]")
   )
   ;; Override values, if any.
   (if (inst != nil) then
     (if (inst->bulkNode != "") then
      netExpr1 = (strcat "[@" inst->bulkNode ":%:" inst->bulkNode "]")
     )
     (if (inst->dnw_node != "") then
      netExpr2 = (strcat "[@" inst->dnw_node ":%:" inst->dnw_node "]")
     )
     (if (inst->sub_node != "") then
      netExpr3 = (strcat "[@" inst->sub_node ":%:" inst->sub_node "]")
     )
   )
   ;; Generate the dynamic "extraTerminals" list.
   extraTerminals = (list (append1 term1 netExpr1)
                (append1 term2 netExpr2)
                (append1 term3 netExpr3) )
 );let
);defun
```

   For example, the function generates the following default value for B.

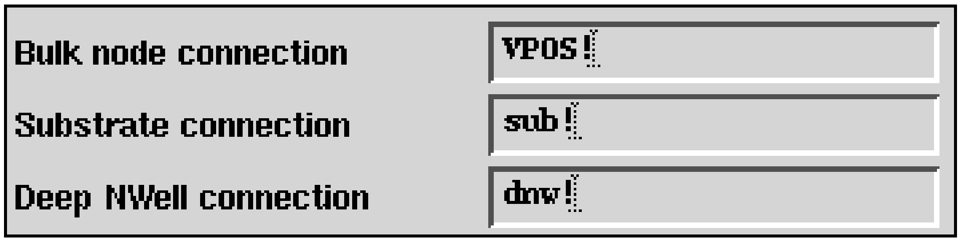
```
(nil name "B" direction "inputOutput" netExpr "[@vbulk_n:%:vssa!]")
```

   Referring to "extraTerminals" on page 657, you see that this value instructs the AMS netlister to create a connection for a terminal `B` in the instance connection port list for all instances of the mosfet device. The terminal is to be an input/output terminal. The netlist expression indicates that a property called `vbulk_n` is to be consulted for the name of the net to which terminal `B` is to be connected. In addition, if `vbulk_n` is not found, the `vssa!` net is to be used.

The most interesting part of this function, however, is the `Override values` section. In that section, the `if` statements of the form

```
(if (inst->bulkNode != "") then
  netExpr1 = (strcat "[@" inst->bulkNode ":%:" inst->bulkNode "]")     )
```

check for a value entered in a field of the Edit Object Properties form and generate a `netExpr` based on that value. In this function, `inst->bulkNode` refers to the value entered in the *Bulk node connection* field. The `dnw_node` and `sub_node` terms refer to the *Substrate connection* and *Deep NWell connection* fields. Assuming that values like the following have been entered into the fields of the Edit Object Properties form for a particular instance,



Bulk node connection	VPOS!
Substrate connection	sub!
Deep NWell connection	dnw!

the function generates an *extraTerminals* list like this.

```
((nil name "B" direction "inputOutput" netExpr "[@VPOS!:%:VPOS!]")
(nil name "DNW" direction "inputOutput" netExpr "[@dnw!:%:dnw!]")
(nil name "SUB" direction "inputOutput" netExpr "[@sub!:%:sub!]"))
```

The netlister then uses the *extraTerminals* list to generate inherited connections in the netlist.

3. Place the function in an override file.

   The function described in this example is associated with only the `mosfet` cell, so the appropriate override file to use is a `libInit.il` in the library that contains that cell.

4. Enter the name of the function in the *extraTerminals* field of the cell CDF.

   a. From the CIW, choose *Tools – CDF – Edit*.

      The Edit Component CDF form appears.

   b. In the *Library Name* and *Cell Name* field, specify the instance masters.

      The form expands to display the information for that master.

   c. In the *CDF Type* field, choose *Base*.

   d. Scroll down to the *Simulation Information* section and click *Edit*.

The Edit Simulation Information form appears.

e. In the *Choose Simulator* field, choose *ams*.

f. Scroll down to the *extraTerminals* field and type the name of the function, using the following format.

```
FUNCTION AMSnmos_dnw_inhExtraTerminals
```

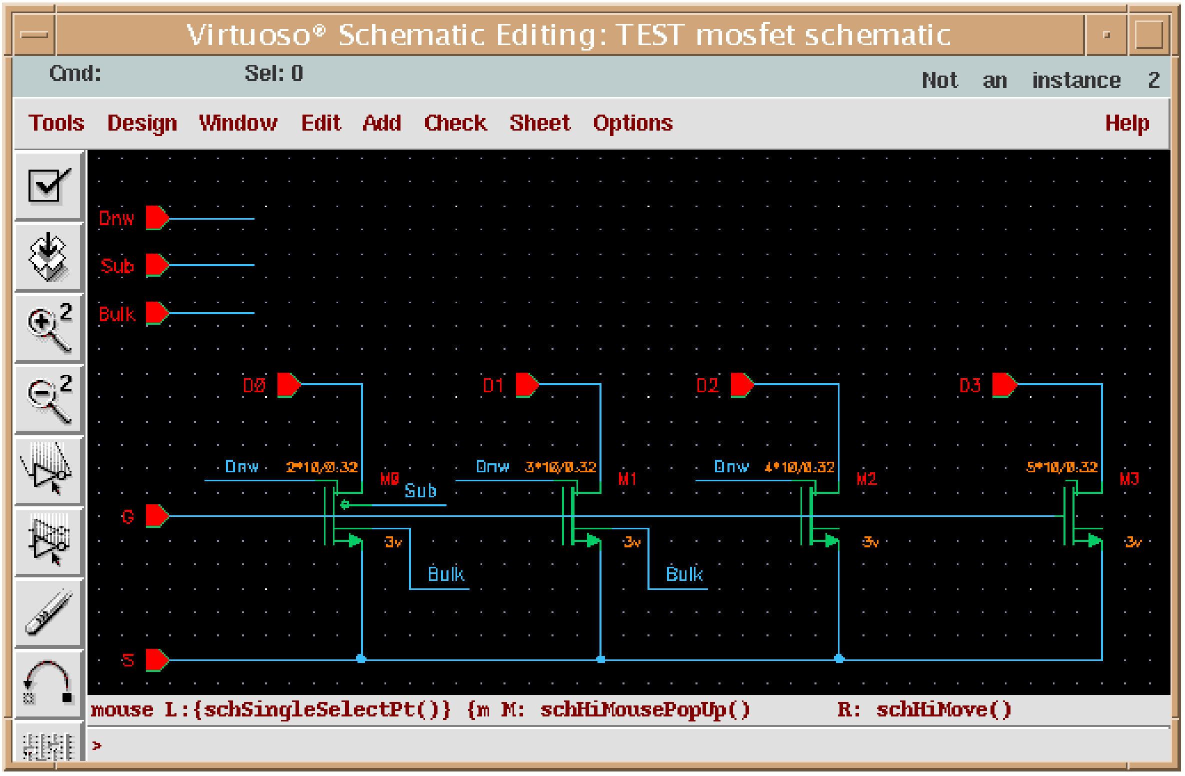
g. In the Edit Simulation Information form, click *OK*.

h. In the Edit Component CDF form, click *OK*.

Now whenever the netlister consults the *extraTerminals* field, the value of the field is calculated by the function.

5. Open the schematic that contains the instances that you want to connect with inherited connections.

For example, a schematic containing four instances of a mosfet might look like this.



6. Highlight an instance for which you want to create an instance-specific inherited connection and choose *Edit – Properties – Objects* from the menu of the schematic editing window.

The Edit Object Properties form appears.

7. Set the values of the programmable nodes for this instance.

   The programmable nodes for the mosfet symbol appear as the *Bulk node connection*, *Substrate connection*, and *Deep NWell connection* fields. The illustration in step 2 shows some possible values. The fields can also be left blank if you want the default net expression for that programmable node of the instance. (The default net expression is defined in the function, as described in step 2.)

8. Netlist the schematic that contains the instances of interest.

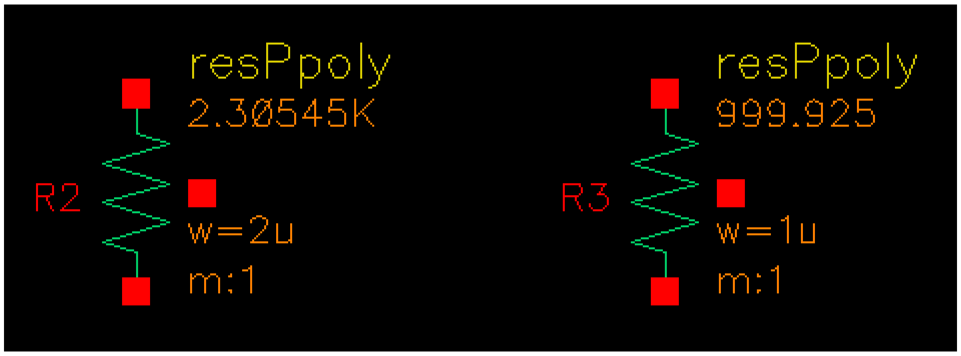
   The inherited connections for the instances are affected by the values you enter in step 7. For example, if the programmable node fields are left blank for instance `M0` and if the fields are set with the values `VPOS!`, `sub!`, and `dnw!` for instance `M3`, the resulting (partial) netlist looks like this.

```
module mosfet ( G,Bulk,D3,S,D1,Sub,Dnw,D0,D2 );

input   G;
input   Bulk;
input   D3;
input   S;
input   D1;
input   Sub;
input   Dnw;
input   D0;
input   D2;

wire
   (* integer inh_conn_prop_name="\\sub! ";
      integer inh_conn_def_value="cds_globals.\\sub! "; *)
  \sub!_sub! ;
wire
   (* integer inh_conn_prop_name="\\dnw! ";
      integer inh_conn_def_value="cds_globals.\\dnw! "; *)
  †nw!_dnw! ;
wire
   (* integer inh_conn_prop_name="\\VPOS! ";
      integer inh_conn_def_value="cds_globals.\\VPOS! "; *)
  \VPOS!_VPOS! ;
wire
   (* integer inh_conn_prop_name="\\bulk! ";
      integer inh_conn_def_value="cds_globals.\\bulk! "; *)
  •ulk!_bulk! ;

ns3v025d #(.as(4.64E-12), .ps(1.664E-05), .m(5), .ad(4.64E-12), .pd(1.664E-
05),
 .l(3.2E-07), .w(1E-05))
 (* integer library_binding = "cmos025";
    integer passed_mfactor = "m"; *)
  M3 ( D3, G, S, \VPOS!_VPOS! , \dnw!_dnw! , \sub!_sub!  );

ns3v025d #(.as(4.64E-12), .ps(1.664E-05), .m(2), .ad(4.64E-12), .l(3.2E-07),
 .pd(1.664E-05), .w(1E-05))
 (* integer library_binding = "cmos025";
    integer passed_mfactor = "m"; *)
  M0 ( D0, G, S, Bulk, Dnw, Sub );
```

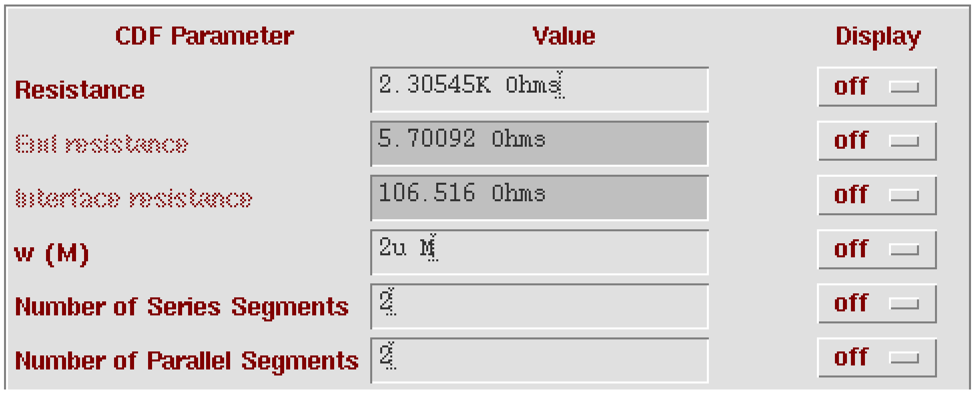
```
...
endmodule
```

Netlisting Schematic Parameterized Cells (Pcells)

AMS Designer netlister does not support schematic parameterized cells (pcells) but this section illustrates how you might use netlisting procedures to work around that limitation. A pcell is a programmable cell defined so that each instance of the cell can be customized. A schematic pcell allows the connectivity of the cell to change according to parameters that are specified for each instance of the cell. For example, assume that you have two instances of the `resPpoly` cell.



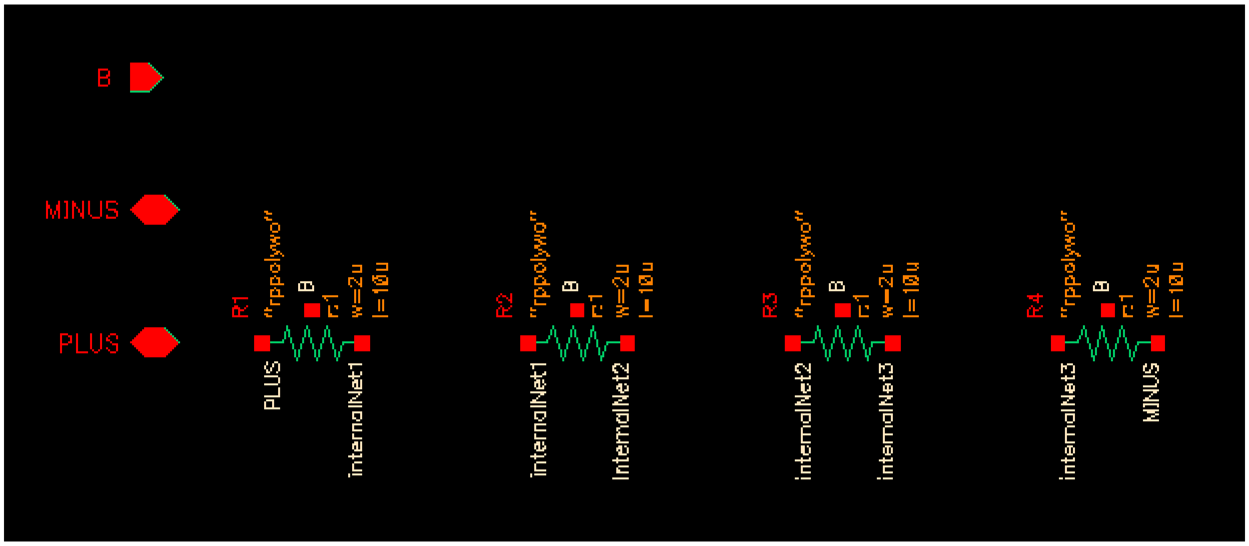
The `R2` instance is set up as follows in the Edit Object Properties form.



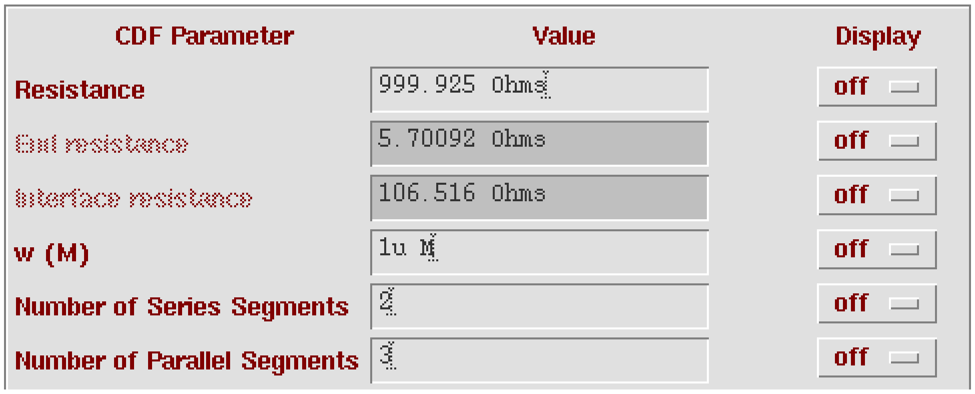
CDF Parameter	Value	Display
Resistance	2.30545K Ohms	off
End resistance	5.70092 Ohms	off
Interface resistance	106.516 Ohms	off
w (M)	2u M	off
Number of Series Segments	2	off
Number of Parallel Segments	2	off

When you *Descend Read* (not *Descend Edit*) into the `R2` schematic, you see a total of four subinstances, created in response to values specified in the *Number of Series*

Segments (2) and the *Number of Parallel Segments* (2) in the CDF Parameter section of the Edit Object Properties form.

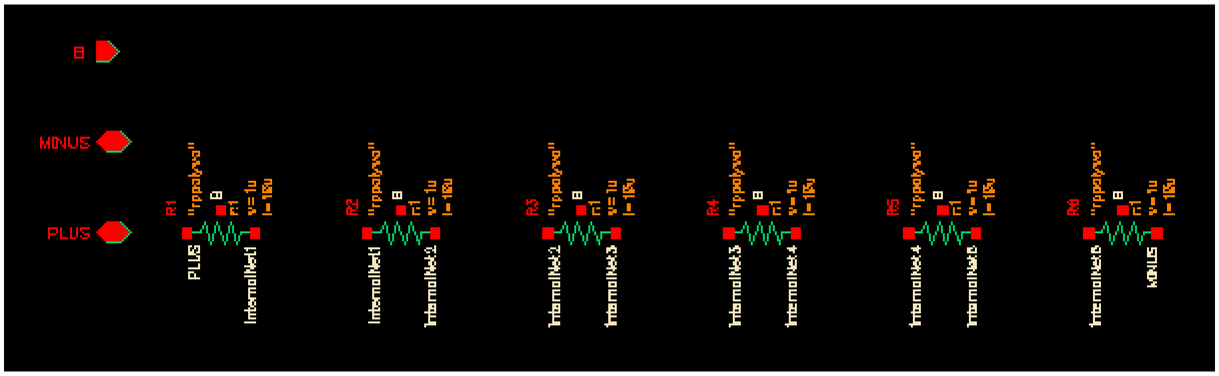


Now assume that instance *R3* is set up as follows in the Edit Object Properties form. Notice that the value specified for *Number of Parallel Segments* (3) is different from the value specified for instance `R2`.



CDF Parameter	Value	Display
Resistance	999.925 Ohms	off
End resistance	5.70092 Ohms	off
Interface resistance	106.516 Ohms	off
w (M)	1u M	off
Number of Series Segments	2	off
Number of Parallel Segments	3	off

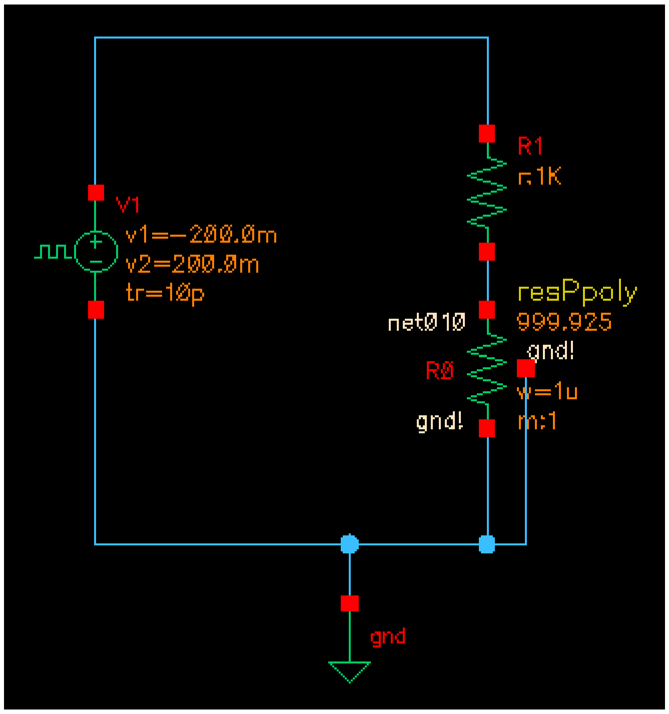
When you *Descend Read* into the `R3` schematic, you see a total of six (2 times 3) subinstances.



Correct netlists must reflect the varying number of subinstances, but, without appropriate customized netlisting procedures, netlists generated by the AMS netlister do not.

The following procedure illustrates one way to work around this limitation. The netlisting procedure given here works for this simple example and iterated instances; more complicated schematic pcells require expanded, more comprehensive procedures. The basic approach is to read the parameters that determine the number and arrangement of the subinstances. The netlister then uses this information to build the netlist statements for the pcells, replacing the netlisting that would otherwise occur for the pcells.

The netlisting procedure for this example netlists the `resPpoly` pcell instance in this simple schematic and another example that has iterated instances.



Here the `R0` `resPpoly` instance is set up with a value of 2 for the *Number of Series Segments* and with a value of 3 for the *Number of Parallel Segments*. As a result, there are six subinstances that need to be netlisted.

To set up for this approach,

1. Identify the information to be obtained.

   The *Number of Series Segments* is stored as `srs` and the *Number of Parallel Segments* is stored as `prl`. These names appear in the Component Parameters section of the Edit Component CDF form, when that form displays information for the `amsd_discrep1.resPpoly` cell.

2. Prepare the netlisting procedure.

   The procedure must find the needed parameters, then use that information to instantiate the subinstances. In the following override file, the main procedure is `UserAmsResNLProc`, which calls the other routines in the file as necessary.

```
(defun UserFindProp (inst name)
  ;; The search order is this:
  ;; 1. Look in inst->parameters (AMS Netlister s params)
  ;; 2. Look in inst->id->prop (the dbInstId s props)
  ;; 3. Look on the master s parameters (AMS Netlister s params)
  ;; After this point, you could go look on:
  ;;   Master s CDF
  ;;   Master s cellview properties
  ;;   And any other places where you expect the param.
  ;; But we stop here.
  (let (prop)
    ;; Get the name from AMS Netlister s params for the instance
    (unless (setq prop (car (exists p inst->parameters (equal p->name name))))
      ;; Nope. No param there.
      ;; Try the dbInstId
      (unless (setq prop (dbGetPropByName inst->id name))
        ;; Nope, no prop on dbInstId
        ;; Try master s params known to AMS Netlister.
        (setq prop
              (car (exists p inst->master->parameters (equal p->name name))))
        )                         ; unless no prop on db inst id
      )                           ; unless no param on instance
    prop
    )                             ; let
  )                               ; defun


procedure( UserAmsResNLProc( formatterId cvId instanceId)
let( ( srs prl model l w instName newInstName newNet internalNet rangeL rangeR)
  srs = UserFindProp( instanceId "srs")->value
  prl = UserFindProp( instanceId "prl")->value
  if(stringp(srs) srs = evalstring(srs))
  if(stringp(prl) prl = evalstring(prl))
  if( ! srs srs = 1 )
  if( ! prl prl = 1 )

  model = UserFindProp( instanceId "model")->value
  l = UserFindProp( instanceId "l")->value
  w = UserFindProp( instanceId "w")->value

  amsPrint( formatterId "\n")

  if( instanceId->range then
    instName = amsGetInstanceName( formatterId instanceId )
    amsPrint(formatterId
             strcat( "\n//***** Beginning Iterated Instance(s) of " instName "
                     netlist."))
    rangeL = car(instanceId->range)
    rangeR = cadr(instanceId->range)
    for( i 0 max(rangeL rangeR)-min(rangeL rangeR)
       newInstName = amsGetInstanceName( formatterId instanceId i)
       UserAmsPrintIteratedInstance( formatterId cvId instanceId newInstName
                                     srs prl i)
    )
    amsPrint(formatterId
             strcat( "\n//***** End of Iterated instance(s) of " instName "
                     netlist."))
  else
    instName = amsGetInstanceName( formatterId instanceId )
    UserAmsPrintIteratedInstance(formatterId cvId instanceId instName srs prl)
  )
);let
)

;UserAmsPrintIteratedInstance prints instance/particular iterated instance.
;Arguments:
```

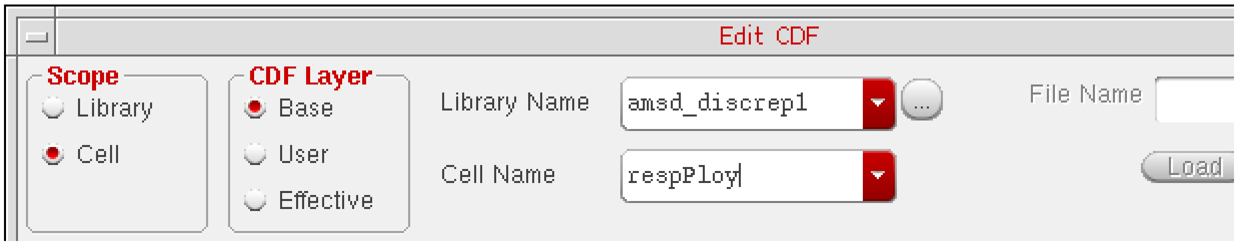
```
;  formatterId : caller passes formatterId to be used in various custom APIs.
;  cvId        : caller passes cellviewId to be used in various custom APIs.
;  instanceId  : caller passes instanceId to be used in various custom APIs.
;  instName    : name of instance, in case of iterated passed expanded name
;                for iterated instance to be netlisted.
;  srs         : srs value indicates PCell expanded in series.
;  prl         : prl value indicates mfactor set accordingly so that ams
;                simulator expand it as instances in parallel, this is
;                internally handled by ams simulator.
;  iter        : (optional) if specified, indicates to print iter-th instance
;                expanded in parallel.
;
procedure( UserAmsPrintIteratedInstance( formatterId cvId instanceId instName
                                         srs prl @optional (iter nil))
   let( ( myPrintString newInst newNet internalNet)
      newInst = instName
      if( srs == 1 then
         amsPrintInstanceMasterName( formatterId cvId instanceId)
         UserAmsPrintResParams( formatterId instanceId)
         UserAmsPrintResLibBind( formatterId instanceId)
         amsPrint( formatterId instName)
         if( iter then
             amsPrintInstancePorts( formatterId instanceId iter )
         else
             amsPrintInstancePorts( formatterId instanceId )
         )
         amsPrint( formatterId ";")
      else
         myPrintString = strcat( "\n//***** Beginning of " instName "
                                  netlist.")
         amsPrint( formatterId myPrintString)
         internalNet = makeTable( "internalNet" nil)
         if( iter then
            internalNet[ 0] =
              amsGetPortExpr( formatterId car(instanceId->ports) iter)
            internalNet[ srs] =
              amsGetPortExpr( formatterId cadr( instanceId->ports) iter)
            internalNet[ srs + 1] = when( caddr( instanceId->ports)
              amsGetPortExpr( formatterId caddr( instanceId->ports) iter))
         else
            internalNet[ 0] = car( instanceId->ports)->expr
            internalNet[ srs] = cadr( instanceId->ports)->expr
            internalNet[ srs + 1] = caddr( instanceId->ports)->expr
         )
         for( i 1 srs
            if( i > 1 then
               amsPrint( formatterId "\n")
               newInst = strcat(" " instName sprintf( nil "_%d" i))
            )
            unless( i == srs
               internalNet[ i] = sprintf(nil "net_%s"
                 amsGetUniqueName(formatterId `net))
            )

            amsPrintInstanceMasterName( formatterId cvId instanceId)
            UserAmsPrintResParams( formatterId instanceId)
            UserAmsPrintResLibBind( formatterId instanceId)
            amsPrint( formatterId newInst)
            if( internalNet[ srs + 1] then
               ; three terminal device
               myPrintString = strcat( " ( " internalNet[ i - 1] ",
                  " internalNet[ i] ", " internalNet[ srs + 1] " )")
            else
               ; two terminal device
               myPrintString = strcat( " ( " internalNet[ i - 1] ",
                  " internalNet[ i] " )")
            )
            amsPrint( formatterId myPrintString)
            amsPrint( formatterId ";")
         );for

         myPrintString = strcat( "//***** End of " instName " netlist.")
         amsPrint( formatterId myPrintString)
      )
   );let
);UserAmsPrintIteratedInstance

procedure( UserAmsPrintResParams( formatterId instanceId)
let( ( myPrintString commaText )
  amsPrint( formatterId " #(" )
  commaText = ""
  foreach( p setof( p instanceId->parameters
                    !member( p->name list("srs" "prl")))
    if( p->name == "m"  then
      myPrintString = sprintf( nil "%s .%s(%f)"
                               commaText p->name atof(p->value) * prl)
    else
      myPrintString = sprintf( nil "%s .%s(%s)" commaText p->name p->value)
    )
    amsPrint( formatterId myPrintString)
    commaText = ","
  )
  amsPrint( formatterId ")" )
)
)

procedure( UserAmsPrintResLibBind( formatterId instanceId)
let( ( myPrintString mfactor )
  amsPrint( formatterId "\n(*")
  myPrintString = strcat("  integer library_binding = \""
                            instanceId->master->libName "\"; ")
  amsPrint( formatterId myPrintString)
  mfactor = car( exists( p instanceId->parameters p->name == "m"))

  when( or( (mfactor->value != "1") (prl > 1))
    amsPrint( formatterId " integer passed_mfactor = \"m\";" )
  )
  amsPrint( formatterId "\n*)\n")
)
)
```

3. Place the override file where it can be read.

   The `UserAmsResNLProc` function is used with only the `resPpoly` cell, so the appropriate location is a `libInit.il` in the library that contains that cell.

4. In the CIW, choose *Tools – CDF – Edit* to open the Edit Component CDF form.

5. In the *Library Name* and *Cell Name* fields of the Edit Component CDF form, type the library and cell name of the `resPpoly` cell.



Scope: Library [✓ Cell]
CDF Layer: [✓ Base] User Effective
Library Name: amsd_discrep1
Cell Name: respPloy

The form expands to display the information for the cell.

6. Scroll down to the *Simulation Information* area and click *Edit*.

   The Edit Simulation Information form appears.

7. In the *Choose Simulator* cyclic, choose *AMS*.

   The form changes to display the values for the AMS simulator.

8. In the *netlistProcedure* field, type `UserAmsResNLProc`, the name of the primary netlisting procedure.

   This step causes the netlister to run the customized netlisting procedure every time the cell is netlisted.

9. Click *OK*.

   The Edit Simulation Information form closes.

10. In the Edit Component CDF form, click *OK*.

    The Edit Component CDF form closes.

11. Netlist the design and examine the netlist.

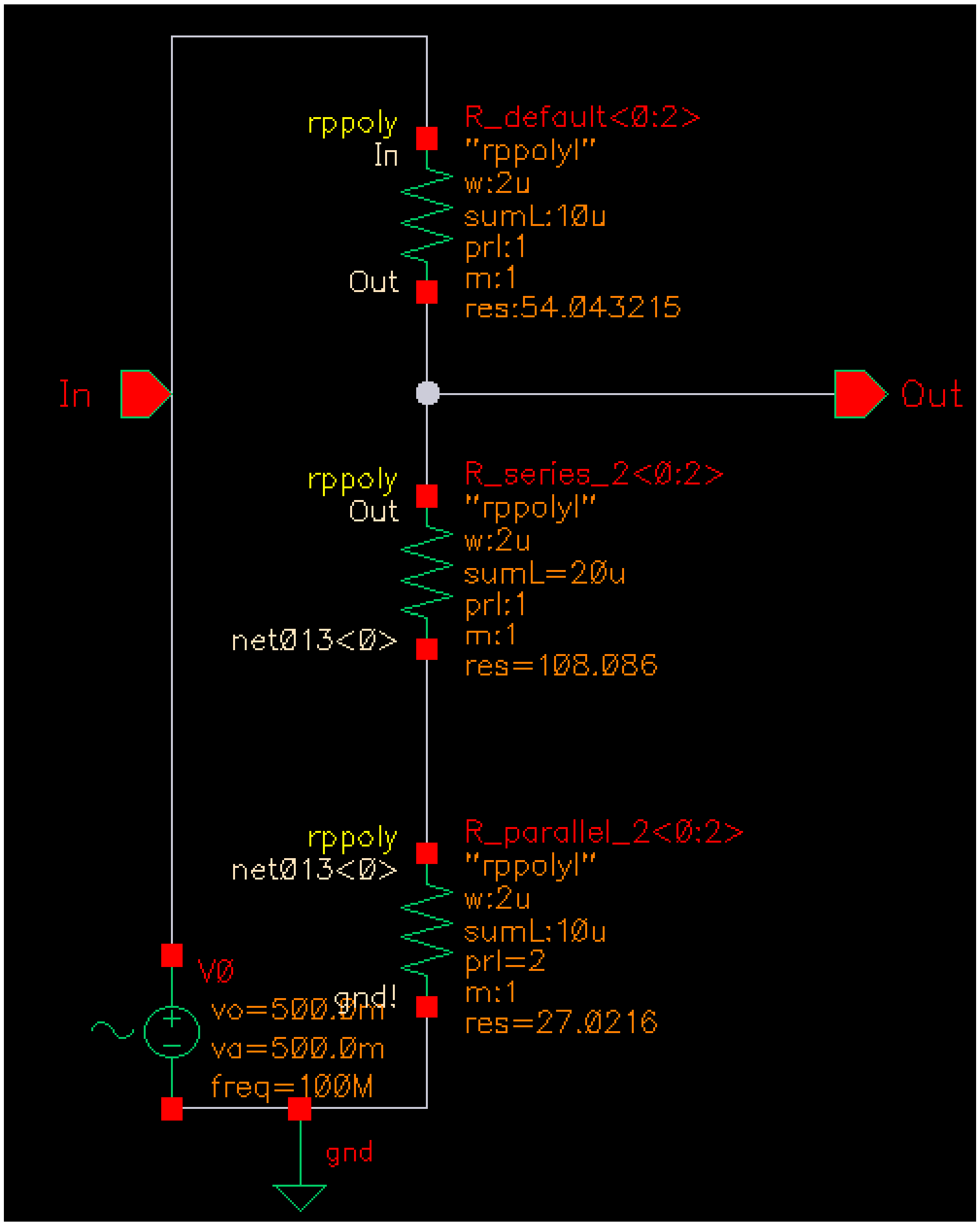
    Notice, in the following netlist, that there are three instantiations in the section labeled

```
//***** Beginning of R0 netlist.
```

These instantiations correspond to the *Number of Parallel Segments* value (3). The value of the `Number of Series Segments` (2) appears in the netlist as the `passed_mfactor` value of 2.000000.

```
// Verilog-AMS netlist generated by the AMS netlister, version 5.10.41_USR2
// Cadence Design Systems, Inc.

`include "disciplines.vams"

module amsd_discrep1 (  );


resistor #(.r(1K)) (*
integer library_binding = "analogLib";
 *)
R1 (
net011, net010 );

//***** Beginning of R0 netlist.
rppolywo  #( .l(33.755u), .w(1u), .m(2.000000), .r(999.925))
(*
  integer library_binding = "amsd_discrep1";
  integer passed_mfactor = "m";
*)
R0
  ( net010, ams_uniq_name_0, cds_globals.„nd!  );

rppolywo  #( .l(33.755u), .w(1u), .m(2.000000), .r(999.925))
(*
  integer library_binding = "amsd_discrep1";
  integer passed_mfactor = "m";
*)
 R0_2
  ( ams_uniq_name_0, ams_uniq_name_1, cds_globals.„nd!  );

rppolywo  #( .l(33.755u), .w(1u), .m(2.000000), .r(999.925))
(*
  integer library_binding = "amsd_discrep1";
  integer passed_mfactor = "m";
*)
 R0_3
  ( ams_uniq_name_1, cds_globals.„nd! , cds_globals.„nd!  );
//***** End of R0 netlist.

vsource #(.type("pulse"), .period(400p), .width(190p), .val0(-200.0m),
 .rise(10p), .val1(200.0m), .fall(10p)) (*
integer library_binding
 = "analogLib";
 *)
V1 ( net011, cds_globals.„nd!  );

endmodule
```

The following example is a bit more complex, using the same custom netlist procedures for a cell that is instantiated using iterated instances:



In this example there are three instances of resistor `rppoly`:

1. `R_default<0:2>` is an iterated instance having range 0 to 2, `prl=1`, `srs=1`, and `m=1`.

   `R_default_0`, `R_default_1`, and `R_default_2` are in parallel and all connect to nets `In` and `Out`.

2. `R_series_2<0:2>` is an iterated instance having range 0 to 2, `prl=1`, `srs=2`, and `m=1`.

   `R_series_2_0`, `R_series_2_1`, and `R_series_2_2` each have two segments in series (`srs=2`) such that:

   - ❑ `R_series_2_0` and `R_series_2_0_2` are in series,
   - ❑ `R_series_2_1` and `R_series_2_1_2` are in series, and
   - ❑ `R_series_2_2` and `R_series_2_2_2` are in series.

3. `R_parallel_2<0:2>` is an iterated instance having range 0 to 2, `prl=2`, `srs=1`, and `m=1`.

   - ❑ `R_parallel_2_0`, `R_parallel_2_1`, and `R_parallel_2` each have two segments in parallel (`prl=2`).
   - ❑ `R_parallel_2_0`, `R_parallel_2_1`, and `R_parallel_2` have parameter `m=2` (the netlister sets `passed_mfactor` accordingly in netlist).

When you netlist this design, the netlister creates the following netlist:

```
// Verilog-AMS netlist generated by the AMS netlister, version 5.10.41.500.5.120.
// Cadence Design Systems, Inc.

`include "disciplines.vams"

(* cds_ams_schematic *)
module test_cell ( Out,In );

output   Out;
input   In;

wire [0:2] net013;


vsource #(.type("sine"), .ampl(500.0m), .freq(100M), .dc(0.0), .sinedc(500.0m))
(*
 integer library_binding = "analogLib";
*)
V0 ( In,
cds_globals.\gnd!  );


//***** Beginning Iterated Instance(s) of R_parallel_2 netlist.
rppolyl #( .m(2.000000), .mismatchflag(1), .w(2u), .mf(2), .l(10u))
(*
 integer library_binding = "myLib";
 integer passed_mfactor = "m";
*)
R_parallel_2_0 ( net013[0],
cds_globals.\gnd!  );
rppolyl #( .m(2.000000), .mismatchflag(1), .w(2u), .mf(2), .l(10u))
(*
 integer library_binding = "myLib";
 integer passed_mfactor = "m";
*)
R_parallel_2_1 ( net013[1],
cds_globals.\gnd!  );
rppolyl #( .m(2.000000), .mismatchflag(1), .w(2u), .mf(2), .l(10u))
(*
 integer library_binding = "myLib";
 integer passed_mfactor = "m";
*)
R_parallel_2_2 ( net013[2],
cds_globals.\gnd!  );

//***** End of Iterated instance(s) of R_parallel_2 netlist.


//***** Beginning Iterated Instance(s) of R_series_2 netlist.

//***** Beginning of R_series_2_0 netlist.
rppolyl #( .m(1.000000), .mismatchflag(1), .w(2u), .mf(1), .l(10u))
(*
 integer library_binding = "myLib";
*)
R_series_2_0
( Out, net_ams_uniq_name_8 );

rppolyl #( .m(1.000000), .mismatchflag(1), .w(2u), .mf(1), .l(10u))
(*
 integer library_binding = "myLib";
*)
R_series_2_0_2
( net_ams_uniq_name_8, net013[0] );
//***** End of R_series_2_0 netlist.

//***** Beginning of R_series_2_1 netlist.
rppolyl #( .m(1.000000), .mismatchflag(1), .w(2u), .mf(1), .l(10u))
(*
 integer library_binding = "myLib";
*)
R_series_2_1
( Out, net_ams_uniq_name_9 );

rppolyl #( .m(1.000000), .mismatchflag(1), .w(2u), .mf(1), .l(10u))
(*
 integer library_binding = "myLib";
*)
R_series_2_1_2
( net_ams_uniq_name_9, net013[1] );
//***** End of R_series_2_1 netlist.

//***** Beginning of R_series_2_2 netlist.
rppolyl #( .m(1.000000), .mismatchflag(1), .w(2u), .mf(1), .l(10u))
(*
integer library_binding = "myLib";
*)
R_series_2_2
( Out, net_ams_uniq_name_10 );

rppolyl #( .m(1.000000), .mismatchflag(1), .w(2u), .mf(1), .l(10u))
(*
integer library_binding = "myLib";
```

```
*)
R_series_2_2_2
( net_ams_uniq_name_10, net013[2] );
//***** End of R_series_2_2 netlist.

//***** End of Iterated instance(s) of R_series_2 netlist.


//***** Beginning Iterated Instance(s) of R_default netlist.
rppolyl #( .m(1.000000), .mismatchflag(1), .w(2u), .mf(1), .l(10u))
(*
integer library_binding = "myLib";
*)
R_default_0 ( In, Out );
rppolyl #( .m(1.000000), .mismatchflag(1), .w(2u), .mf(1), .l(10u))
(*
integer library_binding = "myLib";
*)
R_default_1 ( In, Out );
rppolyl #( .m(1.000000), .mismatchflag(1), .w(2u), .mf(1), .l(10u))
(*
integer library_binding = "myLib";
*)
R_default_2 ( In, Out );

//***** End of Iterated instance(s) of R_default netlist.

endmodule
```

Data Objects Supported for Netlisting

The data used for netlisting is organized into objects. The supported AMS netlister objects are described in the following pages. They are:

- Netlister Object on page 339
- Formatter Object on page 340
- Cellview Object on page 342
- Parameter Object on page 343
- Instance Object on page 345
- Port Object on page 346
- IO Object on page 348
- Wire Object on page 349
- Alias Object on page 351
- Attribute Object on page 352

Netlister Object

The netlister object contains the global options applicable to the AMS netlister. You obtain the ID of the netlister object by calling the `amsGetNetlister` function, with a command like

```
A_netlisterId = amsGetNetlister()
```

The netlister object is available throughout the life of the UNIX process that is running the AMS netlister and is a unique object for that process. The netlister object ID does not change as long as the process remains the same.

Running a command like

```
A_netlisterId->?
```

you find that the netlister object contains the following fields:

Field	Values
`lsbMsb`	boolean `t/nil`
`scalarizeInstances`	boolean `t/nil`

Field	Values
`includeInstCDFParams`	boolean `t`/`nil`
`excludeParams`	*`l_params`*
`expScalingFactor`	`'no` | `'sci` | `'dec`
`modifyParamScope`	`'no` | `'warn` | `'yes`
`vlog`	The ID of the Verilog-AMS formatter object.
`vhdl`	Always `nil`. No VHDL formatter is supported.

All of these fields are read-only. Attempting to change their values results in an error.

When you add new data to the database, be sure that the data adhere to the format and content requirements set by the values of these fields.

Formatter Object

The formatter object contains information about the netlist procedures for the formatter it represents. You use a command like the following to obtain the ID of the formatter:

```
A_formatterId = A_netlisterId->vlog
```

In this release, the only available formatter is the Verilog-AMS (`vlog`) formatter.

The formatter object is created when the `vlog` field of the netlister object is accessed for the first time. After that, the formatter object is available throughout the life of the UNIX process that is running the AMS netlister and is a unique object for that process. The formatter object ID does not change as long as the process remains the same.

Running the command

```
A_formatterId->?
```

you find that the formatter object contains the fields listed in the next two tables.

Table 16-1 Fields of the Formatter Object

Field	Values
`comments`	*`string`*
`headers`	*`string`*
`ifdefLanguageExtensions`	boolean `t`/`nil`

Table 16-1 Fields of the Formatter Object, *continued*

Field	Values
`useDefparam`	boolean `t`/`nil`
`includeFiles`	*`l_strings`*
`paramDefVals`	*`l_paramValPairs`*
`paramGlobalDefVal`	*`paramGlobalDefVal`*
`netlister`	*`A_netlisterId`*
`cellviewId`	*`A_cellviewId`*

All of these fields are read-only. Attempting to change their values results in an error.

The formatter object also lists the netlisting procedures that are in effect. The sections of the netlist that each procedure is responsible for generating are listed in the right column. The entries in that column are the same as the labels shown in Figure 16-1 on page 302.

Table 16-2 Netlisting Procedures Listed in the Formatter Object

Field	Value	Netlist Section
`commentsProc`	`amsPrintComments` or current override procedure	Comment
`headersProc`	`amsPrintHeaders` or current override procedure	Header
`moduleProc`	`amsPrintModule`	All sections except Comment, Header, Includes list, Footer
`footersProc`	`amsPrintFooters`	Footer (not shown)
`moduleNameProc`	`amsPrintModuleName`	Mapped name
`portsProc`	`amsPrintPorts`	Port declarations
`iosProc`	`amsPrintIOs`	Port declarations
`parametersProc`	`amsPrintParameters` or current override procedure	Parameter declarations
`wiresProc`	`amsPrintWires`	Signal declarations
`aliasesProc`	`amsPrintAliases`	Alias list (not shown)

Table 16-2 Netlisting Procedures Listed in the Formatter Object, *continued*

Field	Value	Netlist Section
`instanceProc`	`amsPrintInstance` or current override procedure	Instances
`instanceMasterNameProc`	`amsPrintInstanceMasterName` or current override procedure	Instance master
`instanceParametersProc`	`amsPrintInstanceParameters` or current override procedure	Instance parameters
`instancePortsProc`	`amsPrintInstancePorts` or current override procedure	Instance connections
`attributesProc`	`amsPrintAttributes` or current override procedure	Attributes

Cellview Object

The cellview object contains fields that characterize a cellview.

The cellview object is created when netlisting begins for the cellview and is destroyed when netlisting ends. As a consequence, to view the fields, you need to ask for the information during the netlisting run. You might, for example, use statements like the following in your netlisting procedures override file to obtain a list of the fields.

```
;; Define a print comments functions
(defun MyProc (formatter cellview )
   (printf "Var = %L\n" cellview->??)
   )
```

Running these statements, you find that the cellview object contains the following fields. Some of these fields return information that is collected indirectly and is therefore more costly in terms of processing time and memory. For efficient netlisting, use such indirectly collected information sparingly.

Field	Values	Meaning of the Value
`libName`	*`string`*	Name of the library.
`cellName`	*`string`*	Name of the cell.
`viewName`	*`string`*	Name of the cellview.
`id`	DB	ID of the cellView

Field	Values	Meaning of the Value
`primitive`	`t/nil`	Whether the master is a Spectre primitive. (Ports must be printed by order for instances of primitives.)
`compName`	*`string`*	componentName field from the simInfo.
`termOrder`	*`l_strings`*	List of terminals for the component. (The list can change for each instance.)
`ports`	*`l_ports`*	Collected indirectly. The ports of the cellview.
`parameters`	*`l_parameters`*	Collected indirectly. The parameters of the cellview.
`IOs`	*`l_IOs`*	Collected indirectly. The IOs of the cellview.
`wires`	*`l_wires`*	Collected indirectly. The wires of the cellview.
`aliases`	*`l_aliases`*	Collected indirectly. The aliases of the cellview.

All of these fields are read-only, so attempting to change their values results in an error.

Parameter Object

The parameter object contains information about both cellview parameters and instance parameters. Parameters for the cellview come primarily from the base cell CDF but can also come from pPar references on instance properties and from `[@` and `[+` NLP expressions on instances. Parameters for instances come either from the base cell CDF of the instance master cellview or from the database properties placed on the instance.

The parameter object always contains the final evaluated value that would be written to the netlist in the absence of any netlisting procedures. Consequently, you do not have to parse any AEL or NLP expressions that affect the parameters.

The parameter object for a cellview is created just prior to calling the parametersProc netlisting procedure and is destroyed when the cellview object is destroyed. The parameter object for an instance is created when the instance is created and is destroyed when the instance is destroyed. As a consequence, to view the fields, you need to ask for the information during the netlisting run. You might, for example, use statements like the following in your netlisting procedures override file to obtain a list of the fields.

```
(defun MYPrintParameters (formatterId cvId)

   cellview_params = cvId->parameters
   ;; Consider each parameter
   (foreach param cellview_params
      (printf "Param fields: %L\n" param->?)
   );; foreach

   ;; Call the default print parameters function.
   amsPrintParameters(formatterId cvId)

   );;defun

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

Running these statements, you find that the parameter object contains the following fields.

Field	**Value**	**Meaning of the Value**
`name`	*`string`*	Name of the parameter (as used in the Verilog-AMS netlist).
`cdfName`	*`string`*	Name of the parameter (from the cellview data).
`type`	*`string`*	Type of the parameter (from the parameter declaration).
`dbType`	symbol `int / float`, `string / ael / aelNoNum / aelNum`	Type of the parameter (as found in the DB/ CDF).
`value`	*`string`*, *`integer`*, or *`float`* (as specified by `type`).	Value of the parameter. This is a print-ready string which already contains quotation marks, if quotation marks are required.
`isDefault`	`t`/`nil`	`t` means the value is specified in the base cell CDF defaults; `nil` means the value is overridden on the instance.
`owner`	*`amsobject`*	Cellview or instance object to which the parameter belongs.
`ignore`	`t`/`nil`	`t` means the parameter is to be left out of the netlist; `nil` means the parameter is to be included in the netlist.

Except for `value` and `ignore`, these fields are read-only. You can assign a new value to the `value` field. You can change the value of the `ignore` field to control whether or not the parameter is printed. You cannot change the value of the `type` field so any changed value for the `value` field must be the same type as the original.

Instance Object

The instance object contains information about a particular instance of a cellview.

The instance object is created before calling any of the `instanceProc`, `instanceMasterNameProc`, `instanceParametersProc`, or `instancePortsProc` netlisting procedures and is destroyed when the netlisting procedure returns. As a consequence, to view the fields you need to ask for the information during the netlisting run. You might, for example, use statements like the following in your netlisting procedures override file to obtain a list of the fields.

```
(defun MYPrintInstance (formatterId cvId instanceId)
   (printf "\n/*\n Instance fields: %L \n*/\n" instanceId->?)
;; call the default print instance function.
   (amsPrintInstance formatterId cvId instanceId)
   );;defun
;; Set up the custom netlist procedure
netId = amsGetNetlister()
vlog = netId->vlog
vlog->instanceProc = 'MYPrintInstance
```

Running these statements, you find that the instance object contains the following fields. Some of these fields return information that is collected indirectly and is therefore more costly of processing time and memory. For efficient netlisting, use such indirectly collected information sparingly.

Field	Value	Meaning of the Value
`id`	DB	instanceId.
`name`	*`string`*	Name of the instance.
`range`	*`l_integers`*	`'(` *`x_left x_right`* `),nil` for scalars.
`master`	*`A_masterCellViewId`*	Collected indirectly. The cellViewId of the instance master.

Field	Value	Meaning of the Value
`masterName`	*`t_masterName`*	Collected indirectly. The master name determined after applying the algorithm discussed in "componentName" on page 654.
`attributes`	*`l_attributes`*	Collected indirectly. The attributes of the instance. Always returns `nil`.
`parameters`	*`l_parameters`*	Collected indirectly. The parameters of the instance.
`ports`	*`l_ports`*	Collected indirectly. The ports of the instance. Always returns `nil`.

Except for `masterName`, these fields are read-only. You can change the value of the `masterName` field by overriding the `amsPrintInstanceMasterName` function. If you do override this function, Cadence recommends that your overriding function set `masterName` to reflect the change.

Port Object

The port object contains information about a particular port of a cellview or instance of a cellview.

The port objects for a cellview are created before calling the `portsProc` netlisting procedure. The port objects of a cellview exist during the lifetime of the cellview object and are destroyed when the cellview object is destroyed.

The port objects for an instance are created before and are available during all of the instance netlisting procedures. The port objects for an instance are destroyed when the instance is destroyed. To view the fields you need to ask for the information during the netlisting run. You might, for example, use statements like the following in your netlisting procedures override file to obtain a list of the fields.

```
(defun MYPrintPorts (formatterId cvId)
   cellview_ports = cvId->ports
   ;; Consider each port
   (foreach port cellview_ports
      (printf "Port fields: %L\n" port->?)
   );; foreach
   ;; Call the default print ports function.
   amsPrintPorts(formatterId cvId)
   );;defun
   ;; Set up the custom netlist procedure
netId = amsGetNetlister()
```

```
vlog = netId->vlog
vlog->portsProc = 'MYPrintPorts
```

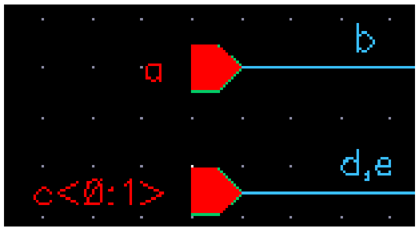
Running these statements on a cellview that has ports, you find that the port object contains the following fields.

Field	**Value**	**Meaning of the Value**
`name`	*`string`*	Base name of the port.
`direction`	*`string`*	Direction of the port.
`expr`	*`list of strings`*	Port expression, if any. Nil for plain ports.
`owner`	*`cellViewId`* or *`InstanceId`*	Cellview or instance object to which the port belongs.

These fields are read-only.

Port Expressions

A port expression is one of the pieces of information used to establish the connection between terminals and nets, as illustrated here. To begin, consider the terminals and nets in this schematic.



Terminal `a` is connected to net `b`, and terminal `c<0:1>` is connected to net `d,e`. In this example, port `a` has the port expression `b`, and port c[0:1] has the port expression `{d,e}`. The netlist created for this module looks like:

```
module port_expr ( .a(b),.c({ d,e }) );
    input   b;
    input   d;
    input   e;
```

The `expr` field of the port object contains the expression (which might be a bundle, enclosed in curly brackets). The constituents of the `expr` field are in the Verilog-AMS namespace. If the `expr` field is not nil and not empty, the netlist procedure must print the port as

```
.name ( expr )
```

Had port `a` in this example been connected to a net also named `a`, there would be no port expression, and the netlist would be:

```
module port_expr ( a,.c({ d,e }) );
    input   a;
    input   d;
    input   e;
```

This illustrates a *plain* port, one that uses no port expression.

For an iterated instance, the `expr` is the complete bundle for all the iterations of the instance port. It is possible, however, to obtain the `expr` associated with a port for a given iteration of an iterated instance. For more information, see "amsGetPortExpr" on page 731.

IO Object

The IO object contains information about the direction, range and type for each port in the port list. When a port expression is used, the IO list is different from the port list. Otherwise, the two lists are the same.

The IO objects are created before calling the `iosProc` netlisting procedure. The IO objects of a cellview exist during the lifetime of the cellview object and are destroyed when the cellview object is destroyed. As a consequence, to view the fields, you need to ask for the information during the netlisting run. You might, for example, use statements like the following in your netlisting procedures override file to obtain a list of the fields.

```
(defun MYPrintIOs (formatterId cvId)
   cellview_IOs = cvId->IOs
   ;; Consider each io object
   (foreach io cellview_IOs
      (printf "IO fields: %L\n" io->?)
   );; foreach
   ;; Call the default print ports function.
   amsPrintIOs(formatterId cvId)
);;defun
;; Set up the custom netlist procedure
netId = amsGetNetlister()
vlog = netId->vlog
vlog->iosProc = 'MYPrintIOs
```

Running these statements, you find that the IO object contains the following fields.

Field	Value	Meaning of the Value
`name`	*string*	Base name of the IO.
`direction`	*string*	Direction of the port.

Field	Value	Meaning of the Value
`range`	*`list of integers`*	`x_left x_right`. Nil for scalars.
`attributes`	*`l_attributes`*	Collected indirectly. The attributes of the IO object.

These fields are read-only.

Wire Object

The wire object is an abstraction for an internal wire of a cellview. Properties on nets are merged or unified to obtain a single set of non-conflicting properties, which can specify the type or disciple for the wire. There can be net expressions on the wire.

A scalar wire does not have to be declared but is declared if the wire

- Is a vector
- Has a type other than wire
- Has a discipline
- Has a net expression, in which case attributes are written for the wire.

Global signals are not declared.

The wire objects are created before calling the `wiresProc` netlisting procedure. Wire objects exist during the lifetime of the cellview object and are destroyed when the cellview object is destroyed. As a consequence, to view the fields, you need to ask for the information during the netlisting run. You might, for example, use statements like the following in your netlisting procedures override file to obtain a list of the fields.

```
(defun MYPrintWires (formatterId cvId)
   cellview_wires = cvId->wires
   ;; Consider each wire object
   (foreach wire cellview_wires
      (printf "Wire fields: %L\n" wire->?)
   );; foreach
   ;; Call the default print wires function.
   amsPrintWires(formatterId cvId)
   );;defun
;; Set up the custom netlist procedure
netId = amsGetNetlister()
vlog = netId->vlog
vlog->wiresProc = 'MYPrintWires
```

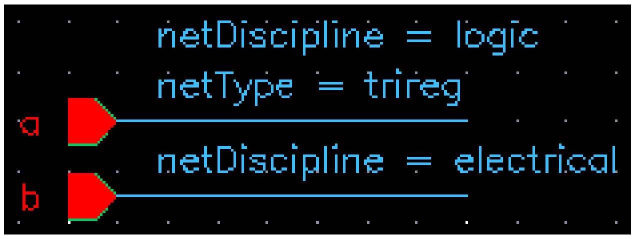
Running these statements, you find that the wire object contains the following fields.

Field	**Value**	**Meaning of the Value**
`name`	*`string`*	Base name of the IO.
`range`	*`list of integers`*	`x_left x_right`. Nil for scalars.
`type`	*`string`*	Type of the wire, determined from the `netType` property.
`discipline`	*`string`*	Discipline of the wire, determined from the `netDiscipline` property.
`global`	*`boolean`*	Whether the wire is a global signal.
`inh`	*`boolean`*	Whether the wire is an inherited signal.
`attributes`	*l_attributes*	Collected indirectly. The attributes of the wire object.

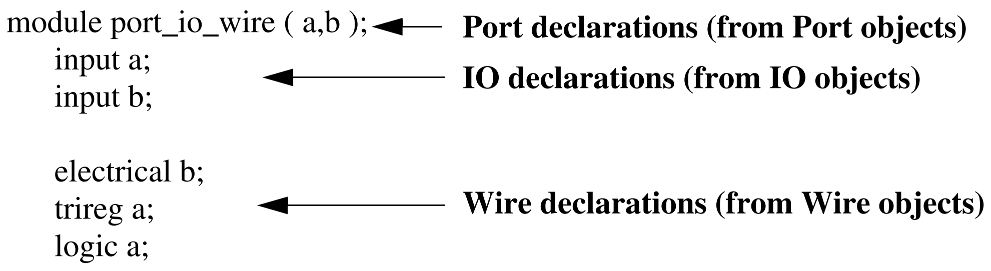
These fields are read-only.

Example

The following netlist example shows the relationship between port, IO, and wire objects.



Here, `a` and `b` are available as port, IO, and wire objects. The wire objects for `a` and `b` contain the type and discipline properties. The complete port list, IO list and wire list is as follows.



```
module port_io_wire ( a,b );
    input a;
    input b;

    electrical b;
    trireg a;
    logic a;
```

Port declarations (from Port objects)
IO declarations (from IO objects)
Wire declarations (from Wire objects)

Alias Object

The alias object contains the information required to alias two signals.

The alias objects are created before calling the `aliasesProc` netlisting procedure. The alias objects of a cellview exist during the lifetime of the cellview object and are destroyed when the cellview object is destroyed. As a consequence, to view the fields, you need to ask for the information during the netlisting run. You might, for example, use statements like the following in your netlisting procedures override file to obtain a list of the fields.

```
(defun MYPrintAliases (formatterId cvId)
   cellview_aliases = cvId->aliases
   ;; Consider each alias object
   (foreach alias cellview_aliases
      (printf "Alias fields: %L\n" alias->?)
   );; foreach
   ;; Call the default print aliases function.
   amsPrintAliases(formatterId cvId)
   );;defun
;; Set up the custom netlist procedure
netId = amsGetNetlister()
vlog = netId->vlog
vlog->aliasesProc = 'MYPrintAliases
```

Running these statements, you find that the alias object contains the following fields.

Field	Value	Meaning of the Value
`assocs`	*`l_associations`*	Each association contains two lists, the source component and the equivalent destination component. Each component includes the base name of that component and the start and stop indices (which are both nil for scalars).
`width`	*`x_width`*	Width (number of bits) in the alias.
`name`	*`t_instName`*	Name used for the alias instance.
`srcBundle`	*`boolean`*	Whether the source connection is a bundle.
`dstBundle`	*`boolean`*	Whether the destination connection is a bundle.
`attributes`	*`l_attributes`*	Collected indirectly. The attributes of the alias object.

These fields are read-only.

Aliases return broken down information which is most useful if you want to use a non-default scheme to alias two signals. If all you want to do is print an existing alias in the default manner, simply call the default `amsPrintAliases` function. If you want to print extra aliases, or delete existing aliases, you can use the `amsPrintAlias` helper function to print the aliases that you need.

Attribute Object

The attribute object holds information that AMS Designer uses to create attributes. Attributes are used to pass data to the AMS elaborator and simulator that otherwise cannot be passed using the Verilog-AMS language itself. The attributes translate as special instructions to the elaborator and simulator and, in almost all cases, are required to make the simulator behave properly.

Attribute objects are created along with the object they belong to, they exist throughout the lifetime of their owner object, and are destroyed when their owner is destroyed. As a consequence, to view the fields, you need to ask for the information during the netlisting run. You might, for example, use statements like the following in your netlisting procedures override file to obtain a list of the fields.

```
(defun MYPrintAttributes (formatterId objectId)
   object_attributes = objectId->attributes
   ;; Consider each attribute object
   (foreach attribute object_attributes
      (printf "Attribute fields: %L\n" attribute->?)
   );; foreach
   ;; Call the default print attributes function.
   amsPrintAttributes(formatterId objectId)
   );;defun
;; Set up the custom netlist procedure
netId = amsGetNetlister()
vlog = netId->vlog
vlog->attributesProc = ‘MYPrintAttributes
```

Running these statements, you find that the attribute object contains the following fields.

Field	Value	Meaning of the Value
`name`	*`string`*	The attribute.
`range`	*`list of integers`*	`x_left x_right`. Nil for scalars.
`value`	*`list of strings`*	The length of list matches the value of `range`.

These fields are read-only.

The attributes supported for AMS Designer are:

Attribute	Purpose
`library_binding`	Specifies the library of the component.
`view_binding`	Specifies the view of the component.
`elaboration_binding`	Specifies the original name of the iterated instance.
`groundSensitivity`	Specifies the ground sensitivity attributes.
`supplySensitivity`	Specifies the supply sensitivity attributes.
`inh_conn_prop_name`	Determines the name of the property, from the net expression.
`inh_conn_def_value`	Determines the name of the default net, from the net expression.
`cds_net_set`	Specifies an array of strings that identifies the netSet properties.
`passed_mfactor`	Identifies the parameter on the instance that carries an m-factor value for the instance.

A
Variables for ams.env Files

The Virtuoso® AMS Designer environment creates a temporary `ams.env` file in the netlist directory. The variables and values in `ams.env` files specify the basic behavior of the AMS netlister and AMS Designer. In general, you have no reason to edit this file.

This appendix contains the following sections:

List of ams.env Variables

The variables that you can use in `ams.env` are all included in the default `ams.env` file. In each entry of the `ams.env` file, the first column is the application, the second column is the variable, the third column is the data type, and the fourth column contains the value to be used. For additional information about individual variables, see “Detailed Descriptions of ams.env Variables” on page 362.

The default `ams.env` file contains the following entries.

```
amsDirect         amsCompMode              boolean   nil
amsDirect         amsDefinitionViews       string    ""
amsDirect         amsExcludeParams         string    ""
amsDirect         amsExpScalingFactor      cyclic    "no"
amsDirect         amsLSB_MSB               boolean   nil
amsDirect         amsMaxErrors             int       50
amsDirect         amsScalarInstances       boolean   t
amsDirect         amsVerbose               boolean   nil
amsDirect         artistStateDirectory     string    "~/.artist_states"
amsDirect         confirmADEStateImport    boolean   t
amsDirect         defaultRunDir            string    ""
amsDirect         hdlVarFile               string    ""
amsDirect         implicitTmpDir           string    ""
amsDirect         includeInstCdfParams     boolean   nil
amsDirect         initFile                 string    ""
amsDirect         logFileName              string    "ams_direct.log"
amsDirect         modifyParamScope         cyclic    "no"
amsDirect         netlistToRunDir          boolean   nil
amsDirect         useRunDirNetlistsOnly    boolean   t
amsDirect         useEffectiveCDF          boolean   nil
amsDirect         simRunDirLoc             string    ""
amsDirect.prep    allowUndefParams         boolean   t
amsDirect.prep    analogControlFile        string    ""
amsDirect.prep    cdsGlobalsLib            string    ""
amsDirect.prep    cdsGlobalsView           string    ""
amsDirect.prep    compileExcludeLibs       string    ""
amsDirect.prep    compileMode              cyclic    "incremental"
amsDirect.prep    connectRulesCell         string    "mixedsignal"
amsDirect.prep    connectRulesCell2        string    "ConnRules_5V_full"
amsDirect.prep    connectRulesLib          string    ""
amsDirect.prep    connectRulesView         string    ""
amsDirect.prep    detailedDisciplineRes    boolean   nil
amsDirect.prep    discipline               string    "logic"
amsDirect.prep    forceGlobalSync          boolean   nil
amsDirect.prep    language                 string    "verilog"
amsDirect.prep    ncelabArguments          string    ""
amsDirect.prep    ncsimArguments           string    ""
amsDirect.prep    ncsimGUI                 boolean   t
amsDirect.prep    ncsimTcl                 boolean   nil
amsDirect.prep    netlistMode              cyclic    "incremental"
amsDirect.prep    runNcelab                boolean   t
amsDirect.prep    runNcsim                 boolean   t
amsDirect.prep    simVisScriptFile         string    ""
amsDirect.prep    timescale                string    "1ns/1ns"
amsDirect.prep    use5xForVHDL             boolean   t
amsDirect.prep    useNcelabNowarn          boolean   t
amsDirect.prep    useNcelabSdfCmdFile      boolean   t
amsDirect.prep   useNcsimNowarn          boolean   t
amsDirect.prep   wfFilter                boolean   nil
amsDirect.prep   useSimVisScriptFile     boolean   t
amsDirect.prep   vlogGroundSigs          string    "gnd!"
amsDirect.prep   vlogSupply0Sigs         string    ""
amsDirect.prep   vlogSupply1Sigs         string    ""
amsDirect.prep   wfDefaultDatabase       string    "waves"
amsDirect.prep   wfDefInstCSaveAll       boolean   nil
amsDirect.prep   wfDefInstCSaveLvl       int       1
amsDirect.prep   wfDefInstSaveCurrents   boolean   nil
amsDirect.prep   wfDefInstSaveVoltages   boolean   t
amsDirect.prep   wfDefInstVSaveAll       boolean   nil
amsDirect.prep   wfDefInstVSaveLvl       int       1
amsDirect.prep   wfDefInstVSaveObjects   cyclic    "All_data"
amsDirect.prep   wfFilterSpec            cyclic    "none"
amsDirect.prep   ncelabAccess            cyclic    "Read"
amsDirect.prep   ncelabAfile             string    ""
amsDirect.prep   ncelabAnnoSimtime       boolean   nil
amsDirect.prep   ncelabCoverage          boolean   nil
amsDirect.prep   ncelabDelayMode         cyclic    "None"
amsDirect.prep   ncelabDelayType         cyclic    "None"
amsDirect.prep   ncelabDisableenht       boolean   nil
amsDirect.prep   ncelabEpulseFiltering   cyclic    "None"
amsDirect.prep   ncelabEpulseNeg         boolean   nil
amsDirect.prep   ncelabExpand            boolean   nil
amsDirect.prep   ncelabExtendtcheckdatalimit  int  0
amsDirect.prep   ncelabExtendtcheckreferencelimit   int   0
amsDirect.prep   ncelabGenafile          string    ""
amsDirect.prep   ncelabIeee1634          boolean   nil
amsDirect.prep   ncelabInterconnmultisrc boolean   nil
amsDirect.prep   ncelabLibverbose        boolean   nil
amsDirect.prep   ncelabLoadpli1          string    ""
amsDirect.prep   ncelabLoadvpi           string    ""
amsDirect.prep   ncelabLogFileAction     cyclic    "Overwrite log file"
amsDirect.prep   ncelabMaxErrors         int       50
amsDirect.prep   ncelabMessages          boolean   nil
amsDirect.prep   ncelabMixEsc            boolean   nil
amsDirect.prep   ncelabModelFilePaths    string    ""
amsDirect.prep   ncelabmodelIncDirs      string    ""
amsDirect.prep   ncelabNeverwarn         boolean   nil
amsDirect.prep   ncelabNoautosdf         boolean   nil
amsDirect.prep   ncelabNocopyright       boolean   nil
amsDirect.prep   ncelabNoipd             boolean   nil
amsDirect.prep   ncelabNonegtchk         boolean   nil
amsDirect.prep   ncelabNonotifier        boolean   nil
amsDirect.prep   ncelabNosource          boolean   nil
amsDirect.prep   ncelabNostdout          boolean   nil
amsDirect.prep   ncelabNoTchkMsg         boolean   nil
amsDirect.prep   ncelabNoTchkXgen        boolean   nil
amsDirect.prep   ncelabNotimingchecks    boolean   nil
amsDirect.prep   ncelabNovitalaccl       boolean   t
amsDirect.prep   ncelabNoVpdmsg          boolean   nil
amsDirect.prep   ncelabNoVpdXgen         boolean   nil
amsDirect.prep   ncelabNowarn            string    ""
amsDirect.prep   ncelabNtcWarn           boolean   nil
amsDirect.prep   ncelabOmicheckl vl      cyclic    "Standard"
amsDirect.prep   ncelabPathpulse         boolean   nil
amsDirect.prep   ncelabPlinooptwarn      boolean   nil
amsDirect.prep   ncelabPlinowarn         boolean   nil
amsDirect.prep   ncelabPresrvResFn       boolean   nil
amsDirect.prep   ncelabPulseE            int       100
amsDirect.prep   ncelabPulseIntE          int       100
amsDirect.prep   ncelabPulseIntR          int       100
amsDirect.prep   ncelabPulseR             int       100
amsDirect.prep   ncelabRelax              boolean   nil
amsDirect.prep   ncelabSdfCmdFile         string    ""
amsDirect.prep   ncelabSdfNocheckCelltype boolean  nil
amsDirect.prep   ncelabSdfNoHeader        boolean   nil
amsDirect.prep   ncelabSdfNoWarnings      boolean   nil
amsDirect.prep   ncelabSdfprecision       string    ""
amsDirect.prep   ncelabSdfverbose         boolean   nil
amsDirect.prep   ncelabSdfWorstcaseRounding boolean  nil
amsDirect.prep   ncelabsolverInfo         string    "Spectre"
amsDirect.prep   ncelabStatus             boolean   t
amsDirect.prep   ncelabTopLvlGeneric      string    ""
amsDirect.prep   ncelabUpdate             boolean   t
amsDirect.prep   ncelabUseAddArgs         boolean   nil
amsDirect.prep   ncelabUseAfile           boolean   nil
amsDirect.prep   ncelabUseExtendtcheckdatalimit        boolean   nil
amsDirect.prep   ncelabUseExtendtcheckreferencelimit   boolean   nil
amsDirect.prep   ncelabUseGenafile        boolean   nil
amsDirect.prep   ncelabUseGeneric         boolean   nil
amsDirect.prep   ncelabUsePulseE          boolean   nil
amsDirect.prep   ncelabUsePulseIntE       boolean   nil
amsDirect.prep   ncelabUsePulseIntR       boolean   nil
amsDirect.prep   ncelabUsePulseR          boolean   nil
amsDirect.prep   ncelabUseSdfprecision    boolean   nil
amsDirect.prep   ncelabV93                boolean   nil
amsDirect.prep   ncelabVipdelay           cyclic    "Typical"
amsDirect.prep   ncsimEpulseNoMsg         boolean   nil
amsDirect.prep   ncsimExtassertmsg        boolean   nil
amsDirect.prep   ncsimLoadvpi             string    ""
amsDirect.prep   ncsimLogFileAction       cyclic    "Overwrite log file"
amsDirect.prep   ncsimMaxErrors           int       50
amsDirect.prep   ncsimMessages            boolean   nil
amsDirect.prep   ncsimNeverwarn           boolean   nil
amsDirect.prep   ncsimNocifcheck          boolean   nil
amsDirect.prep   ncsimNosource            boolean   nil
amsDirect.prep   ncsimNostdout            boolean   nil
amsDirect.prep   ncsimNowarn              string    ""
amsDirect.prep   ncsimOmichecklvl         cyclic    "None"
amsDirect.prep   ncsimPlinooptwarn        boolean   nil
amsDirect.prep   ncsimPlinowarn           boolean   nil
amsDirect.prep   ncsimProfile             boolean   nil
amsDirect.prep   ncsimProfthread          boolean   nil
amsDirect.prep   ncsimRedmem              boolean   nil
amsDirect.prep   ncsimStatus              boolean   nil
amsDirect.prep   ncsimUnbuffered          boolean   nil
amsDirect.prep   ncsimUpdate              boolean   t
amsDirect.prep   ncsimUseAddArgs          boolean   nil
amsDirect.simcntl   dcop                  boolean   nil
amsDirect.simcntl   paramRangeCheckFile   string    ""
amsDirect.simcntl   scaddlglblopts        string    ""
amsDirect.simcntl   scaddltranopts        string    ""
amsDirect.simcntl   scglobalminr          string    "0.0"
amsDirect.simcntl   scannotate            cyclic    "status"
amsDirect.simcntl   scapprox              boolean   nil
amsDirect.simcntl   scaudit               cyclic    "detailed"
amsDirect.simcntl   sccheckstmt           cyclic    "all"
amsDirect.simcntl   sccmin                string    "0.0"
amsDirect.simcntl   sccompatible          cyclic    "spectre"
amsDirect.simcntl   scdebug               boolean   nil
```

amsDirect.simcntl	scdiagnose	boolean	nil
amsDirect.simcntl	scdigits	int	5
amsDirect.simcntl	scerror	boolean	t
amsDirect.simcntl	scerrpreset	cyclic	"moderate"
amsDirect.simcntl	scfastbreak	boolean	nil
amsDirect.simcntl	scgmin	string	"1e-12"
amsDirect.simcntl	scgmincheck	cyclic	"max_v_only"
amsDirect.simcntl	schomotopy	cyclic	"all"
amsDirect.simcntl	sciabstol	string	"1e-12"
amsDirect.simcntl	scic	cyclic	"all"
amsDirect.simcntl	scicstmt	string	""
amsDirect.simcntl	scignshorts	boolean	nil
amsDirect.simcntl	scinfo	boolean	t
amsDirect.simcntl	scinventory	cyclic	"detailed"
amsDirect.simcntl	sclimit	cyclic	"dev"
amsDirect.simcntl	sclteratio	string	""
amsDirect.simcntl	scmacromod	boolean	nil
amsDirect.simcntl	scmaxiters	int	5
amsDirect.simcntl	scmaxnotes	int	5
amsDirect.simcntl	scmaxnotestologfile	int	5
amsDirect.simcntl	scmaxrsd	string	""
amsDirect.simcntl	scmaxstep	string	""
amsDirect.simcntl	scmaxwarn	int	5
amsDirect.simcntl	scmaxwarntologfile	int	5
amsDirect.simcntl	scmethod	cyclic	"<Default value>"
amsDirect.simcntl	scnarrate	boolean	t
amsDirect.simcntl	scnotation	cyclic	"eng"
amsDirect.simcntl	scnote	boolean	t
amsDirect.simcntl	scopptcheck	boolean	t
amsDirect.simcntl	scpivabs	string	"0.0"
amsDirect.simcntl	scpivotdc	boolean	nil
amsDirect.simcntl	scpivrel	string	"1e-3"
amsDirect.simcntl	scquantities	cyclic	"no"
amsDirect.simcntl	screadic	string	""
amsDirect.simcntl	screadns	string	""
amsDirect.simcntl	screlref	cyclic	"<Default value>"
amsDirect.simcntl	screltol	string	""
amsDirect.simcntl	scrforce	string	"1.0"
amsDirect.simcntl	scscale	int	1
amsDirect.simcntl	scscalem	int	1
amsDirect.simcntl	scalem	string	"1.0"
amsDirect.simcntl	scale	string	"1.0"
amsDirect.simcntl	scmodelevaltype	cyclic	"s"
amsDirect.simcntl	scmosvres	string	"0.05"
amsDirect.simcntl	scscfincfile	string	""
amsDirect.simcntl	scscftimestamp	string	""
amsDirect.simcntl	scscfusefileflag	boolean	nil
amsDirect.simcntl	scskipcount	int	0
amsDirect.simcntl	scskipdc	cyclic	"no"
amsDirect.simcntl	scskipstart	string	"0.0"
amsDirect.simcntl	scskipstop	string	"0.0"
amsDirect.simcntl	scspeed	int	0
amsDirect.simcntl	scspscflag	boolean	nil
amsDirect.simcntl	scstats	boolean	nil
amsDirect.simcntl	scstep	string	""
amsDirect.simcntl	scstop	string	"0.0"
amsDirect.simcntl	scstrobedelay	string	"0.0"
amsDirect.simcntl	scstrobeperiod	string	"0.0"
amsDirect.simcntl	sctemp	string	"27.0"
amsDirect.simcntl	sctempeffects	cyclic	"all"
amsDirect.simcntl	sctitle	string	""

amsDirect.simcntl	sctnom	string	"27.0"
amsDirect.simcntl	sctopcheck	cyclic	"full"
amsDirect.simcntl	sctransave	cyclic	"allpub"
amsDirect.simcntl	scusemodeleval	boolean	nil
amsDirect.simcntl	scvabstol	string	"1e-6"
amsDirect.simcntl	scwarn	boolean	t
amsDirect.simcntl	scwrite	string	""
amsDirect.simcntl	scwritefinal	string	""
amsDirect.simcntl	simcompat	cyclic	"spectre"
amsDirect.simcntl	start	string	"0.0"
amsDirect.simcntl	useScaddlglblopts	boolean	t
amsDirect.simcntl	useScaddltranopts	boolean	t
amsDirect.simcntl	useScic	boolean	t
amsDirect.simcntl	useScreadic	boolean	t
amsDirect.simcntl	useScreadns	boolean	t
amsDirect.simcntl	useScscfincfile	boolean	t
amsDirect.simcntl	useScwrite	boolean	t
amsDirect.simcntl	useScwritefinal	boolean	t
amsDirect.simcntl	usimAbstoli	string	"1e-12"
amsDirect.simcntl	usimAbstolv	string	"1e-6"
amsDirect.simcntl	usimAddlOptions	string	""
amsDirect.simcntl	usimAnalog	cyclic	"Default"
amsDirect.simcntl	usimCapFile	string	""
amsDirect.simcntl	usimCgnd	string	"1e-20"
amsDirect.simcntl	usimCgndr	string	"0"
amsDirect.simcntl	usimDCMethod	cyclic	"Complete DC"
amsDirect.simcntl	usimDcut	boolean	nil
amsDirect.simcntl	usimDcutField	string	""
amsDirect.simcntl	usimDiodeMethod	cyclic	"Analog table"
amsDirect.simcntl	usimDpfFile	string	""
amsDirect.simcntl	usimDumpStep	string	""
amsDirect.simcntl	usimenableNA	boolean	nil
amsDirect.simcntl	usimenablePA	boolean	nil
amsDirect.simcntl	usimenableRA	boolean	nil
amsDirect.simcntl	usimenableTA	boolean	nil
amsDirect.simcntl	usimLshort	string	"0.0"
amsDirect.simcntl	usimLvshort	string	"0.0"
amsDirect.simcntl	usimMaxstep	string	""
amsDirect.simcntl	usimMaxstepStart	string	"0.0"
amsDirect.simcntl	usimMaxstepStop	string	"0.0"
amsDirect.simcntl	usimMaxstepSubckt	string	""
amsDirect.simcntl	usimMosMethod	cyclic	"Analog/MS table"
amsDirect.simcntl	usimNALimit	string	"0"
amsDirect.simcntl	usimNAOutputSort	cyclic	"max_vo"
amsDirect.simcntl	usimNASortIs	cyclic	"inc"
amsDirect.simcntl	usimOutputStart	string	"0.0"
amsDirect.simcntl	usimPostl	cyclic	"No RCR"
amsDirect.simcntl	usimRAAgeDomain	cyclic	"loglog"
amsDirect.simcntl	usimRAAgeMethod	cyclic	"interp"
amsDirect.simcntl	usimRAAgeproc	string	""
amsDirect.simcntl	usimRAAgingTime	string	"10y"
amsDirect.simcntl	usimRADeltaD	string	"0.1"
amsDirect.simcntl	usimRADeltaDToggle	boolean	t
amsDirect.simcntl	usimRAMinAge	string	"0.0"
amsDirect.simcntl	usimRAMode	cyclic	"HCI only"
amsDirect.simcntl	usimRANBTIAgeproc	string	""
amsDirect.simcntl	usimRcrfmax	string	"1e9"
amsDirect.simcntl	usimRshort	string	"1e-6"
amsDirect.simcntl	usimRvshort	string	"1e-6"
amsDirect.simcntl	usimSimMode	cyclic	"Mixed signal"
amsDirect.simcntl	usimSpeed	int	0

```
amsDirect.simcntl   usimSpefFile         string    ""
amsDirect.simcntl   usimSpfFile          string    ""
amsDirect.simcntl   usimTol              string    "0.01"
amsDirect.simcntl   usimTranAddlOptions  string    ""
amsDirect.simcntl   usimUseAddlOptions   boolean   t
amsDirect.simcntl   usimVcdFile          string    ""
amsDirect.simcntl   usimVcdInfoFile      string    ""
amsDirect.simcntl   usimVectorFile       string    ""
amsDirect.simcntl   usimWFAbstoli        string    "1e-12"
amsDirect.simcntl   usimWFAbstolv        string    "1e-6"
amsDirect.simcntl   usimWFFilter         boolean    t
amsDirect.simcntl   usimWFReltol         string    "0.0"
amsDirect.simcntl   usimWFTres           string    "1e-12"
amsDirect.vlog   allowDeviantBuses       cyclic    "no"
amsDirect.vlog   allowIllegalIdentifiers cyclic    "warn"
amsDirect.vlog   allowNameCollisions     cyclic    "warn"
amsDirect.vlog   allowSparseBuses        cyclic    "warn"
amsDirect.vlog   amsEligibleViewTypes    string    "schematic"
amsDirect.vlog   checkAndNetlist         boolean   nil
amsDirect.vlog   checkOnly               boolean   nil
amsDirect.vlog   compileAsAMS            boolean   t
amsDirect.vlog   excludeViewNames        string    ""
amsDirect.vlog   headerText              cyclic    "none"
amsDirect.vlog   ifdefLanguageExtensions boolean   nil
amsDirect.vlog   includeFiles            string    "(disciplines.vams)"
amsDirect.vlog   ncvlogArguments         string    ""
amsDirect.vlog   netlistAfterCdfChange   boolean   nil
amsDirect.vlog   paramDefVals            string    ""
amsDirect.vlog   paramGlobalDefVal       string    "0"
amsDirect.vlog   processViewNames        string    ""
amsDirect.vlog   prohibitCompile         boolean   nil
amsDirect.vlog   templateFile            string    ""
amsDirect.vlog   templateScript          string    ""
amsDirect.vlog   useDefparam             boolean   nil
amsDirect.vlog   useNowarn               boolean   t
amsDirect.vlog   useProcessViewNamesOnly boolean   nil
amsDirect.vlog   verboseUpdate           boolean   t
amsDirect.vlog   checktasks              boolean   nil
amsDirect.vlog   errOutInconsistentMasters    boolean   nil
amsDirect.vlog   ieee1364                boolean   nil
amsDirect.vlog   ignoreIllegalCDFParams  boolean   nil
amsDirect.vlog   noline                  boolean   nil
amsDirect.vlog   incdir                  string    ""
amsDirect.vlog   lexpragma               boolean   nil
amsDirect.vlog   logFileAction           cyclic    "Overwrite log file"
amsDirect.vlog   macro                   string    ""
amsDirect.vlog   markcelldefines         boolean   nil
amsDirect.vlog   netlistUDFAsMacro       boolean   nil
amsDirect.vlog   bindCdsAliasLib         boolean   t
amsDirect.vlog   bindCdsAliasView        boolean   t
amsDirect.vlog   maxErrors               int       50
amsDirect.vlog   messages                boolean   nil
amsDirect.vlog   neverwarn               boolean   nil
amsDirect.vlog   nomempack               boolean   nil
amsDirect.vlog   nopragmawarn            boolean   nil
amsDirect.vlog   nostdout                boolean   nil
amsDirect.vlog   nowarn                  string    ""
amsDirect.vlog   pragma                  boolean   nil
amsDirect.vlog   status                  boolean   nil
amsDirect.vlog   update                  boolean   t
amsDirect.vlog   vloglinedebug           boolean   nil
```

amsDirect.vlog	ncvlogUseAddArgs	boolean	nil
amsDirect.vlog	iterInstExpFormat	string	"%b_%i"
amsDirect.vlog	netClashFormat	string	"%b_netclash"
amsDirect.vlog	instClashFormat	string	"%b_instclash"
amsDirect.vlog	aliasInstFormat	string	"ams_alias_inst_%i"

Detailed Descriptions of ams.env Variables

Details for these `ams.env` file variables appear alphabetically, by variable name, in the sections that follow.

aliasInstFormat

Specifies the format to be used to create instances of the `cds_alias` module.

Syntax

`amsDirect.vlog aliasInstFormat string "`*`format`*`"`

Value

`format`	All characters, except those listed below, are printed exactly as included in *`format`*. The following characters have the indicated special meanings.

`%i`	Index number of the current `cds_alias` instance
`%%`	Prints the `%` character

The default value of *`format`* is `ams_alias_inst_%i`, which produces names such as `ams_alias_inst_1`, `ams_alias_inst_2`, and so on.

If the resulting name is illegal in Verilog-AMS, the name is mapped. If the mapped name clashes with the name of another object, the name undergoes collision mapping.

Example

```
amsDirect.vlog aliasInstFormat string "cds_alias_%i"
```

Tells AMS netlister to create instance names with a suffixed index number. In this example, instances of the `cds_alias` module are given names like

```
cds_alias_1
cds_alias_2
cds_alias_3
```

allowDeviantBuses

Controls the netlisting of bus specifications when there are conflicting bus ranges. Bus ranges conflict when, in references to the same bus, the indexes sometimes go from smaller to larger and other times go from larger to smaller.

Syntax

`amsDirect.vlog allowDeviantBuses cyclic "no" | "warn" | "yes`

Values

`no`	Netlisting halts immediately when the AMS netlister encounters conflicting bus ranges. This is the default. This value corresponds to the *No – Print Errors* value used in the graphical user interface (GUI).
`warn`	Netlisting continues when the AMS netlister encounters conflicting bus ranges if it is possible to create a valid netlist. The AMS netlister tells you how the non-compliant bus data is transformed. The generated netlist is likely to be less readable than one created from compliant bus data. This value corresponds to the *Yes – Print Warnings* value used in the GUI.
`yes`	Netlisting continues when the AMS netlister encounters conflicting bus ranges if it is possible to create a valid netlist. The AMS netlister does not issue a warning. This value corresponds to the *Yes – Silently* value used in the GUI.

Example

Here is an example of conflicting bus ranges:

```
a<0:7>
a<7:6>
a<5:0>
a<2:4>
```

Here is the same example in Verilog-AMS:

```
a[0:7]
{a[7],a[6]}
{a[5],a[4],a[3],a[2],a[1],a[0]}
a{2:4}
```

Using the variable

```
amsDirect.vlog allowDeviantBuses cyclic "yes"
```

tells the AMS netlister to handle conflicting bus ranges whenever possible, without issuing a warning. This example sets the netlisting behavior for data netlisted into the Verilog®-AMS language.

allowIllegalIdentifiers

Controls the netlisting of non-compliant identifiers.

Syntax

```
amsDirect.vlog allowIllegalIdentifiers cyclic "no" | "warn" | "yes
```

Values

`no`	Netlisting halts immediately when the AMS netlister encounters a non-compliant identifier. This value corresponds to the *No – Print Errors* value used in the graphical user interface (GUI).
`warn`	Maps non-compliant identifiers to names that are legal in the target language and issues a warning telling you how the name is mapped. This is the default. This value corresponds to the *Yes – Print Warnings* value used in the GUI.
`yes`	Maps non-compliant identifiers to names that are legal in the target language. The AMS netlister does not issue a warning. This value corresponds to the *Yes – Silently* value used in the GUI.

Description

If you specify `warn` or `yes`, the AMS netlister maps non-compliant identifiers to the target language. However, mapping identifiers results in a less readable netlist.

Identifiers are non-compliant if one or more of the following situations applies:

- Identifiers do not follow the syntax required by the netlist language you plan to use
- Identifiers are reserved words in the netlist language

  For a list of Verilog-AMS reserved words, see the "Verilog-AMS Keywords" appendix in the *Cadence Verilog-AMS Language Reference*.
- Identifiers do not map cleanly to the netlist language
- Identifiers are not unique within the design

Because the determination of non-compliance depends on the target netlist language, it is possible to have identifiers that are compliant for one target language and non-compliant for another. To ensure that identifiers are compliant for every target netlist language, use the following syntax.

```
basic_identifier ::=
      letter {[_] letter_or_digit}

letter_or_digit ::=
      a-z | 0-9
```

For example, the following identifiers are compliant for every target language.

```
an_identifier_name
a_2nd_name
a_name2
```

The following identifiers, because they do not use the suggested syntax, might be non-compliant for some target languages.

```
2identifier      // Should begin with a letter.
My_identifer     // Should not use uppercase letters.
an_identifier_   // Should end with a letter or digit.
a&b              // Should not use characters other than a-z, 0-9, and underscore.
```

allowNameCollisions

Controls the netlisting of names that do not comply with AMS Designer environment guidelines because they are not unique.

Syntax

```
amsDirect.vlog allowNameCollisions cyclic "no" | "warn" | "yes
```

Values

`no`	Netlisting halts immediately when the AMS netlister encounters a non-unique name. This value corresponds to the *No – Print Errors* value used in the graphical user interface (GUI).
`warn`	Maps non-unique names to system-generated names that are legal in the target language, and issues a warning. This is the default. This value corresponds to the *Yes – Print Warnings* value used in the GUI.
`yes`	Maps non-unique names to system-generated names that are legal in the target language. The AMS netlister does not issue a warning.This value corresponds to the *Yes – Silently* value used in the GUI.

Description

To comply with AMS Designer environment guidelines, each instance, cell, terminal, parameter, and net in your design must have a unique name. If the names of these components are not unique, the AMS netlister acts as shown in the table below.

How Verilog-AMS Handles Non-Unique Identifiers

Objects sharing a name	AMS netlister action
module terminal, cell	No mapping occurs, and netlisting proceeds normally
parameter, module terminal	Netlisting fails
instance terminal, parameter of the same instance	No mapping occurs, and a warning is issued.
parameter, cell	No mapping occurs, and netlisting proceeds normally

How Verilog-AMS Handles Non-Unique Identifiers, *continued*

Objects sharing a name	AMS netlister action
net, parameter	Net identifier maps to `*netName*_netclash`
net, module terminal	Net identifier maps to `*netName*_netclash`. (However, no mapping occurs when the net and module terminal are connected to each other.)
net, cell	Net identifier maps to `*netName*_netclash`
instance, net	Instance identifier maps to `*instName*_instclash`
instance, parameter	Instance identifier maps to `*instName*_instclash`
instance, module terminal	Instance identifier maps to `*instName*_instclash`
instance, cell	Instance identifier maps to `*instName*_instclash`

allowSparseBuses

Controls the netlisting of sparse buses.

Syntax

`amsDirect.vlog allowSparseBuses cyclic "no" | "warn" | "yes"`

Values

`no`	Netlisting halts immediately when the AMS netlister encounters a sparse bus. This value corresponds to the *No – Print Errors* value used in the graphical user interface (GUI).
`warn`	Overdeclares any sparse buses and issues a warning. This is the default. This value corresponds to the *Yes – Print Warnings* value used in the GUI.
`yes`	Overdeclares any sparse buses. The AMS netlister does not issue a warning. This value corresponds to the *Yes – Silently* value used in the GUI.

Description

Sparse buses do not comply with AMS Designer environment guidelines because you must declare buses as a contiguous vector of bits before they are used in Verilog-AMS. If you specify `warn` or `yes`, the AMS netlister overdeclares sparse buses so it can continue netlisting.

Example

Here is an example of a sparse bus:

```
b<5:0:2>
```

which is the same as

```
b<5>, b<3>, b<1>
```

Using the variable

```
amsDirect.vlog allowSparseBuses cyclic "yes"
```

tells the AMS netlister to handle sparse buses whenever possible, without issuing a warning. In this example, the AMS netlister overdeclares this bus in order to continue netlisting:

```
module XXX (.b({b[5],,b{3],,b[1]}), ...);
    input [5:1] b;

    ...
```

allowUndefParams

Controls whether undeclared parameters can be overridden.

Syntax

`amsDirect.prep allowUndefParams boolean t | nil`

Values

`t`	The elaborator allows undeclared parameters to be overridden. This is the default.
`nil`	The elaborator stops when it encounters a value override for an undeclared parameter.

Description

By default, the elaborator reports an error and stops when it encounters a value override for an undeclared parameter. Specifying `t` for the `allowUndefParams` variable tells the elaborator to allow undeclared parameters to be overridden.

Example

```
amsDirect.prep allowUndefParams boolean t
```

Tells the elaborator to permit overriding the values of undeclared parameters, such as by using a `defparam` statement or by overriding the value when an instance is declared.

amsCompMode

Controls whether the AMS Designer environment supports certain properties used in legacy VHDL modules. Note, however, that the `amsCompMode` variable is not supported in this release.

Syntax

`amsDirect amsCompMode boolean t | nil`

Values

`t`	Specifies that certain properties used in legacy VHDL modules are to be supported by the AMS Designer environment.
`nil`	Specifies that certain properties used in legacy VHDL modules are *not* to be supported by the AMS Designer environment.

Description

The following legacy properties are supported by the AMS Designer environment if the `amsCompMode` variable is set to `t`. If the variable is set to `nil`, the properties are ignored and omitted from the netlist.

- `vhdlAttributeDefList`
- `vhdlComponentDecl`
- `vhdlFormalPortFuncName`
- `vhdlPackageComponents`
- `vhdlPackageNames`

amsDefinitionViews

Specifies a list of views that can be used to determine the vectored terminal range direction and terminal order for cellviews being netlisted. This capability is useful when the cellview being netlisted needs to be netlisted in accordance with another view of the cell, such as the placed master. AMS Designer does not provide a graphical interface for setting this variable.

To use the `amsDefinitionViews` list, the netlister

1. Determines whether there is a `termOrder` property for the cellview being netlisted. If so, that property determines the vectored terminal range direction and terminal order and the `amsDefinitionViews` list has no effect.
2. Determines whether the first listed view exists. If it does, no more views are considered. If the first view does not exist, the search through the list continues until the netlister finds a view that exists or reaches the end of the list.
3. If the identified existing view has a `portOrder` property, uses that information to determine the vectored terminal range direction and terminal order of the cellview being netlisted. If the `portOrder` property does not exist, the netlister checks the view for vectored terminals used in their entirety and uses that ordering. If the ordering is still not determined for one or more terminals, the ordering specified by the `amsLSB_MSB` environment variable is used.
4. If none of the listed views exists, uses the `portOrder` property of the cellview being netlisted (if that cellview has a `portOrder` property) to determine the vectored terminal range direction and terminal order. If the `portOrder` property does not exist, the netlister checks the cellview being netlisted for vectored terminals used in their entirety and uses that ordering. If the ordering is still not determined for one or more terminals, the ordering specified by the `amsLSB_MSB` environment variable is used.

Syntax

```
amsDirect.vlog amsDefinitionViews string "list"
```

Value

list	A string of space-separated views to be consulted for terminal order and vectored terminal range directions. The view names are considered to be in the cellview namespace. Any included views that are created or imported by the CIW must be accompanied by a shadow cellview. The default value is an empty string.

Example

```
amsDirect.vlog amsDefinitionViews string "symbol verilog"
```

amsEligibleViewTypes

Specifies the cellview types that trigger netlisting.

Syntax

`amsDirect.vlog amsEligibleViewTypes string "`*`list`*`"`

Value

`list`	A list of one or more of the following cellview types: `schematic`, `symbolic`, `maskLayout` (extracted view only, based on the last extraction timestamp), and `netlist`. Cellview types must be separated by spaces in the list. If you do not specify a cellview for netlisting (by using the `amsdirect -view` option, for example), the AMS netlister generates netlists for each of the cellview types included in the list. The default for *`list`* is `schematic`.

Example

```
amsDirect.vlog amsEligibleViewTypes string "schematic symbolic"
```

Tells the AMS netlister to netlist `schematic` and `symbolic` cellviews (unless, for example, a view is specified by using the `amsdirect -view` option). This example sets the netlisting behavior for data netlisted into the Verilog-AMS language.

amsExcludeParams

Lists parameters to be omitted from the netlist.

Syntax

`amsDirect amsExcludeParams string "`*`list`*`"`

Value

`list`	A list of parameters that are not to be netlisted. `list` is a string of space-separated parameter names. The default is an empty string.

Example

```
amsDirect amsExcludeParams string "fix unfix"
```

Tells the AMS netlister not to netlist the parameters `fix` and `unfix` when they are found associated with components in this design.

Note that if a cell has valid information in the `ams` section of the CDF simInfo, the contents of the simInfo are always obeyed, regardless of the value of the `amsExcludeParams` variable. For example, for a cell `mycell`, if `param1` and `param2` are in the *instParameters* field of the simInfo and `param1` is also listed in the `amsExcludeParams` variable, then `amsExcludeParams` has no effect. When `mycell` (or any instance of `mycell`) is netlisted, `param1` is always printed.

You can use the *excludeParameters* simInfo field in conjunction with the `amsExcludeParams` `ams.env` variable and the `amsExcludeParams` CDF parameter to precisely specify parameters at the cell, design, and library levels that are not to be netlisted. For more information, see "Excluding Parameters from Netlisting" on page 203.

amsExpScalingFactor

Controls the expansion of scaling factors for parameter values.

Syntax

```
amsDirect amsExpScalingFactor cyclic "no" | "dec" | "sci"
```

Values

`no`	Includes scaling factor suffixes in netlists without expanding them. This is the default.
`dec`	Expands scaling factor suffixes in decimal notation.
`sci`	Expands scaling factor suffixes in scientific notation.

Description

Some simulators do not support scaling factors or support only a subset of the scaling factors used in designs. If the simulator you plan to use is one of these simulators, you can use the `amsExpScalingFactor` variable to expand scaling factors so the factors do not appear in netlists.

The following table shows the scaling factor suffixes and the target simulators that support them.

Scaling Factor Suffixes and Target Simulators

Suffix		Scaling Factor (e^x)	AEL	Verilog-AMS	Spectre	SKILL
`Y`	Yotta	10^{24}	See note below.			
`Z`	Zetta	10^{21}	See note below.			
`T`	Tera	10^{12}	yes	yes	yes	yes
`G`	Giga	10^9	yes	yes	yes	yes
`M`	Mega	10^6	yes	yes	yes	yes
`ME`	Mega	10^6	yes			yes
`K`	Kilo	10^3	yes	yes	yes	yes

Scaling Factor Suffixes and Target Simulators, *continued*

Suffix		Scaling Factor (e^x)	AEL	Verilog-AMS	Spectre	SKILL
`k`	kilo	10^3	yes		yes	yes
`%`	percent	10^{-2}	yes		yes	yes
`c`	percent	10^{-2}			yes	
`m`	milli	10^{-3}	yes	yes	yes	yes
`u`	micro	10^{-6}	yes	yes	yes	yes
`n`	nano	10^{-9}	yes	yes	yes	yes
`p`	pico	10^{-12}	yes	yes	yes	yes
`f`	femto	10^{-15}	yes	yes	yes	yes
`a`	atto	10^{-18}	yes	yes	yes	yes
`z`	zepto	10^{-21}	See note below.			
`y`	yocto	10^{-24}	See note below.			

Note: AMS Designer always expands the `Y`, `Z`, `z`, and `y` scaling factors, using scientific notation, regardless of the value of the `amsExpScalingFactor` variable.

Example

A few examples of expanded scaling factor suffixes are shown below.

5.46T = 5.46e12 = 5,460,000,000,000
5.46G = 5.46e9 = 5,460,000,000
5.46M = 5.46e6 = 5,460,000
5.46K = 5.46e3 = 5,460
5.46% = 5.46e-2 = 0.0546
5.46u = 5.46e-6 = 0.00000546

amsLSB_MSB

Controls the bit order used to netlist a bus when the following conditions are all true:

- The information derived from views listed by the `amsDefinitionViews` environment variable is insufficient to determine the bit order.
- The `portOrder` property of the cellview being netlisted is insufficient to determine the bit order.
- The bus is not used in its entirety anywhere in the cellview being netlisting.

To summarize, the `amsLSB_MSB` variable is used only when the bit order cannot be determined by using the `amsDefinitionViews` variable.

Syntax

```
amsDirect amsLSB_MSB boolean t | nil
```

Values

`t`	Orders the bits as [LSB : MSB] when constructing buses.
`nil`	Orders the bits as [MSB : LSB] when constructing buses. This is the default.

Description

By default, the AMS netlister orders the bits as follows:

[MSB : LSB]

which is most significant bit to least significant bit. Specifying the `t` value for this variable reverses the bit order.

amsMaxErrors

Halts the AMS netlister when it reaches a certain number of errors. If the netlister encounters any design error, it does not produce a netlist.

Syntax

`amsDirect amsMaxErrors int` *`maxErrors`*

Value

`maxErrors`	A positive integer. Halts netlisting after this number of errors occur. The default is 50.

Example

```
amsDirect amsMaxErrors int 12
```

Tells the AMS netlister to halt netlisting when it encounters 12 errors.

amsScalarInstances

Controls the netlisting of iterated instances.

Syntax

```
amsDirect amsScalarInstances boolean t | nil
```

Values

`t`	Scalarizes iterated instances. This is the default.
`nil`	Produces an array of instances.

Description

By default, the AMS netlister scalarizes iterated instances. You can use this variable to produce an array of instances in Verilog-AMS netlists instead.

amsVerbose

Controls whether the netlister issues informational messages.

Syntax

`amsDirect amsVerbose boolean t | nil`

Values

`t`	Places a checkmark next to the *Print informational messages* field on the *Netlister* pane of the AMS Options window. This tells the netlister to issue verbose messages.
`nil`	Removes the checkmark, indicating that verbose messages are not issued while netlisting. This is the default.

Example

```
amsDirect amsVerbose boolean t
```

Removes the checkmark next to the *Print informational messages* field. As a result, verbose messages are not issued during netlisting.

analogControlFile

Specifies the analog simulation control file to be used.

Syntax

`amsDirect.prep analogControlFile string "`*`file`*`"`

Value

`file`	The analog simulation control file to be used. If *`file`* is specified with an absolute path, the analog simulation control file is stored at that location. If *`file`* is specified with a relative path, the path is determined relative to the run directory (not to the current working directory). The default is an empty string, which means that the value that appears in the AMS Run Simulation form is *`runDir`*`/`*`topLevelCell`*`.scs`.

Example

```
amsDirect.prep analogControlFile string "sch.scs"
```

artistStateDirectory

Specifies the directory used to seed the *From ADE state directory* field in the Import from ADE State form. The specified directory is expected to be the top level of the saved states directory structure.

Syntax

`amsDirect artistStateDirectory string "`*`directory`*`"`

Value

`directory`	The path and directory to be used to seed the form. The default is `~/.artist_states`, which is the default directory used by the Analog Design Environment (ADE) for saved states.

Example

```
amsDirect artistStateDirectory string "~/.mystatesdir"
```

bindCdsAliasLib

Adds the `library_binding = "basic"` attribute to instances of the `cds_alias` module that the AMS netlister adds to netlists.

This attribute specifies the library binding for instances of the `cds_alias` module. This specification is necessary when the `basic` library is not included in the Virtuoso Hierarchy Editor *Library List*. Regardless of the setting of the `bindCdsAliasLib` variable, the `basic` library, which contains the `cds_alias` module, must be defined in the `cds.lib` file.

Syntax

`amsDirect.vlog bindCdsAliasLib boolean t | nil`

Values

`t`	Adds the `library_binding = "basic"` attribute to automatically inserted instances of the `cds_alias` module. This is the default.
`nil`	Does not add the `library_binding` attribute to instances of the `cds_alias` module.

Examples

- The variable

  ```
  amsDirect.vlog bindCdsAliasLib boolean t
  ```

  tells the AMS netlister to add the `library_binding` attribute to automatically inserted instances of the `cds_alias` module, producing a statement similar to the following:

  ```
  cds_alias #(.width(1)) (* integer library_binding = "basic"; *)
      ams_alias_inst_0 (net015, net014[0]);
  ```

- The variables

  ```
  amsDirect.vlog bindCdsAliasLib boolean t
  amsDirect.vlog bindCdsAliasView boolean t
  ```

  tell the AMS netlister to add the `library_binding` and `view_binding` attributes to automatically inserted instances of the `cds_alias` module, producing a statement similar to the following:

  ```
  cds_alias #(.width(1)) (* integer library_binding = "basic";
      integer view_binding = "functional"; *)
      ams_alias_inst_0 (net015,net014[0]);
  ```

bindCdsAliasView

Adds the `view_binding = "functional"` attribute to instances of the `cds_alias` module that the AMS netlister adds to netlists.

This attribute specifies the view binding for instances of the `cds_alias` module. This specification is necessary when the `functional` view is not included in the Virtuoso Hierarchy Editor *View List*. Regardless of the setting of the `bindCdsAliasView` variable, the `basic` library, which contains the `cds_alias` module, must be defined in the `cds.lib` file.

Syntax

`amsDirect.vlog bindCdsAliasView boolean t | nil`

Values

`t`	Adds the `view_binding = "functional"` attribute to automatically inserted instances of the `cds_alias` module. This is the default.
`nil`	Does not add the `view_binding` attribute to instances of the `cds_alias` module.

Examples

- amsDirect.vlog bindCdsAliasView boolean t

  Tells the AMS netlister to add the `view_binding` attribute to automatically inserted instances of the `cds_alias` module, producing a statement similar to the following:

  ```
  cds_alias #(.width(1)) (* integer view_binding = "functional"; *)
      ams_alias_inst_0 (net015, net014[0]);
  ```

- The variables

  ```
  amsDirect.vlog bindCdsAliasLib boolean t
  amsDirect.vlog bindCdsAliasView boolean t
  ```

  tell the AMS netlister to add the `library_binding` and `view_binding` attributes to automatically inserted instances of the `cds_alias` module, producing a statement similar to the following:

  ```
  cds_alias #(.width(1)) (* integer library_binding = "basic";
      integer view_binding = "functional"; *)
      ams_alias_inst_0 (net015,net014[0]);
  ```

cdsGlobalsLib

Specifies the library to hold the `cds_globals` module created by AMS Designer.

Syntax

```
amsDirect.prep cdsGlobalsLib string "lib_name"
```

Value

`lib_name`	The library to hold the `cds_globals` module. The default is an empty string.

Description

AMS Designer automatically generates the `cds_globals` module that contains the global signals. The cell name for the module is fixed, but you can use this variable to specify the library name.

Example

```
amsDirect.prep cdsGlobalsLib string "myglobelib"
```

Tells AMS Designer to store the `cds_globals` module in the `myglobelib` library.

cdsGlobalsView

Specifies the view for the `cds_globals` module created by AMS Designer.

Syntax

`amsDirect.prep cdsGlobalsView string "`*`view_name`*`"`

Value

`view_name`	The view to be used for the `cds_globals` module. The default is an empty string.

Description

AMS Designer automatically generates the `cds_globals` module that contains the global signals. The cell name for the module is always `cds_globals`, but you can use this variable to specify the view name to be used.

Example

```
amsDirect.prep cdsGlobalsLib string "myglobelib"
amsDirect.prep cdsGlobalsView string "globeview”
```

Tell AMS Designer to store the `cds_globals` module in the `myglobelib` library, in the `cds_globals` cell, and to use a view name of `globeview`. (For information about `cdsGlobalsLib`, see “cdsGlobalsLib” on page 388.)

checkAndNetlist

Checks cellview data for Verilog-AMS compatibility and generates a Verilog-AMS netlist if no errors are found.

Syntax

`amsDirect.vlog checkAndNetlist boolean t | nil`

Values

`t`	Generates a netlist for the cellview if it does not find any errors while checking cellview data.
`nil`	Does not check cellview data and does not generate a netlist. This is the default.

Description

You can use this variable to create error-dependent netlists in Verilog-AMS. The `checkAndNetlist` variable takes precedence over the `checkOnly` variable.

Example

```
amsDirect.vlog checkOnly boolean nil
amsDirect.vlog checkAndNetlist boolean t
```

Tell the AMS netlister to generate a Verilog-AMS netlist for the cellview if there are no errors. In this example, the `nil` value for the `checkOnly` variable is ignored because the `t` value for the `checkAndNetlist` variable takes precedence.

checkOnly

Checks cellview data for Verilog-AMS compatibility without generating a Verilog-AMS netlist.

Syntax

`amsDirect.vlog checkOnly boolean t | nil`

Values

`t`	Checks cellview data, but does not generate a netlist for the cellview.
`nil`	Does not check cellview data or generate a netlist. This is the default.

Description

You can use this variable to check cellviews in Verilog-AMS. The `checkAndNetlist` variable takes precedence over the `checkOnly` variable.

Example

```
amsDirect.vlog checkOnly boolean t
```

Tells the AMS netlister to check a cellview for Verilog-AMS compliance but not to create a netlist for the cellview.

checktasks

Checks for the presence of non-predefined system tasks or functions in the source code.

Syntax

```
amsDirect.vlog checktasks boolean  t | nil
```

Values

`t`	Checks for the presence of non-predefined system tasks or functions in the source code.
`nil`	Does not check for the presence of non-predefined system tasks or functions in the source code.This is the default.

Example

```
amsDirect.vlog checktasks boolean t
```

Tells AMS Designer to compile Verilog files with the `-checktasks` option. As a result, the generated command might look like this.

```
ncvlog -checktasks
```

compileAsAMS

Specifies whether a Verilog file is handled as a Verilog-AMS file during compilation.

Syntax

`amsDirect.vlog compileAsAMS boolean t | nil`

Values

`t`	All Verilog files are compiled with the `-ams` option. This is the default.
`nil`	Verilog files with the extensions `.vams` or `.va` are compiled with the `-ams` option. Verilog files with the extension `.v` are compiled without the `-ams` option. If a Verilog file is actually a link, the decision to use or omit the `-ams` option is based on the extension of the name of the physical file that is the target of the link.

Description

You can use this variable to specify that Verilog-D files are not to be compiled with the `-ams` option. You might need to avoid using the `-ams` option, for example, if the Verilog-D files that you are compiling contain identifiers that are keywords in Verilog-AMS.

AMS Designer assumes that any file (or target of a file that is a link) with a `.v` extension contains Verilog-D code.

Example

```
amsDirect.vlog compileAsAMS boolean t
```

Tells AMS Designer to compile Verilog-D files with the `-ams` option. As a result, the generated command might look like this.

```
ncvlog -ams
```

compileExcludeLibs

Specifies libraries to be excluded when AMS Designer runs in compile all mode.

Syntax

`amsDirect.prep compileExcludeLibs string "`*`list_of_libraries`*`"`

Value

`list_of_libraries`	A list of library names separated by white space. Libraries with these names are not considered when AMS Designer runs in compile all mode. The default is an empty string.

Description

In compile all mode (such as when the `compileMode` variable is set to `"all"`), the default behavior of AMS Designer is to compile every cell referenced in the design hierarchy. However, when you use the `compileExcludeLibs` variable, cells in the design hierarchy that belong to a library listed in *`list_of_libraries`* are not compiled.

Read-only cellviews are never compiled. Nevertheless, if your design uses many cells from read-only libraries, the compile all step might run faster if you include those read-only libraries in *`list_of_libraries`*.

Example

```
amsDirect.prep compileExcludeLibs string "compiledLib readOnlyLib"
```

Tells AMS Designer not to attempt to compile any cells that belong to either the `compiledLib` or the `readOnlyLib` library.

compileMode

Specifies the conditions under which AMS Designer (working through the AMS netlister) compiles modules. When a module to be compiled is a VHDL or VHDL-AMS module, both the entity and the architecture are compiled.

Syntax

`amsDirect.prep compileMode cyclic "none" | "incremental" | "all"`

Values

`none`	Specifies that nothing is to be compiled.
`incremental`	Specifies that only newly netlisted modules are to be compiled. This is the default.
`all`	Specifies that, for each cellview in the design configuration, the `ncvlog` or `ncvhdl` compiler is to compile the netlist specified by the `master.tag` file for the view. If the master netlist is a Verilog, Verilog-A, or Verilog-AMS file, the `ncvhdl` compiler also compiles the first netlist found in files named `vhdl.vhms` or `vhdl.vhd` (in that order). If the master netlist is a VHDL or VHDL-AMS file, the `ncvlog` compiler also compiles the first netlist found in files named `verilog.vams`, `verilog.va`, `verilog.v`, or `veriloga.va` (in that order), These compilations occur whether or not the cell is netlisted in this run. Note that AMS Designer issues an error if the netlist specified by the `master.tag` file is a VHDL-AMS file, but the installed simulator does not support VHDL-AMS. AMS Designer compiles VHDL (digital) and VHDL-AMS design units in an order that resolves compilation order dependencies. If there is no `master.tag` file for the cellview, the `ncvlog` compiler compiles the first netlist found in files named `verilog.vams`, `verilog.va`, `verilog.v`, or `veriloga.va` (in that order). Similarly, the `ncvhdl` compiler also compiles the first netlist found in files named `vhdl.vhms` or `vhdl.vhd` (in that order).

In summary, specifying `all` causes a maximum of two files to be compiled for each cellview: one Verilog, Verilog-A, or Verilog-AMS cellview to be compiled by `ncvlog`; one VHDL or VHDL-AMS cellview to be compiled by `ncvhdl`.

Example

```
amsDirect.prep compileMode cyclic "incremental"
```

Tells AMS Designer to compile only newly created netlists.

confirmADEStateImport

Determines whether a dialog box opens to caution users that importing an ADE state overwrites existing netlister and compiler settings.

Syntax

```
amsDirect confirmADEStateImport boolean t | nil
```

Values

`t`	A dialog box appears when an ADE state is imported. The text of the dialog is `This action will cause your current netlister and compiler settings to be overwritten. A backup copy will be saved, but your current settings may change. Continue with import?`
`nil`	The dialog box does not appear.

Example

```
amsDirect       confirmADEStateImport   boolean nil
```

This example turns off the confirmatory dialog box so that clicking *OK* in the Import from ADE State window immediately imports the selected state.

connectRulesCell

Specifies the cell that contains the `connectrules` module.

Syntax

`amsDirect.prep connectRulesCell string "`*`cell`*`"`

Value

`cell`	The cell that contains the `connectrules` module. The default is `mixedsignal`.

Description

Depending on the version of the simulator that you are using, either the `connectRulesCell` or the `connectRulesCell2` variable is effective. The effective member of the pair, in conjunction with the `connectRulesLib` and `connectRulesView` variables, specifies the `connectrules` module.

Example

```
amsDirect.prep connectRulesLib string "mylib"
amsDirect.prep connectRulesCell string "comparator"
amsDirect.prep connectRulesView string "connectrules"
```

When the `connectRulesCell` variable is effective, these examples tell the elaborator and simulator to use the following `connectrules` module.

```
mylib.comparator:connectrules
```

connectRulesCell2

Specifies the cell that contains the `connectrules` module.

Syntax

`amsDirect.prep connectRulesCell2 string "`*`cell`*`"`

Value

`cell`	The cell that contains the `connectrules` module. The default is `ConnRules_5V_full`.

Description

Depending on the version of the simulator that you are using, either the `connectRulesCell` or the `connectRulesCell2` variable is effective. The effective member of the pair, in conjunction with the `connectRulesLib` and `connectRulesView` variables, specifies the `connectrules` module.

Example

```
amsDirect.prep connectRulesLib string "mylib"
amsDirect.prep connectRulesCell2 string "compar2"
amsDirect.prep connectRulesView string "connectrules"
```

When the `connectRulesCell2` variable is effective, these variables tell the elaborator and simulator to use the following `connectrules` module.

```
mylib.compar2:connectrules
```

connectRulesLib

Specifies the library that contains the `connectrules` module.

Syntax

`amsDirect.prep connectRulesLib string "`*`lib`*`"`

Value

`lib`	The library that contains the `connectrules` module. The default is an empty string.

Description

This variable, in conjunction with the `connectRulesCell` and `connectRulesView` variables, specifies the `connectrules` module.

Example

```
amsDirect.prep connectRulesLib string "mylib"
amsDirect.prep connectRulesCell string "comparator"
amsDirect.prep connectRulesView string "connectrules"
```

Tell the elaborator and simulator to use the following `connectrules` module.

```
mylib.comparator:connectrules
```

connectRulesView

Specifies the cellview that contains the `connectrules` module.

Syntax

`amsDirect.prep connectRulesView string "`*`view`*`"`

Value

`view`	The cellview that contains the `connectrules` module. The default is an empty string.

Description

This variable, in conjunction with the `connectRulesCell` and `connectRulesLib` variables, specifies the `connectrules` module.

Example

```
amsDirect.prep connectRulesLib string "mylib"
amsDirect.prep connectRulesCell string "comparator"
amsDirect.prep connectRulesView string "connectrules"
```

Tell the elaborator and simulator to use the following `connectrules` module.

```
mylib.comparator:connectrules
```

defaultRunDir

Specifies a directory to be used as the current run directory when the AMS menu is installed or when the `amsdesigner` command is run.

Syntax

`amsDirect defaultRunDir string "`*`rundir`*`"`

Value

`rundir`	The directory to be used as the run directory. The default is an empty string.

Description

An empty string for this variable means that the run directory must be specified in some other way, either by using the graphical user interface or by associating a run directory with a configuration. If, in the AMS Run Directory form, you turn on *Always use this run directory for this configuration*, that specification takes precedence over the value set by the `defaultRunDir` ams.env variable.

Example

```
amsDirect defaultRunDir string "newrundir"
```

Tells AMS Designer to use the `newrundir` directory as the run directory, unless this designation is overridden in some other way.

detailedDisciplineRes

Specifies the kind of discipline resolution to be used.

Syntax

```
amsDirect.prep detailedDisciplineRes boolean t | nil
```

Values

`t`	AMS Designer uses the detailed method of discipline resolution.
`nil`	AMS Designer uses the default method of discipline resolution. This is the default.

Description

For a description of these methods, see the “Discipline Resolution Method” section of Chapter 11, in the *Cadence Verilog-AMS Language Reference*.

Example

```
amsDirect.prep detailedDisciplineRes boolean nil
```

Specifies that the default method of discipline resolution is to be used.

discipline

Specifies a default discipline for discrete nets for which a discipline is either not specified or cannot be determined through discipline resolution.

Syntax

`amsDirect.prep discipline string "`*`discipline`*`"`

Value

`discipline`	The discipline to be used for discrete nets of otherwise unknown discipline. The default is `logic`.

Example

```
amsDirect.prep discipline string "logic"
```

Specifies that the `logic` discipline is to be used for discrete nets that do not have a known discipline.

errOutInconsistentMasters

Controls whether the netlister terminates with an error when it encounters an unbound master cellview.

Syntax

`amsDirect.vlog errOutInconsistentMasters boolean t | nil`

Values

`t`	The netlister terminates with an error if it encounters any unbound master cellviews.
`nil`	The netlister does not terminate with an error when it encounters unbound master cellviews. This is the default value.

Example

```
amsDirect.vlog errOutInconsistentMasters boolean t
```

The netlister terminates with an error if it encounters any unbound master cellviews.

excludeViewNames

Specifies the names of cellviews that are not to be netlisted.

Syntax

`amsDirect.vlog excludeViewNames string "`*`list_of_view_names`*`"`

Value

`list_of_view_names`	A list of view names separated by white space. Cellviews with these names are not netlisted. The default is an empty string.

Description

Normally, changes to cellviews while netlisting is enabled or changes to the CDF of cells while the `netlistAfterCdfChange` variable is set to `t` trigger netlisting. However, cells whose names are included in *`list_of_view_names`* are not netlisted.

Example

```
amsDirect.vlog excludeViewNames string "sch[0-3]"
```

hdlVarFile

Specifies the name of the `hdl.var` file to be used with the `ncvlog`, `ncelab`, and `ncsim` commands.

Syntax

`amsDirect.prep hdlVarFile string "`*`file`*`"`

Value

`file`	An `hdl.var` file to be used with the `-hdlvar` option of the `ncvlog`, `ncelab`, and `ncsim` commands. If *`file`* is not specified, the `-hdlvar` option is not used with these commands. The default is an empty string.

Description

If *`file`* is an empty string, the `ncvlog`, `ncelab`, and `ncsim` commands run without the `-hdlvar` option. As a result, each application looks for an `hdl.var` file in the directory where that application started. If there is no `hdl.var` file in that location, the program issues a warning. Because `ncvlog` starts in the directory where you start the Cadence software and `ncelab` and `ncsim` start in the run directory, the programs are likely to use different `hdl.var` files if you do not specify them explicitly.

If you use a relative path, be aware that paths are relative to the directory where the program starts. The `ncvlog` program starts in the current working directory so the path is relative to that directory. However, the `ncelab` and `ncsim` programs start in the run directory so the path for them is relative to the run directory. As a consequence, the different programs are likely to use different `hdl.var` files.

To be sure that all the programs find the appropriate `hdl.var` file, use an absolute path.

Example

```
amsDirect.prep hdlVarFile string "prepvarfile"
```

Specifies that the `ncvlog`, `ncelab`, and `ncsim` commands generated by AMS Designer are to include the following option.

```
-hdlvar "prepvarfile"
```

headerText

Specifies the kind of header to be used at the beginning of netlists generated by AMS Designer.

Syntax

`amsDirect.vlog headerText cyclic "none" | "file" | "script"`

Values

`none`	Specifies that the default header is to be used.
`file`	Specifies that the header of the netlist is to consist of the default header followed by the text of a file. The name of the file containing the text is specified by the `templateFile` variable. For more information, see “templateFile” on page 561.
`script`	Specifies that the header of the netlist is to consist of the default header followed by the text generated by running a script. The name of the file containing the script is specified by the `templateScript` variable. For more information, see “templateScript” on page 562.

Example

```
amsDirect.vlog headerText cyclic "none"
```

Tells AMS Designer to insert the default header at the beginning of each generated netlist. As a result, each netlist begins with lines like the following.

```
// Verilog-AMS netlist generated by the AMS netlister, version 4.4.6.100.43.
// Cadence Design Systems, Inc.
```

ieee1364

Checks the source code for compatibility with the IEEE standard described in *IEEE-1364 Verilog Hardware Description Language Reference Manual*.

Syntax

`amsDirect.vlog ieee1364 boolean  t | nil`

Values

`t`	Checks the source code for compatibility with the IEEE standard described in *IEEE-1364 Verilog Hardware Description Language Reference Manual*.
`nil`	Does not check the source code for compatibility with the IEEE standard. This is the default.

Example

```
amsDirect.vlog ieee1364 boolean t
```

Tells AMS Designer to compile Verilog files with the `-ieee1364` option. As a result, the generated command might look like this.

```
ncvlog -ieee1364
```

ifdefLanguageExtensions

Controls the netlisting of attributes.

Syntax

`amsDirect.vlog ifdefLanguageExtensions boolean t | nil`

Values

`t`	Generates `` `ifdef INCA`` clauses in the netlist for attribute statements.
`nil`	Does not generate `` `ifdef INCA`` clauses. This is the default.

Description

If you plan to use a compiler that does not support the Cadence attribute statements, you can use this variable to enclose the statements in an `` `ifdef INCA`` clause. Note that this clause produces a Verilog-AMS netlist that is more difficult to read.

Example

You need to copy your netlists to a different location where they will be used in a purely text based flow without using configurations and the Virtuoso® Hierarchy Editor. In this situation, the library bindings in the netlist need to be disabled.

With the `ifdefLanguageExtensions` variable set to `nil`, the netlist looks like this.

```
vsource #(.dc(3), .type("dc"))  (*
integer library_binding = "analogLib";  *) V0 ( cds_globals.\vdd! ,
cds_globals.„nd!  );
vsource #(.dc(-3), .type("dc"))  (*
integer library_binding = "analogLib";  *) V1 ( cds_globals.\vss! ,
cds_globals.„nd!  );
```

Setting the `ifdefLanguageExtensions` variable to `t` results in a netlist where the library bindings are enclosed in `` `ifdef INCA`` clauses, so that they can be turned off.

```
vsource #(.dc(3), .type("dc"))
‘ifdef INCA (* integer library_binding = "analogLib";  *) ‘endif
V0 ( cds_globals.\vdd! , cds_globals.„nd!  );
vsource #(.dc(-3), .type("dc"))
‘ifdef INCA (* integer library_binding = "analogLib";  *) ‘endif
V1 ( cds_globals.\vss! , cds_globals.„nd!  );
```

ignoreIllegalCDFParams

Specifies whether to ignore non-compliant CDF parameters when netlisting.

For example, CDF parameters such as `min`, `max`, and `abs` for the `vcvs` cell in the analogLib library are non-compliant because they are reserved keywords in the Verilog-AMS language. By default, the netlister issues a warning or error message (depending on the value of the `allowIllegalIdentifiers` variable). You can turn off this notification by setting the `ignoreIllegalCDFParams` variable to `t`.

Syntax

```
amsDirect.vlog ignoreIllegalCDFParams boolean t | nil
```

Values

`t`	Netlisting does not notify you about any non-compliant CDF parameter names.
`nil`	Netlisting notifies you (by warning or error message) about any non-compliant CDF parameter names. This is the default. **Note:** The type of notification depends on the value of the `allowIllegalIdentifiers` variable.

implicitTmpDir

Specifies the implicit temporary (TMP) directory. The AMS Designer environment ensures that the specified directory is the same as the run directory.

The `implicitTmpDir` variable has no effect when the `netlistToRunDir` variable is set to `nil` or when the `useRunDirNetlistsOnly` variable is set to `nil`.

Note: The AMS Designer environment sets this variable automatically, and you should not change the setting by hand. To control netlisting into temporary directories, you need to set only the `netlistToRunDir` variable, and, optionally, the `useRunDirNetlistsOnly` variable.

Syntax

`amsDirect implicitTmpDir string "`*`implicitTmpDir`*`"`

Value

`implicitTmpDir`	Automatically set, by the AMS Designer environment, to the run directory.

incdir

Specifies directories to be searched for files specified by the `` `include `` compiler directive.

Syntax

`amsDirect.vlog incdir string "`*`dirs_to_search`*`"`

Value

`dirs_to_search`	Directories to be searched for specified files. The format must be as illustrated in the following example. The default is an empty string.

Example

```
amsDirect.vlog incdir string "11-LevelOneDir11-LevelTwoDir"
```

Generates a command that includes two `-incdir` options.

```
ncvlog
    -incdir LevelOneDir
    -incdir LevelTwoDir
```

includeFiles

Specifies a list of files to be included with the `` `include `` directive at the top of each Verilog-AMS netlist that is created by the AMS netlister.

Syntax

```
amsDirect.vlog includeFiles string "(file_to_include_1)
         { (file_to_include_N) }"
```

Value

file_to_include_N	Files to be included in the netlist by the `` `include `` compiler directive. If you list more than a single file, separate the files with spaces. The default is `disciplines.vams`.

Example

```
amsDirect.vlog includeFiles string "(disciplines.vams) (func1.h) (func2.h)"
```

Tells AMS netlister to include the files `func1.h` and `func2.h` at the top of the netlist. As a result, the netlist contains the lines:

```
`include "disciplines.vams"
`include "func1.h"
`include "func2.h"
```

includeInstCdfParams

Specifies how the AMS netlister handles CDF parameters.

Syntax

```
amsDirect.vlog includeInstCdfParams boolean t | nil
```

Values

`t`	For each instance, writes to the netlist all parameters found in the CDF for the instance master.
`nil`	For each instance, writes to the netlist only CDF parameters actually set on the instance and all CDF parameters containing pPar or atPar expressions. This is the default.

Description

This variable is ignored if the instance master has an `ams` section in the simulation information (simInfo) section of the CDF. In this case, the AMS netlister does only what the simInfo says to do. For more information, see “The ams Fields” on page 647.

initFile

Specifies a SKILL file to be loaded at startup. The function definitions and code in the file are used to override netlist procedures.

Syntax

`amsDirect initFile string "`*`path`*`"`

Value

`path`	The path and filename of a SKILL file. The path can contain shell environment variables (consisting of a `$` followed by alphanumeric characters). A path that begins with a `/` (slash) is considered an absolute path. A path that does not begin with a `/` is considered to be relative to the current working directory (CWD).

The file can contain SKILL function definitions and code to override netlist procedures. The user code can call

- Core SKILL language functions
- DB functions (which begin with `db`)
- DDPI functions (which begin with `dd`)
- CDF functions (which begin with `cdf`)
- AMS functions (which begin with `ams`)

The SKILL code must not assume that other contexts are loaded by default, though the code can load other contexts as necessary.

Example

```
amsDirect initFile string "$YOUR_INSTALL_DIR/tools/dfII/local/amsProcs.il"
```

Tells AMS netlister to load the `amsProcs.il` file.

instClashFormat

Specifies the format to be used to map the names of instances that collide with names of other netlist constructs.

Syntax

`amsDirect.vlog instClashFormat string "`*`format`*`"`

Value

`format`	All characters, except those listed below, are printed exactly as included in *`format`*. The following characters have the indicated special meanings.

`%b`	Original name of the instance
`%%`	Prints the `%` character

The default value of *`format`* is `%b_instclash`, which produces a mapped name like `origname_instclash` for an instance originally named `origname`.

If the resulting name is illegal in Verilog-AMS, the name is mapped. If the mapped name clashes with the name of another object, the name undergoes collision mapping.

Example

```
amsDirect.vlog instClashFormat string "%b_iclash"
```

Tells AMS netlister to map clashing instance names with a suffixed `_iclash`. For example, you have an instance `samp` with a name that clashes with a net named `samp`. The AMS netlister maps the instance to the system-generated name `samp_iclash`.

iterInstExpFormat

Specifies the format to be used for the names of constituent elements generated by the expansion of an iterated instance.

Syntax

```
amsDirect.vlog iterInstExpFormat string "format"
```

Value

format	All characters, except those listed below, are printed exactly as included in *format*. The following characters have the indicated special meanings.

`%b`	Base name of the instance
`%l` (small L)	Left bound of the range
`%r`	Right bound of the range
`%i`	Index of the current iteration
`%%`	Prints the `%` character

The default value of *format* is `%b_%i`, which produces names like `instbn_1`, `instbn_2`, and so on, where `instbn` is the base name of the instance.

If a resulting name is illegal in Verilog-AMS, the name is mapped. If the mapped name clashes with the name of another object, the name undergoes collision mapping.

Example

```
amsDirect.vlog iterInstExpFormat string "%b_%l_%r_%i"
```

Tells AMS netlister to generate names that include the left and right bounds. For example, you have an iterated instance with the name `scatstr`. The names of the expanded instances are:

```
scatstr_1_3_1
scatstr_1_3_2
scatstr_1_3_3
```

language

Specifies the language to be used for netlists.

Syntax

```
amsDirect.prep language string "verilog"
```

Value

`verilog`	Specifies that the language to be used for netlists is Verilog-AMS. This is the default.

Description

In this release, Verilog-AMS is the only supported language.

lexpragma

Enables processing of lexical pragmas.

Syntax

```
amsDirect.vlog lexpragma boolean  t | nil
```

Values

`t`	Turns on processing of lexical pragmas.
`nil`	Turns of processing of lexical pragmas. This is the default

Description

Lexical pragmas are pragmas that can be associated with any Verilog or VHDL construct to indicate that translation/synthesis is turned off. The following pragmas are classified as lexical pragmas:

- cadence translate_off and cadence translate_on (also: synopsys translate_off and synopsys translate_on)
- cadence synthesis_off and cadence synthesis_on (also: synopsys synthesis_off and synopsys synthesis_on)
- rtl_synthesis off and rtl_synthesis on

If you compile with the `-lexpragma` option, any HDL constructs between a translate_off/ synthesis_off pragma and a translate_on/synthesis_on pragma are treated as comments. For example, if the source code contains the following pragmas, `’define CI2CLKP 10` is treated as a comment.

```
’define CI2CLKP 512
// cadence translate_off
’define CI2CLKP 10
// cadence translate_on
```

If you use both -pragma and -lexpragma, lexical pragmas are processed with -lexpragma.

Example

```
amsDirect.vlog lexpragma boolean t
```

Tells AMS Designer to compile Verilog files with the `-lexpragma` option. As a result, the generated command might look like this.

```
ncvlog -lexpragma
```

logFileAction

Controls the generation of log files.

Syntax

```
amsDirect.vlog logFileAction cyclic "Overwrite log file" | "Append log file" |
     "No log file"
```

Values

`Overwrite log file`	Overwrites the log file each time ncvlog runs. This is the default.
`Append log file`	Appends all ncvlog log information to a single file.
`No log file`	Specifies that no log file be created.

Examples

- The `ams.env` variable

  ```
  amsDirect.vlog logFileAction cyclic "Overwrite log file"
  ```

  generates an `ncvlog` command similar to the following.

  ```
  ncvlog
  -logfile /usr1/cds11752/alpha6/test8/SAR_A2D/tutorial_run/ncvlog.log
  ```

- The `ams.env` variable

  ```
  amsDirect.vlog logFileAction cyclic "Append log file"
  ```

  generates an `ncvlog` command similar to the following.

  ```
  ncvlog
      -logfile /usr1/cds11752/alpha6/test8/SAR_A2D/tutorial_run/ncvlog.log
      -append_log
  ```

- The `ams.env` variable

  ```
  amsDirect.vlog logFileAction cyclic "No log file"
  ```

  generates an `ncvlog` command similar to the following.

  ```
  ncvlog
      -nolog
  ```

logFileName

Sets the name of the log file.

Syntax

`amsDirect logFileName string "`*`logFileName`***`"`**

Value

`logFileName`	Specifies the name of the log file. The default is `ams_direct.log`.

Description

When the AMS netlister processes a design, it creates a log file that contains errors, warnings, and informational messages about the design. You can use this variable to name the log file.

The *`logFileName`* that you specify with this variable interacts with the `CDS_LOG_PATH` environment variable to determine the actual log file name that is used.

- If *`logFileName`* is an absolute path, the log file is written to *`logFileName`*.
- If *`logFileName`* is a relative path and
  - `CDS_LOG_PATH` is null, *`logFileName`* is placed in the current directory.
  - `CDS_LOG_PATH` is non-null, the value of `CDS_LOG_PATH` is prepended to the *`logFileName`*.
- Setting both *`logFileName`* and the `CDS_LOG_PATH` to absolute paths causes a fatal error.

Note that the `-LOg` option of the `amsdirect` command takes precedence over the `logFileName` variable. For more information, see "Netlisting from the UNIX Command Line" on page 189.

Examples

- The `logFileName` variable is not used and `CDS_LOG_PATH` environment variable is unset. The default `logFileName` is used and log data goes to `ams_direct.log` in the current directory.
- The `logFileName` variable is not used and `CDS_LOG_PATH` environment variable is set to the absolute path

  ```
  /usr1/dave/test7
  ```

  Log data goes to

  ```
  /usr1/dave/test7/ams_direct.log
  ```

- The `logFileName` variable is set to the absolute path

  ```
  /usr1/dave/test8/test8_log2
  ```

  The `CDS_LOG_PATH` environment variable is set to the absolute path

  ```
  /usr1/dave/test8
  ```

  The `logFileName` variable takes precedence and the log data is written to

  ```
  /usr1/dave/test8/test8_log2
  ```

- The `logFileName` variable is set to the relative path

  ```
  ./usr1/dave/test8/test8_log2
  ```

  The `CDS_LOG_PATH` environment variable is set to

  ```
  /usr1/dave/test8
  ```

  In this case, the `CDS_LOG_PATH` is prepended to the `logFileName` and log data goes to

  ```
  /usr1/dave/test8/usr1/dave/test8/test8_log2
  ```

macro

Defines macros for the `ncvlog` command.

Syntax

`amsDirect.vlog macro string "`*`macros`*`"`

Value

`macros`	Macros to be defined. The format must be as illustrated in the following example. The default is an empty string.

Example

```
amsDirect.vlog macro string "4-gate2-or4-slow8-'16’h03'"
```

Generates an `ncvlog` command similar to the following.

```
ncvlog
    -define gate=or
    -define slow=16’h03
```

markcelldefines

Inserts `` `celldefine `` and `` `endcelldefine `` compiler directives to tag module instances as cell instances.

Syntax

`amsDirect.vlog markcelldefines boolean  t | nil`

Values

`t`	Inserts `` `celldefine `` and `` `endcelldefine `` compiler directives to tag module instances as cell instances.
`nil`	Does not insert `` `celldefine `` and `` `endcelldefine `` compiler directives to tag module instances as cell instances. This is the default.

Example

```
amsDirect.vlog  markcelldefines boolean t
```

Generates an `ncvlog` command similar to the following.

```
ncvlog
    -libcell
```

maxErrors

Stops compilation if the number of errors reaches the specified maximum limit.

Syntax

`amsDirect.vlog maxErrors int` *`maxErrors`*

Value

`maxErrors`	A positive integer. Halts compilation after this number of errors occur. The default is 50.

Example

```
amsDirect.vlog maxErrors int 50
```

Tells AMS Designer to compile Verilog files with the `-errormax` option. As a result, the generated command might look like this.

```
ncvlog -errormax 50
```

messages

Prints informational messages as the compiler runs.

Syntax

```
amsDirect.vlog messages boolean  t | nil
```

Values

`t`	Prints informational messages as the compiler runs.
`nil`	Does not print informational messages as the compiler runs. This is the default.

Example

```
amsDirect.vlog messages boolean t
```

Tells AMS Designer to compile Verilog files with the `-messages` option. As a result, the generated command might look like this.

```
ncvlog -messages
```

modifyParamScope

Specifies that the AMS netlister treat atPar and dotPar expressions as pPar and iPar expressions, respectively.

Syntax

```
amsDirect modifyParamScope cyclic "no" | "warn" | "yes"
```

Values

`no`	Prints an error message and halts netlisting when the AMS netlister finds atPar or dotPar expressions. This is the default.
`warn`	Generates a warning when the AMS netlister finds an atPar or dotPar expression and treats the atPar or dotPar expression as a pPar or iPar expression, respectively.
`yes`	Treats atPar and dotPar expressions as pPar and iPar expressions, respectively. No warning messages are generated.

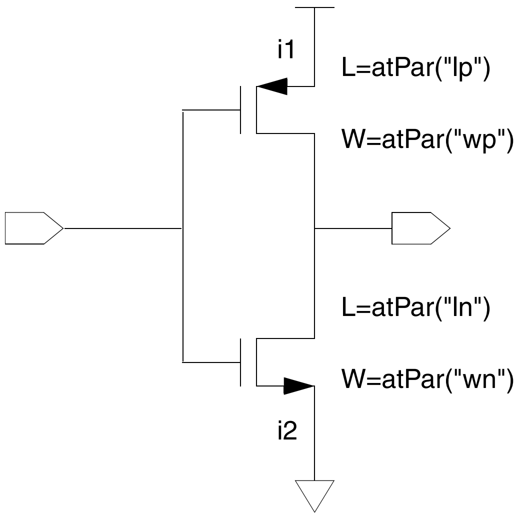
Description

The AMS netlister netlists one cellview at a time; it cannot see hierarchical dependencies defined or resolved outside of the current cellview. In addition, Verilog-AMS requires that passed parameters be resolved through the level of hierarchy immediately preceding the cellview to which the parameter applies. In other words, parameter passing cannot skip levels of the hierarchy. Because atPar and dotPar expressions allow parameters to be resolved in non-contiguous levels of the hierarchy, the AMS netlister does not support these expressions.

If you specify `warn` or `yes`, the AMS netlister treats atPar and dotPar expressions as pPar and iPar expressions, respectively, and generates a netlist. However, to avoid incorrect simulation results, you must ensure that the block instantiating the cell sets the instance parameters appropriately.

Example

Consider the following example of an inverter that employs atPar expressions. Assume that the nmos has defaults of `ln=3u` and `wn=20u` and that the pmos has defaults of `lp=3u` and `wp=40u`.



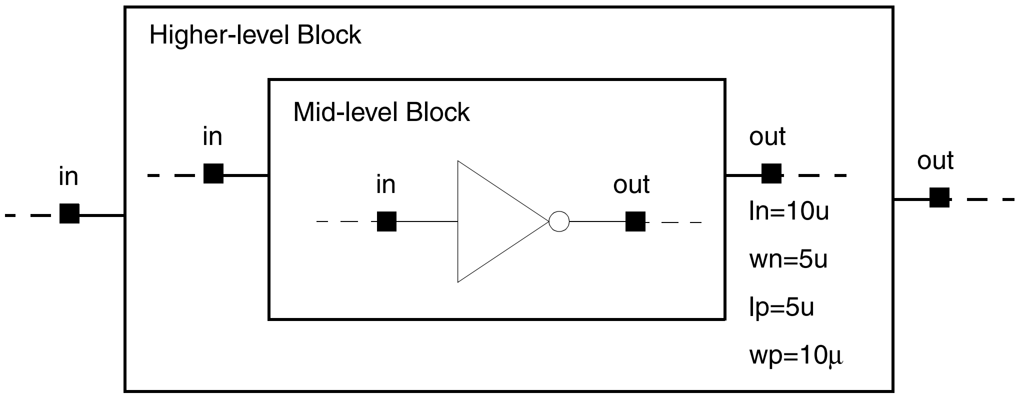
When this inverter is netlisted by the AMS netlister, it has an instance of an nmos and an instance of a pmos, each with parameters to be passed in:

```
pmos #(.W(wp), .L(lp)) i1 ( port_connections );
nmos #(.W(wn), .L(ln)) i2 ( port_connections );
```

The inverter module netlisted by the AMS netlister also has parameter statements for the parameters that are to supply values to the nmos and pmos instances:

```
parameter ln = 3u;
parameter wn = 20u;
parameter lp = 3u;
parameter wp = 40u;
```

Now assume that this inverter is instantiated in a mid-level block, as follows:



The definition of the atPar expression allows the values for the parameters `ln`, `wn`, `lp`, and `wp` to be provided at any level of the hierarchy above the mid-level block. In the preceding diagram, the values set in the higher-level block override the defaults defined in the nmos and pmos, and are used during the simulation:

- `ln` is set to 10µ for the simulation
- `wn` is set to 5µ for the simulation
- `lp` is set to 5µ for the simulation
- `wp` is set to 10µ for the simulation

This behavior is not possible when using Verilog-AMS. Verilog-AMS allows parameters to be passed from one level of hierarchy to the next level below, but the passing must be between contiguous levels. This behavior is identical to what is accomplished by pPar expressions. To be able to generate a netlist for the example, the AMS netlister must treat the atPar expressions as it does pPar expressions, expecting that any overriding of the parameters is done at the level of hierarchy immediately above.

Now assume that the AMS netlister is instructed to treat atPar expressions as it does pPar expressions. In this case, the higher-level block has an instance of the mid-level block, with the parameters set:

```
midlevel #(.ln(10u), .wn(5u), .lp(5u), .wp(10u)) i1 ( port_connections );
```

This instantiation assumes that the mid-level module has parameter declarations for the four parameters being passed in. However, the mid-level block does not reference these parameters at all, so no parameter declarations are printed by the AMS netlister.

The mid-level block has an instance of the inverter, passing no parameters at all:

```
inverter i1 ( port_connections );
```

Thus, when the nmos and pmos parameters are resolved, they are set to the defaults, because no values are passed in to override them:

- `ln` is set to 3µ for the simulation
- `wn` is set to 20µ for the simulation
- `lp` is set to 3µ for the simulation
- `wp` is set to 40µ for the simulation

Notice how these simulation values differ from those listed earlier. This example illustrates how simply instructing the AMS netlister to treat atPar expressions as pPar expressions might not produce the results you expect. To avoid incorrect results, ensure that parameters are passed in accordance with Verilog-AMS restrictions.

ncelabAccess

Sets the visibility access for all objects in the design.

Syntax

```
amsDirect.prep ncelabAccess cyclic  "Off" | "Read" | "Read/Write" |
     "Connectivity" | "All"
```

Values

`Off`	Equivalent to the option `-access -r-w-c`. This is the default.
`Read`	Appends the option `-access +r-w-c`.
`Read/Write`	Appends the option `-access +r+w-c`.
`Connectivity`	Appends the option `-access +r-w+c`.
`All`	Appends the option `-access +r+w+c`.

Example

```
amsDirect.prep  ncelabAccess    cyclic  "Read/Write"
```

Generates an `ncelab` command that looks like this.

```
ncelab amslib.top:config -access +r+w-c
```

ncelabAfile

Specifies an access file. An access file is a text file that lets you set the visibility access for particular instances or portions of a design.

Syntax

`amsDirect.prep ncelabAfile string "`*`path_and_file`*`"`

Value

`path_and_file`	The default is an empty string.

Example

```
amsDirect.prep ncelabAfile string "/usr1/alpha6/test8/SAR_A2D/afile.acs"
```

Generates an `ncelab` command like the following.

```
ncelab amslib.top:config -afile /usr1/alpha6/test8/SAR_A2D/afile.acs
```

ncelabAnnoSimtime

Enables the use of PLI/VPI routines that modify delays at simulation time.

Syntax

`amsDirect.prep ncelabAnnoSimtime boolean  t | nil`

Values

`t`	
`nil`	This is the default.

Description

The PLI/VPI routines that modify routines are `acc_replace_delays`, `acc_append_delays`, and `vpi_put_delays`.

If you do not specify this option at elaboration time, but then run a PLI/VPI routine that tries to modify delays at simulation time, AMS Designer issues a message and does not modify delays.

This option disables optimizations in the simulator that take delays into account and has some performance impact. Use this option only if you intend to modify delays at simulation time.

Using this option sets the default access to simulation objects to read/write when the design is elaborated. Do not use this option if you want to run in regression mode.

Example

```
amsDirect.prep ncelabAnnoSimtime boolean t
```

Tells the AMS netlister to prepare to simulate with routines that modify delays at simulation time. As a result, the generated `ncelab` command looks like the following.

```
ncelab amslib.top:config -anno_simtime
```

ncelabArguments

Specifies additional arguments to be passed to the `ncelab` elaborator.

Syntax

`amsDirect.prep ncelabArguments string "`*`arguments`*`"`

Value

`arguments`	One or more arguments to be passed to the `ncelab` elaborator. The default is an empty string.

Description

If *`arguments`* is an empty string, the `ncelab` command includes just the arguments listed on the *Elaborator* pane of the AMS Options window. You can use the `ncelabArguments` variable with a non-empty string to pass additional arguments to the elaborator.

Example

```
amsDirect.prep ncelabArguments string "-libverbose"
```

Adds the `-libverbose` argument to the other arguments normally used on the `ncelab` command.

ncelabCoverage

Enables code coverage instrumentation for the digital part of the design.

Syntax

`amsDirect.prep ncelabCoverage boolean  t | nil`

Values

`t`	Enables code coverage instrumentation.
`nil`	Turns off code coverage instrumentation. This is the default.

Example

```
amsDirect.prep ncelabCoverage boolean t
```

Generates an `ncelab` command like the following.

```
ncelab amslib.top:config -coverage
```

ncelabDelayMode

Specifies the delay mode to be used for digital Verilog-AMS portions of the hierarchy.

Syntax

```
amsDirect.prep ncelabDelayMode cyclic "None" | "Zero" | "Unit" | "Path" |
     "Distributed"
```

Values

`None`	Delays simulate as specified in the model's source description files. This is the default.
`Zero`	Similar to `Unit` delay mode in that the simulator ignores all module path delay information, timing checks, and structural and continuous assignment delays.
`Unit`	The AMS simulator ignores all module path delay information and timing checks and converts all non-zero structural and continuous assignment delay expressions to a unit delay of one simulation time unit.
`Path`	The AMS simulator derives its timing information from specify blocks. When a module contains a specify block with one or more module path delays, all structural and continuous assignment delays within that module (with the exception of `trireg` charge decay times) are set to zero.
`Distributed`	The AMS simulator ignores all module path delay information and uses all distributed delays and timing checks. Distributed delays are delays on nets, primitives, or continuous assignments–in other words, delays other than those specified in procedural assignments and specify blocks.

Example

```
amsDirect.prep ncelabDelayMode cyclic "Unit"
```

Generates an `ncelab` command like the following.

```
ncelab amslib.top:config -delay_mode Unit
```

When you elaborate with this command, the AMS simulator ignores all module path delay information and timing checks and converts all non-zero structural and continuous assignment delay expressions to a unit delay of one simulation time unit.

ncelabDelayType through ncelabMessages

These `ams.env` `ncelab*` variables correspond to `ncelab` command options as follows:

ams.env Variable	ncelab Command Option
`ncelabDelayType`	`-MAxdelays`, `-MIndelays`, `-TYpdelays`
`ncelabDisableenht`	`-DISAble_enht`
`ncelabEpulseFiltering`	`-EPULSE_ONDetect`, `-EPULSE_ONEvent`
`ncelabEpulseNeg`	`-EPULSE_NEg`
`ncelabExpand`	`-EXPand`
`ncelabExtendtcheckdatalimit`	`-EXTEND_TCHECK_Data_limit`
`ncelabExtendtcheckreferencelimit`	`-EXTEND_TCHECK_Reference_limit`
`ncelabGenafile`	`-GENAfile`
`ncelabIeee1634`	`-IEEe1364`
`ncelabInterconnmultisrc`	`-CAint`
`ncelabLibverbose`	`-LIBVerbose`
`ncelabLoadpli1`	`-LOADPli1`
`ncelabLoadvpi`	`-LOADVpi`
`ncelabLogFileAction`	`-LOGfile`, `-NOLog`, `-APpend_log`
`ncelabMaxErrors`	`-ERrormax`
`ncelabMessages`	`-MEssages`

For information about `ncelab` command options, see Chapter 8 of *NC-Verilog Simulator Help*.

ncelabMixEsc

Controls whether the `-mixesc` option is passed on the `ncelab` command. The `-mixesc` option is required when elaborating if you instantiate VHDL or VHDL-AMS in a Verilog or Verilog-AMS module and you use escaped entity, port, or generic names within the VHDL or VHDL-AMS descriptions.

Syntax

`amsDirect.prep ncelabMixEsc boolean t | nil`

Values

`t`	Places a checkmark next to the *Allow mixed-case, escaped identifiers in VHDL* field, on the *VHDL* pane of the AMS Options window. As a result, the `-mixesc` option is passed on the `ncelab` command.
`nil`	Removes the checkmark, indicating that the `-mixesc` option is not to be passed on the `ncelab` command. This is the default.

Example

```
amsDirect.prep ncelabMixEsc boolean t
```

Places the checkmark next to the *Allow mixed-case, escaped identifiers in VHDL* field. As a result, the elaborator is able to distinguish VHDL entities whose escaped names differ only by the case of letters.

ncelabModelFilePaths

Caution

Do not hand-edit this variable.

This variable contains the information that populates the model files table on the Model Library Setup form.

Syntax

`amsDirect.prep  ncelabModelFilePaths    string  "`*`model_files`*`"`

Value

`model_files`	The status, paths, names, and sections of analog model files. The default value is an empty string.

Example

```
amsDirect.prep  ncelabModelFilePaths   string  "9-isEnabled5-false4-path31-$PROJ3/
SAR_A2D/spectreprim3.scs:9-isEnabled5-false4-path31-$PROJ3/SAR_A2D/
spectreprim2.scs7-section7-typical:9-isEnabled4-true4-path58-/usr1/cds11752/
alpha6/vhdltestdir/SAR_A2D/spectre_prim.scs"
```

ncelabNeverwarn through ncelabVipdelay

These `ams.env` `ncelab*` variables correspond to `ncelab` command options as follows:

ams.env Variable	ncelab Command Option
`ncelabNeverwarn`	`-NEVerwarn`
`ncelabNoautosdf`	`-NOAutosdf`
`ncelabNocopyright`	`-NOCopyright`
`ncelabNoipd`	`-NOIpd`
`ncelabNonegtchk`	`-NONEg_tchk`
`ncelabNonotifier`	`-NONOtifier`
`ncelabNosource`	`-NOSOurce`
`ncelabNostdout`	`-NOSTdout`
`ncelabNoTchkMsg`	`-NO_TCHK_Msg`
`ncelabNoTchkXgen`	`-NO_TCHK_Xgen`
`ncelabNotimingchecks`	`-NOTImingchecks`
`ncelabNovitalaccl`	`-NOVitalaccl`
`ncelabNoVpdmsg`	`-NO_VPD_Msg`
`ncelabNoVpdXgen`	`-NO_VPD_Xgen`
`ncelabNowarn`	`-NOWarn`
`ncelabNtcWarn`	`-NTC_Warn`
`ncelabOmichecklvl`	`-OMicheckinglevel`
`ncelabPathpulse`	`-PAthpulse`
`ncelabPlinooptwarn`	`-PLINOOptwarn`
`ncelabPlinowarn`	`-PLINOWarn`
`ncelabPresrvResFn`	`-PReserve`
`ncelabPulseE`	`-PULSE_E`
`ncelabPulseIntE`	`-PULSE_INT_E`
`ncelabPulseIntR`	`-PULSE_INT_R`

ams.env Variable	ncelab Command Option
`ncelabPulseR`	`-PULSE_R`
`ncelabRelax`	`-Relax`
`ncelabSdfCmdFile`	`-SDF_Cmd_file`
`ncelabSdfNocheckCelltype`	`-SDF_NOCHECK_Celltype`
`ncelabSdfNoHeader`	`-NO_Sdfa_header`
`ncelabSdfNoWarnings`	`-SDF_NO_Warnings`
`ncelabSdfprecision`	`-SDF_Precision`
`ncelabSdfverbose`	`-SDF_Verbose`
`ncelabSdfWorstcaseRounding`	`-SDF_Worstcase_rounding`
`ncelabsolverInfo`	
`ncelabStatus`	`-STatus`
`ncelabTopLvlGeneric`	`-GENEric`
`ncelabUpdate`	`-UPDate`
`ncelabUse5x4vhdl`	`-USE5X4VHdl`
`ncelabUseAddArgs`	None. Determines whether additional specified arguments are used on the `ncelab` command.
`ncelabUseAfile`	`-AFile`
`ncelabUseExtendtcheckdatalimit`	`-EXTEND_TCHECK_Data_limit`
`ncelabUseExtendtcheckreferencelimit`	`-EXTEND_TCHECK_Reference_limit`
`ncelabUseGenafile`	None. Determines whether the option to create the access file is included on the `ncelab` command.
`ncelabUseGeneric`	None. Determines whether the option to use the generic value is included on the `ncelab` command.
`ncelabUsePulseE`	`-PULSE_E`
`ncelabUsePulseIntE`	`-PULSE_INT_E`
`ncelabUsePulseIntR`	`-PULSE_INT_R`

ams.env Variable	**ncelab Command Option**
`ncelabUsePulseR`	`-PULSE_R`
`ncelabUseSdfprecision`	`-SDF_Precision`
`ncelabV93`	`-V93`
`ncelabVipdelay`	`-VIPDMAx, -VPIDMIn`

For information about `ncelab` command options, see Chapter 8 of *NC-Verilog Simulator Help*.

ncsimArguments

Specifies additional arguments to be passed to the `ncsim` simulator.

Syntax

`amsDirect.prep ncsimArguments string "``arguments`**`"`**

Value

`arguments`	One or more arguments to be passed to the `ncsim` simulator. The default is an empty string.

Description

```
ncsim configLib.cell:view -analogcontrol fileName -amslic
```

Illustrates the form of the default command that AMS Designer uses to run the simulator,

If *`arguments`* is an empty string, the `ncsim` command includes just the arguments listed on the *Simulator* pane of the AMS Option window. You can use the `ncsimArguments` variable with a non-empty string to pass additional arguments to the simulator.

Example

```
amsDirect.prep ncsimArguments string "-status"
```

Adds the `-status` argument to the other arguments normally used on the `ncsim` command.

ncsimEpulseNoMsg through ncsimExtassertmsg

These `ams.env` `ncsim*` variables correspond to `ncsim` command options as follows:

ams.env Variable	ncsim Command Option
`ncsimEpulseNoMsg`	`-EPulse_no_msg`
`ncsimExtassertmsg`	`-EXTassertmsg`

For information about `ncsim` command options, see Chapter 9 of *NC-Verilog Simulator Help*.

ncsimGUI

Controls whether the simulator runs with a graphical user interface (GUI).

Syntax

```
amsDirect.prep ncsimGUI boolean t | nil
```

Values

`t`	Opens the GUI when the simulator runs. This is the default.
`nil`	The GUI does not open. Depending on the value of the `ncsimTcl` variable, either the Tcl interface opens or the simulator runs in batch mode.

Examples

```
amsDirect.prep ncsimGUI boolean t
```

Directs the environment to open the GUI when the simulator runs.

ncsimLoadvpi through ncsimStatus

These `ams.env` `ncsim*` variables correspond to `ncsim` command options as follows:

ams.env Variable	ncsim Command Option
`ncsimLoadvpi`	`-LOADVPi`
`ncsimLogFileAction`	`-LOGfile`, `-NOLOg`, `-APPEND_Log`
`ncsimMaxErrors`	`-ERrormax`
`ncsimMessages`	`-Messages`
`ncsimNeverwarn`	`-NEverwarn`
`ncsimNocifcheck`	`-NOCIfcheck`
`ncsimNosource`	`-NOSOurce`
`ncsimNostdout`	`-NOSTdout`
`ncsimNowarn`	`-NOWarn`
`ncsimOmichecklvl`	`-Omicheckinglevel`
`ncsimPlinooptwarn`	`-PLINOOptwarn`
`ncsimPlinowarn`	`-PLINOWarn`
`ncsimProfile`	`-PROFIle`
`ncsimProfthread`	`-PROFThread`
`ncsimRedmem`	`-REdmem`
`ncsimStatus`	`-STATus`

For information about `ncsim` command options, see Chapter 9 of *NC-Verilog Simulator Help*.

ncsimTcl

Controls whether the simulator opens a Tcl command window. A Tcl command window allows text-based interaction with the simulator.

This variable has an effect only when the `ncsimGUI` variable is set to `nil`.

Syntax

```
amsDirect.prep ncsimTcl boolean t | nil
```

Values

`t`	Opens the Tcl command window when the simulator runs.
`nil`	Runs the simulation in batch mode.This is the default.

Example

```
amsDirect.prep ncsimTcl boolean t
```

If the `ncsimGUI` variable is set to `nil`, this example directs the environment to run the simulation in Tcl mode.

ncsimUnbuffered through ncsimUseAddArgs

These `ams.env` `ncsim*` variables correspond to `ncsim` command options as follows:

ams.env Variable	ncsim Command Option
`ncsimUnbuffered`	`-UNbuffered`
`ncsimUpdate`	`-UPdate`
`ncsimUseAddArgs`	None. Determines whether to use additional specified arguments on the `ncsim` command line.

For information about `ncsim` command options, see Chapter 9 of *NC-Verilog Simulator Help*.

ncvhdlArguments

Specifies arguments, in addition to the standard arguments, to be passed to the `ncvhdl` compiler.

Syntax

`amsDirect.vhdl ncvhdlArguments string` `"`*`arguments`*`"`

Value

`arguments`	One or more arguments to be passed to the `ncvhdl` compiler.

Description

By default, when the AMS netlister runs the compiler to compile a VHDL module, it uses the command

```
ncvhdl -use5x -work lib
```

where *`lib`* is the working library.

You can use the `ncvhdlArguments` variable to pass additional arguments to the compiler.

Example

If the working library is `myworklib`, then using the variable

```
amsDirect.vhdl ncvhdlArguments string "-status"
```

runs the `ncvhdl` compiler with the command

```
ncvhdl -use5x -work myworklib -status
```

ncvlogArguments

Specifies additional arguments to be passed to the `ncvlog` compiler.

Syntax

`amsDirect.vlog ncvlogArguments string "`*`arguments`*`"`

Value

`arguments`	One or more arguments to be passed to the `ncvlog` compiler. The default is an empty string.

Description

If *`arguments`* is an empty string, the `ncvlog` command includes just the arguments listed on the *Compiler* pane of the AMS Options window. You can use the `ncvlogArguments` variable with a non-empty string to pass additional arguments to the compiler.

Example

```
amsDirect.vlog ncvlogArguments string "-status"
```

Adds the `-status` argument to the other arguments normally used on the `ncvlog` command.

ncvlogUseAddArgs

Controls whether the additional compiler arguments specified by the `ncvlogArguments` variable are used on the `ncvlog` command.

Syntax

```
amsDirect.vlog ncvlogUseAddArgs boolean  t | nil
```

Values

`t`	The additional compiler arguments specified by the `ncvlogArguments` variable are used.
`nil`	The additional compiler arguments specified by the `ncvlogArguments` variable are *not* used. This is the default.

netClashFormat

Specifies the format to be used to map the names of nets that collide with names of other netlist constructs.

Syntax

amsDirect.vlog netClashFormat string "*format*"

Value

format	All characters, except those listed below, are printed exactly as included in *format*. The following characters have the indicated special meanings.

`%b`	Original name of the net
`%%`	Prints the `%` character

The default value of *format* is `%b_netclash`, which produces a mapped name like `nname_netclash` for a net originally named `nname`.

If the resulting name is illegal in Verilog-AMS, the name is mapped. If the mapped name clashes with the name of another object, the name undergoes collision mapping.

Example

```
amsDirect.vlog netClashFormat string "%b_nclash"
```

Tells AMS netlister to map clashing net names with a suffixed `_nclash`. For example, you have a net `samp` with a name that clashes with an instance named `samp`. The AMS netlister maps the net to the system-generated name `samp_nclash`.

netlistAfterCdfChange

Controls netlist generation for the cellview when the CDF information for the cell is updated from the CDF editor.

Syntax

`amsDirect.vlog netlistAfterCdfChange boolean t | nil`

Values

`t`	Generates netlists for the eligible cellviews of the cell after CDF information is updated (provided that no errors are found while checking CDF data).
`nil`	Does not generate a netlist. This is the default.

Description

```
amsDirect.vlog netlistAfterCdfChange boolean t
```

Tells the AMS netlister to generate a Verilog-AMS netlist for the cell whose CDF is being updated. However, the netlister does not generate a netlist if checking the CDF information reveals any errors.

netlistMode

Controls netlisting.

Syntax

```
amsDirect.prep netlistMode cyclic "none" | "incremental" | "all"
```

Values

`none`	Turns off netlisting.
`incremental`	Netlists cellviews in the hierarchy only if their HDL data is not synchronized with their cellview data. This is the default.
`all`	Netlists all cellviews in the hierarchy, regardless of whether their HDL data is synchronized with their cellview data.

Example

```
amsDirect.prep netlistMode cyclic "all"
```

Tells AMS Designer (working through the AMS netlister) to netlist all the cellviews that can be netlisted.

netlistToRunDir

Controls whether the software writes netlist files to the current run directory or to the master libraries specified by the `cds.lib` file. (For the purpose of this discussion, note that master libraries also include explicit TMP directories and TMP libraries that result from using the `ASSIGN AllLibs` statement in the `cds.lib` file.)

Note: This variable interacts with the `useRunDirNetlistsOnly` variable. Using the default values of the `netlistToRunDir` and `useRunDirNetlistsOnly` variables, the software writes netlists to the master libraries.

Syntax

`amsDirect netlistToRunDir boolean t | nil`

Values

`t`	Writes new netlists to the run directory. Use the `useRunDirNetlistsOnly` variable to specify whether you want netlists in master libraries also considered. **Note:** Set `netlistToRunDir` to `t` if you want to match the behavior of the Analog Design Environment (ADE).
`nil`	Writes new netlists to the master libraries that hold the netlisted schematics. This is the default value and the default behavior for AMS Designer.

Controlling Netlisting into Run Directories

Using the default values of the `netlistToRunDir` and `useRunDirNetlistsOnly` variables, the software writes netlists to the master libraries.

ADE has a different default behavior, where netlists are written to run directories. To match this behavior in AMS Designer, set `netlistToRunDir` to `t` (and use the default value for the `useRunDirNetlistsOnly` variable).

Alternatively, you can establish a behavior that facilitates using shared netlists located in master libraries without the need to update the `cds.lib` file with `TMP` assignments for libraries that have out-of-date netlists. In this shared netlist approach, netlists in master libraries can be generated once and then used in multiple configurations and simulations, and by multiple users.To establish this behavior, set `netlistToRunDir` to `t` and set `useRunDirNetlistsOnly` to `nil`.

There is an additional difference to be aware of when you share netlists between ADE and AMS Designer. In AMS Designer, the default value for the `amsDefinitionViews` variable is `""` but in ADE, the default value is `"symbol schematic extracted"`. (See "amsDefinitionViews" on page 374.) Because the `amsDefinitionViews` setting determines module port ordering and bus details, using these default values means that a netlist created by AMS Designer differs from the netlist created by ADE for the same design.

If you use, for example, the *Tools – AMS – Netlist* command in the CIW to fill a master library with netlists and later, with `useRunDirNetlistsOnly` set to `nil`, you run design preparation, nothing more is netlisted. However, if you use the same master library in ADE, with `useRunDirNetlistsOnly` set to `nil,` and then netlist the design, all of the netlists are regenerated into the run directory. That difference in behavior occurs because the `amsDefinitionViews` variables in the two environment are different by default. The solution, which allows you to take full advantage of netlist sharing, is to set the `amsDefinitionViews` variable in the AMS Designer environment to `"symbol schematic extracted"`.

Example 1

You add the following to your `ams.env` file before running AMS Designer.

```
amsDirect netlistToRunDir boolean t
amsDirect useRunDirNetlistsOnly boolean t
```

The `netlistToRunDir` variable specifies that netlists (and other intermediate files produced by netlisting) are to be written to the run directory.

The `useRunDirNetlistsOnly` variable specifies that the program is to look for netlists only in the run directory. If a required netlist is not found, the netlist is created in the run directory and then used. To determine whether incremental netlisting is necessary, only netlists found in the run directory are considered and netlists that might exist in the master library are ignored.

The default value for `useRunDirNetlistsOnly` is `t`, so you could omit the second statement in the example and the behavior would be the same.

Example 2

You add the following to your `ams.env` file before running AMS Designer.

```
amsDirect netlistToRunDir boolean t
amsDirect useRunDirNetlistsOnly boolean nil
```

The `netlistToRunDir` variable specifies that netlists (and other intermediate files produced by netlisting) are to be written to the run directory.

The `useRunDirNetlistsOnly boolean nil` value specifies that the program is allowed to look for netlists in master libraries.

Example 3

You do not include the `netlistToRunDir` or `useRunDirNetlistsOnly` variables in your `ams.env` file or you set those variables to their default values.

```
amsDirect netlistToRunDir boolean nil
amsDirect useRunDirNetlistsOnly boolean t
```

The `nil` value for the `netlistToRunDir` variable indicates that all netlist information is assumed to be in and is written to the master libraries specified by the `cds.lib` file. You must have write permission to the master libraries (or have corresponding writable TMP directories), or netlisting fails. This behavior is the same behavior AMS Designer had before netlisting to the run directory was supported.

The `useRunDirNetlistsOnly` variable has no effect when `netlistToRunDir` is set to `nil`.

netlistUDFAsMacro

Determines whether user-defined functions (UDFs) are flagged as errors or are converted to macro references. AMS Designer does not provide a graphical interface for setting this variable.

Syntax

`amsDirect.vlog netlistUDFAsMacro boolean t | nil`

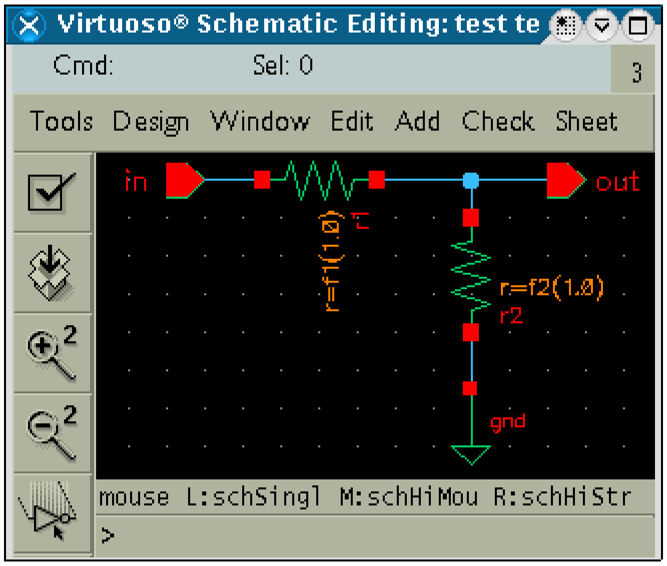
Values

`t`	Specifies that the netlister is to convert UDFs to macro references.
`nil`	Specifies that the netlister is to flag UDFs with errors and not produce a netlist. This is the default.

Description

`amsDirect.vlog netlistUDFAsMacro boolean t`

Tells the AMS netlister to convert UDFs to macro references. For example, the following schematic uses UDFs to specify the value of the resistors.


Virtuoso® Schematic Editing: test te
Cmd:
Sel: 0
3
Tools Design Window Edit Add Check Sheet
in
out
r=f1(1.0)
r1
r=f2(1.0)
r2
gnd
mouse L:schSingl M:schHiMou R:schHiStr
>

The netlister uses equivalent macro references in the netlist.

```
resistor #(.r(‘f1(1.0)))  (* ... *) R1 ( ... );
resistor #(.r(‘f2(1.0)))  (* ... *) R1 ( ... );
```

The referenced macros must be defined in an accessible location, as described in “Netlisting User-Defined Functions” on page 626.

neverwarn

Suppresses all warning messages.

Syntax

```
amsDirect.vlog neverwarn boolean  t | nil
```

Values

`t`	Suppresses all warning messages.
`nil`	Warning messages are displayed. This is the default.

noline

Tells the compiler not to locate the source line of errors, potentially improving performance.

Syntax

```
amsDirect.vlog noline boolean  t | nil
```

Values

`t`	The compiler does *not* locate the source line of errors.
`nil`	The compiler locates the source line of errors. This is the default.

nomempack

Prepares design units for access by the PLI routine `tf_nodeinfo`.

Syntax

`amsDirect.vlog nomempack boolean  t | nil`

Values

`t`	Prepares design units for access by the PLI routine `tf_nodeinfo`.
`nil`	Does not prepare design units for access by the PLI routine `tf_nodeinfo`. This is the default.

Example

```
amsDirect.vlog nomempack boolean t
```

Tells AMS Designer to compile Verilog files with the `-nomempack` option. As a result, the generated command might look like this.

```
ncvlog -nomempack
```

nopragmawarn

Suppresses warning messages related to pragmas.

Syntax

```
amsDirect.vlog nopragmawarn boolean  t | nil
```

Values

`t`	Suppresses warning messages related to pragmas.
`nil`	Displays warning messages related to pragmas. This is the default.

nostdout

Suppresses printing of output to the screen but does not change what is written to the log file.

Syntax

```
amsDirect.vlog nostdout boolean  t | nil
```

Values

`t`	Suppresses printing of output to the screen.
`nil`	Prints output to the screen. This is the default.

nowarn

Suppresses warning messages that have specified codes.

Syntax

`amsDirect.vlog nowarn string "`*`msgcodes`*`"`

Value

`msgcodes`	The default is an empty string.

paramDefVals

Specifies a list of Verilog-AMS module parameters and their associated defaults.

Syntax

```
amsDirect.vlog paramDefVals string "{ ( [type:] parameter_name=value ) } "
```

Values

type	The type of *parameter_name*: `integer` or `real`.
parameter_name	A Verilog-AMS module parameter.
value	The default associated with *parameter_name*.

Description

The AMS netlister uses this list of parameters when it generates the parameter list for the cellview that is being netlisted and defaults for one or more of those parameters do not appear in the design data. This variable does not affect the generation of the list parameters that are passed into an instantiated cell.

The AMS netlister assumes that all parameter names are in the cellview name space.

The default for `paramDefVals` is an empty string.

Example

```
amsDirect.vlog paramDefVals string "(real:l=1.0)(count=0)(w=1.1)"
```

Specifies defaults for the parameters `l`, `count`, and `w`. The `l` parameter is specified as a real.

paramGlobalDefVal

Specifies a global module parameter default to be used when a CDF value is not available and the AMS netlister cannot find the parameter name in the `paramDefVals` variable.

Syntax

```
amsDirect.vlog paramGlobalDefVal string "value"
```

Value

value	Specifies the global module parameter default to be used. The default is 0.

Description

The AMS netlister uses this global value only when it generates the parameter list for the cellview that is being netlisted and defaults for one or more of those parameters do not appear in the design data. This variable does not affect the generation of the list parameters that are passed into an instantiated cell.

pragma

Parses pragmas contained in HDL source files.

Syntax

```
amsDirect.vlog pragma boolean  t | nil
```

Values

`t`	The compiler parses pragmas contained in HDL source files.
`nil`	The compiler does not parse pragmas contained in HDL source files.

Example

```
amsDirect.vlog pragma boolean t
```

Tells AMS Designer to compile Verilog files with the `-pragma` option. As a result, the generated command might look like this.

```
ncvlog -pragma
```

useRunDirNetlistsOnly

Controls whether the netlister considers netlists in the run directory only or also, if necessary, considers netlists it finds in the master libraries defined by `cds.lib` files. (For the purpose of this discussion, master libraries also include explicit TMP directories and TMP libraries that result from using the `ASSIGN AllLibs` statement in the `cds.lib` file.)

Note: The `useRunDirNetlistsOnly` variable has no effect when the `netlistToRunDir` variable is set to `nil`. Using the default values of the `netlistToRunDir` and `useRunDirNetlistsOnly` variables, the software writes netlists to the master libraries.

Syntax

`amsDirect useRunDirNetlistsOnly boolean t | nil`

Values

`t`	Specifies that the netlister is to consider, and, if necessary, update, netlists only from the run directory. Netlists in master libraries are ignored as the netlister determines whether incremental netlisting is needed for any particular object. This is the default.
`nil`	Specifies that the netlister is to first consider, and, if necessary, update, netlists found in the run directory. If a required netlist does not exist in the run directory, the netlister is then to consider netlists from the master libraries specified by the `cds.lib` file. If the necessary netlists are not found in the master libraries or is out of date, the netlister generates corresponding new netlists in the run directory.

Description

As noted, setting this variable to `nil` allows the program to consider netlists located in explicit TMP and master libraries. More specifically, a setting of `nil` means that the program is to

1. Look first for the netlist in the run directory.

   If the netlist is found and is up to date, use that netlist. If the netlist is found, but is not up to date, update it and then use it.

2. If the netlist is not found in the run directory, look in the explicit TMP directory (if one is used).

   If the netlist is found and is up to date, use that netlist. If the netlist is found but is not up to date, create an up-to-date netlist in the run directory and use it.

3. If the netlist is not found in the explicit TMP directory, look in the master library.

   If the netlist exists in the master library and is up to date, use that information. If the netlist is not found or is found but is not up to date, create an up-to-date netlist in the run directory and use it.

Example

See "Example 1" on page 459, "Example 2" on page 459, and "Example 3" on page 460.

processViewNames

Specifies the names of cellviews that are to be netlisted.

Syntax

amsDirect.vlog processViewNames string "*list_of_view_names*"

Value

list_of_view_names	A list of view names separated by spaces. Cellviews with these names are netlisted. The default is an empty string.

Description

The following conditions trigger netlisting.

- Changes to cellviews included in *list_of_view_names* while netlisting is enabled.
- Changes to the CDF of cells containing any of the cellviews included in *List_of_view_names* while the `netlistAfterCdfChange` variable is set to `t`.

Using this variable is an alternative to specifying the eligible view types with `amsEligibleViewTypes` and the views to exclude from netlisting with `excludeViewNames`.

Example

```
amsDirect.vlog processViewNames string "sch1 sch[3-4]"
```

prohibitCompile

Controls the automatic compilation of the generated netlist.

Syntax

```
amsDirect.vlog prohibitCompile boolean t | nil
```

Values

`t`	Prohibits the automatic compilation of the generated netlist.
`nil`	Automatically compiles the netlist. This is the default.

Description

By default, the AMS netlister automatically compiles the netlist. If you specify `t`, the netlister does not automatically compile the netlist.

runNcelab

Controls whether the elaborator runs when you click *Run* in the AMS Run Simulation form.

Syntax

```
amsDirect.prep runNcelab boolean t | nil
```

Values

`t`	Runs the elaborator. This is the default.
`nil`	Does not run the elaborator.

Example

```
amsDirect.prep runNcelab boolean nil
```

Tells AMS Designer not to run the elaborator.

runNcsim

Controls whether the simulator runs when you click *Run* in the AMS Run Simulation form.

Syntax

`amsDirect.prep runNcsim boolean t | nil`

Values

`t`	Runs the simulator. This is the default.
`nil`	Does not run the simulator.

Example

```
amsDirect.prep runNcsim boolean nil
```

Tells AMS Designer not to run the simulator when you click *Run* in the AMS Run Simulation form.

scaddlglblopts

Specifies options to be appended to the end of the `options` card in the generated simulation control file.

Syntax

`amsDirect.simcntl scaddlglblopts string "`*`options`*`"`

Value

`options`	A list of options separated by spaces.

Description

This variable specifies additional options to be added to the `options` statement in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl scaddlglblopts string "rawfile = \"/hm/kat/amsAnalysis\""
```

In response, the generated simulation control file contains

```
amsOptions options
+ gmin_check = all
+ inventory = detailed
+ rawfile = "/hm/kat/amsAnalysis"
```

scaddltranopts

Specifies additional options to be appended to the end of the `tran` card in the simulation control file.

Syntax

`amsDirect.simcntl scaddltranopts string "`*`options`*`"`

Value

`options`	A list of options separated by spaces.

Description

This variable specifies additional options to be appended to the `tran` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl scaddltranopts string "outputstart=0.0005"
```

In response, the generated simulation control file contains

```
amsAnalysis tran
+ stop = 0.001
+ outputstart=0.0005
```

scale

Specifies the scaling factor for device instances.

Syntax

`amsDirect.simcntl scale string "`*`factor`*`"`

Value

`factor`	The scaling factor for device instances. The default is 1.0.

Description

This variable determines the value assigned to the `options scale` option in a generated simulation control file.

scalem

Specifies the scaling factor for models.

Syntax

`amsDirect.simcntl scalem string "`*`factor`*`"`

Value

`factor`	The scaling factor for models. The default is 1.0.

Description

This variable determines the value assigned to the `options scalem` option in a generated simulation control file.

scannotate

Specifies the degree of annotation for the transient analysis.

Syntax

`amsDirect.simcntl` scannotate cyclic "sweep" | "no" | "title" | "status" | "steps"

Values

`sweep`

`no`

`title`

`status`

`steps`

scapprox

Specifies that approximate models are to be used. The difference between approximate and exact models is generally very small.

Syntax

`amsDirect.simcntl scapprox boolean t | nil`

Values

`t`	The simulator uses approximate models.
`nil`	The simulator uses exact models. This is the default.

Description

This variable determines the value assigned to the `options approx` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scapprox        boolean t
```

In response, the generated simulation control file contains

```
amsOptions options
+ approx = yes
```

scaudit

Specifies the extent of the information to be returned about the time required by various parts of the simulation.

Syntax

`amsDirect.simcntl scaudit cyclic "detailed" | "no" | "brief" | "full"`

Values

`detailed`

`no`

`brief`

`full`

Description

This variable determines the value assigned to the `options audit` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scaudit cyclic  "full"
```

In response, the generated simulation control file contains

```
amsOptions options
+ audit = full
```

scchecksmt

Unsupported by AMS Designer.

Performs a check analysis at any point in a simulation to be sure that the value of component parameters are reasonable. For more information, see “The check Statement” in the “Control Statements” chapter of the *Virtuoso Spectre Circuit Simulator User Guide*.

sccmin

Specifies the minimum capacitance from each node to ground.

Syntax

`amsDirect.simcntl sccmin string "`*`capacitance`*`"`

Value

`capacitance`	The minimum capacitance from each node to ground.

Description

This variable determines the value assigned to the `tran cmin` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       sccmin  string  "0.1"
```

In response, the generated simulation control file contains

```
amsAnalysis tran
+ stop = 0.003
+ cmin = 0.1
```

sccompatible

Specifies a simulator. AMS Designer changes device models to improve consistency with the models in the specified simulator.

Syntax

```
amsDirect.simcntl sccompatible cyclic "spectre" | "spice2" | "spice3" |
     "hspice" | "spiceplus"
```

Values

`spectre`

`spice2`

`spice3`

`hspice`

`spiceplus`

Description

This variable determines the value assigned to the `options compatible` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       sccompatible    cyclic  "spiceplus"
```

In response, the generated simulation control file contains

```
amsOptions options
+ compatible = spiceplus
```

scdebug

Prints debugging information.

Syntax

`amsDirect.simcntl scdebug boolean t | nil`

Values

`t`	The simulator prints debugging information.
`nil`	The simulator does *not* print debugging information. This is the default.

Description

This variable determines the value assigned to the `options debug` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scdebug boolean t
```

In response, the generated simulation control file contains

```
amsOptions options
+ debug = yes
```

scdiagnose

Prints information that might help diagnose accuracy and convergence problems.

Syntax

`amsDirect.simcntl scdiagnose boolean t | nil`

Values

`t`	The simulator prints diagnostic information.
`nil`	The simulator does *not* print diagnostic information. This is the default.

Description

This variable determines the value assigned to the `options diagnose` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scdiagnose      boolean t
```

In response, the generated simulation control file contains

```
amsOptions options
+ diagnose = yes
```

scdigits

Specifies the number of digits used when printing numbers.

Syntax

```
amsDirect.simcntl scdigits int digits
```

Value

digits	The number of digits used when printing numbers

Description

This variable determines the value assigned to the `options digits` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scdigits        int     8
```

In response, the generated simulation control file contains

```
amsOptions options
+ digits = 8
```

scerror

Prints error messages.

Syntax

`amsDirect.simcntl scerror boolean t | nil`

Values

`t`	The simulator prints error messages. This is the default.
`nil`	The simulator does *not* print error messages.

Description

This variable determines the value assigned to the `options error` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scerror boolean nil
```

In response, the generated simulation control file contains

```
amsOptions options
+ error = no
```

scerrpreset

Specifies a collection of parameter settings for the analysis. The collection you specify affects simulation speed and accuracy.

Syntax

`amsDirect.simcntl scerrpreset cyclic "moderate" | "conservative" | "liberal"`

Values

`moderate`	Simulation accuracy approximates a SPICE2 style simulator.
`conservative`	Simulation is the most accurate but also the slowest. This setting is appropriate for sensitive analog circuits.
`liberal`	Simulation is fast but less accurate. This setting is suitable for digital circuits or for analog circuits that have only short time constants.

Description

This variable determines the value assigned to the `tran errpreset` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scerrpreset     cyclic  "conservative"
```

In response, the generated simulation control file includes

```
amsAnalysis tran
+ stop = 0.001
+ errpreset = conservative
```

scfastbreak

Specifies the evaluation method to use for VHDL-AMS `break` statements.

Syntax

`amsDirect.simcntl scfastbreak boolean t | nil`

Values

`t`	Requests a method of evaluating VHDL-AMS `break` statements that is often faster than the default method. Under some circumstances, the method chosen by setting `scfastbreak` to `t` does not comply with the VHDL-AMS standard. Possible non-compliance with the standard arises when the `break` statement is associated with a discontinuity that causes a zero-delay `Q'ABOVE` event. The `Q'ABOVE` event might be reported with a tiny delay, rather than the expected zero delay. This method might also produce simulation results that differ slightly from the results obtained when the default method is used.
`nil`	Requests the `break` statement evaluation method that complies strictly with the VHDL-AMS standard. This is the default.

Example

```
amsDirect.simcntl scfastbreak boolean t
```

Directs the simulator to use the potentially faster method of evaluating VHDL-AMS `break` statements.

scglobalminr

Specifies the threshold below which resistance inside devices is ignored.

Syntax

`amsDirect.simcntl scglobalminr string "`*`resistance`*`"`

Value

`resistance`	The threshold resistance.

Description

This variable determines the value assigned to the `options minr` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scglobalminr string  "1e-5"
```

In response, the generated simulation control file contains

```
amsOptions options
+ minr = 1e-5
```

scgmin

Specifies the minimum conductance across each nonlinear device.

Syntax

`amsDirect.simcntl scgmin string "`*`conductance`*`"`

Value

`conductance`	The minimum conductance.

Description

This variable determines the value assigned to the `options gmin` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scgmin  string  "1e-11"
```

In response, the generated simulation control file contains

```
amsOptions options
+ gmin = 1e-11
```

scgmincheck

Specifies how the effect of `scgmin` is to be reported.

Syntax

`amsDirect.simcntl scgmincheck cyclic "max_v_only" | "max_only" | "no" | "all"`

Values

`max_v_only`

`max_only`

`no`

`all`

Description

This variable determines the value assigned to the `options gmin_check` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scgmincheck     cyclic  "max_only"
```

In response, the generated simulation control file contains

```
amsOptions options
+ gmin_check = max_only
```

schomotopy

Specifies the method to use if convergence fails on the initial DC analysis attempt.

Syntax

```
amsDirect.simcntl schomotopy cyclic "all" | "none" | "gmin" | "source" |
     "dptran" | "ptran"
```

Values

`all`

`none`

`gmin`

`source`

`dptran`

`ptran`

Description

This variable determines the value assigned to the `options homotopy` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       schomotopy      cyclic  "source"
```

In response, the generated simulation control file contains

```
amsOptions options
+ homotopy = source
```

sciabstol

Specifies the absolute tolerance for differences in the computed values of the currents in the last two iterations of a solution.

Syntax

`amsDirect.simcntl sciabstol string "`*`tolerance`*`"`

Value

`tolerance`	The absolute tolerance for differences in the computed values of the currents.

Description

This variable determines the value assigned to the `options iabstol` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       sciabstol       string  "1e-10"
```

In response, the generated simulation control file contains

```
amsOptions options
+ iabstol = 1e-10
```

scic

Controls the interaction of various methods of setting the initial conditions.

Syntax

`amsDirect.simcntl scic cyclic "all" | "dc" | "node" | "dev"`

Values

`all`	Uses both `ic` statements and `ic` parameters, and `ic` parameters override `ic` statements.
`dc`	Ignores any initial condition specifiers, and uses the DC solution.
`node`	Uses `ic` statements, and ignores `ic` parameters on capacitors and inductors.
`dev`	Uses `ic` parameters on capacitors and inductors, and ignores `ic` statements.

Description

This variable determines the value assigned to the `tran ic` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scic    cyclic  "node"
```

In response, the generated simulation control file contains

```
amsAnalysis tran
+ stop = 0.001
+ ic = node
```

scicstmt

Specifies initial conditions for nodes and devices in the design.

Syntax

`amsDirect.simcntl scicstmt string "`*`ic_conditions`*`"`

Values

`ic_conditions`	A list of conditions for nodes and devices.

Description

This variable determines the value assigned to the `ic` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl scicstmt string "7=0 out=1 OpAmp1.comp=5 L1:1=1.0u"
```

In response, the generated simulation control file contains

```
ic 7=0 out=1 OpAmp1.comp=5 L1:1=1.0u
```

scignshorts

Tells the simulator to ignore shorted components silently.

Syntax

`amsDirect.simcntl scignshorts boolean t | nil`

Values

`t`	The simulator ignores shorted components silently.
`nil`	The simulator reports shorted components. This is the default.

Description

This variable determines the value assigned to the `options ignshorts` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scignshorts     boolean t
```

In response, the generated simulation control file contains

```
amsOptions options
+ ignshorts = yes
```

scinfo

Prints information messages.

Syntax

`amsDirect.simcntl scinfo boolean t | nil`

Values

`t`	The simulator prints information messages. This is the default.
`nil`	The simulator does *not* print information messages.

Description

This variable determines the value assigned to the `options info` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scinfo  boolean nil
```

In response, the generated simulation control file contains

```
amsOptions options
+ info = no
```

scinventory

Specifies the extent of the information to be returned about the components used in the simulation.

Syntax

`amsDirect.simcntl scinventory cyclic "no" | "brief" | "detailed"`

`no`

`brief`

`detailed`

Description

This variable determines the value assigned to the `options inventory` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scinventory     cyclic  "brief"
```

In response, the generated simulation control file contains

```
amsOptions options
+ inventory = brief
```

sclimit

Specifies the limiting algorithm used to aid DC convergence.

Syntax

```
amsDirect.simcntl sclimit cyclic "dev" | "delta" | "log"
```

Values

`dev`

`delta`

`log`

Description

This variable determines the value assigned to the `options limit` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       sclimit cyclic  "delta"
```

In response, the generated simulation control file contains

```
amsOptions options
+ limit = delta
```

sclteratio

Specifies the ratio to use to compute LTE tolerances from Newton tolerance.

Syntax

`amsDirect.simcntl sclteratio string "`*`ratio`*`"`

Values

`ratio`	The ratio to use to compute LTE tolerances from Newton tolerance.

Description

This variable determines the value assigned to the `tran lteratio` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       sclteratio      string  "8.0"
```

In response, the generated simulation control file contains

```
amsAnalysis tran
+ stop = 0.003
+ lteratio = 8.0
```

scmacromod

Indicates that the circuit contains macromodels. Sometimes specifying this information improves performance.

Syntax

`amsDirect.simcntl scmacromod boolean t | nil`

Values

`t`	Indicates that the circuit contains macromodels.
`nil`	Indicates that the circuit does *not* contain macromodels. This is the default.

Description

This variable determines the value assigned to the `options macromodels` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scmacromod      boolean t
```

In response, the generated simulation control file contains

```
amsOptions options
+ macromodels = yes
```

scmaxiters

Specifies the maximum number of iterations per time step.

Syntax

`amsDirect.simcntl scmaxiters int` *`maxiters`*

Values

`maxiters`	The maximum number of iterations per time step.

Description

This variable determines the value assigned to the `tran maxiters` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scmaxiters      int     10
```

In response, the generated simulation control file contains

```
amsAnalysis tran
+ stop = 0.001
+ maxiters = 10
```

scmaxnotes

Specifies the maximum number of times any particular notice will be issued per analysis.

Syntax

`amsDirect.simcntl scmaxnotes int` *`maxnotes`*

Values

`maxnotes`	The maximum number of times any particular notice will be issued per analysis.

Description

This variable determines the value assigned to the `options maxnotes` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scmaxnotes      int     15
```

In response, the generated simulation control file contains

```
amsOptions options
+ maxnotes = 15
```

scmaxnotestologfile

scmaxrsd

Specifies the threshold below which parasitic node reduction occurs.

Syntax

`amsDirect.simcntl scmaxrsd string "`*`threshold`*`"`

Values

`threshold`	The default value is an empty string, `""` equivalent to a value of zero.

Example

```
amsDirect.simcntl scmaxrsd string "1e-8"
```

Tells the AMS simulator to remove parasitic nodes with resistances smaller than `1e-8`. The simulator then uses a linear correction to model the resistance.

scmaxstep

Specifies the maximum time step.

Syntax

```
amsDirect.simcntl scmaxstep string "maxstep"
```

Values

maxstep	The maximum time step.

Description

This variable determines the value assigned to the `tran maxstep` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scmaxstep       string  ".00002"
```

In response, the generated simulation control file contains

```
amsAnalysis tran
+ stop = 0.003
+ maxstep = .00002
```

scmaxwarn

Specifies the maximum number of times any particular warning will be issued per analysis.

Syntax

`amsDirect.simcntl scmaxwarn int` *`maxwarn`*

Values

`maxwarn`	The maximum number of times any particular warning will be issued per analysis.

Description

This variable determines the value assigned to the `options maxwarn` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scmaxwarn       int     20
```

In response, the generated simulation control file contains

```
amsOptions options
+ maxwarns = 20
```

scmaxwarntologfile

scmethod

Specifies the integration method to use.

Syntax

```
amsDirect.simcntl scmethod cyclic "traponly" | "gear2" | "euler" | "trap" |
     "gear2only" | "trapgear2"
```

Values

`traponly`	Uses almost exclusively the trapezoidal rule method.
`gear2`	Uses the backward-Euler and second-order Gear method.
`euler`	Uses exclusively the backward-Euler method.
`trap`	Uses the backward-Euler and the trapezoidal rule methods.
`gear2only`	Uses almost exclusively Gear's second-order backward-difference method.
`trapgear2`	Allows all three integration methods to be used.

Description

This variable determines the value assigned to the `tran method` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scmethod        cyclic  "traponly"
```

In response, the generated simulation control file contains

```
amsAnalysis tran
+ stop = 0.001
+ method = traponly
```

scmodelevaltype

Specifies whether standard SPICE-like equations or table (accelerated) models are used to evaluate bsim3v3 and bsim4 models.

Syntax

`amsDirect.simcntl scmodelevaltype cyclic "s" | "a"`

Values

`s`	Instructs the simulator not to use table models for any instances. This is the default.
`a`	Instructs the simulator to use table (accelerated) models whenever possible. This global option applies to the entire simulated design. You can override this instruction on specific model cards by setting `mos_method = s` as an option on those cards.

Description

This variable determines the value assigned to the `mos_method` option of the `options` statement in a generated analog simulation control file. AMS Designer writes the `mos_method` option to the analog simulation control file only when the `scusemodeleval` variable is set to `t`. For additional information, see "scusemodeleval" on page 552.

Example

You set the variables

```
amsDirect.simcntl       scusemodeleval boolean t
amsDirect.simcntl       scmodelvaltype cyclic  "a"
```

In response, the generated analog simulation control file contains

```
amsOptions options
+ mos_method = a
```

scmosvres

Specifies the voltage increment for the mosfet table model interpolation grid. Smaller values reduce the interpolation error, but might increase memory consumption. A value of 20mV is appropriate for analog circuits that are extremely sensitive to small model parameter variations, and subthreshold and substrate currents.

Syntax

`amsDirect.simcntl scmosvres string "`*`vresolution`*`"`

Values

vresolution	The default value is `0.05`.

Example

```
amsDirect.simcntl scmosvres string "0.02"
```

Sets the interpolation grid value to 20mV.

scnarrate

Narrates the simulation.

Syntax

`amsDirect.simcntl scnarrate boolean t | nil`

Values

`t`	The simulator narrates the simulation. This is the default.
`nil`	The simulator does *not* narrate the simulation.

Description

This variable determines the value assigned to the `options narrate` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scnarrate       boolean nil
```

In response, the generated simulation control file contains

```
amsOptions options
+ narrate = no
```

scnotation

Specifies the notation to be used when displaying real numbers.

Syntax

```
amsDirect.simcntl scnotation cyclic "eng" | "sci" | "float"
```

Values

`eng`	Uses engineering notation.
`sci`	Uses scientific notation.
`float`	Uses floating point notation.

Description

This variable determines the value assigned to the `options notation` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scnotation      cyclic  "sci"
```

In response, the generated simulation control file contains

```
amsOptions options
+ notation = sci
```

scnote

Prints notice messages.

Syntax

`amsDirect.simcntl scnote boolean t | nil`

Values

`t`	The simulator prints notice messages. This is the default.
`nil`	The simulator does *not* print notice messages.

Description

This variable determines the value assigned to the `options note` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scnote  boolean nil
```

In response, the generated simulation control file contains

```
amsOptions options
+ note = no
```

scopptcheck

Specifies that operating point parameters are to be checked against soft limits.

Syntax

`amsDirect.simcntl scopptcheck boolean t | nil`

Values

`t`	The operating point parameters are checked against soft limits.
`nil`	The operating point parameters are *not* checked against soft limits.

Description

This variable determines the value assigned to the `options opptcheck` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scopptcheck     boolean nil
```

In response, the generated simulation control file contains

```
amsOptions options
+ opptcheck = no
```

scpivabs

Specifies the absolute pivot threshold.

Syntax

`amsDirect.simcntl scpivabs string "`*`threshold`*`"`

Values

`threshold`	The absolute pivot threshold.

Description

This variable determines the value assigned to the `options pivabs` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scpivabs        string  "0.5"
```

In response, the generated simulation control file contains

```
amsOptions options
+ pivabs = 0.5
```

scpivotdc

Specifies that numeric pivoting be used on every iteration of DC analysis.

Syntax

```
amsDirect.simcntl scpivotdc boolean t | nil
```

Values

`t`	The simulator uses numeric pivoting on every iteration of DC analysis.
`nil`	The simulator does not use numeric pivoting on every iteration of DC analysis. This is the default.

Description

This variable determines the value assigned to the `options pivotdc` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scpivotdc       boolean t
```

In response, the generated simulation control file contains

```
amsOptions options
+ pivotdc = yes
```

scpivrel

Specifies the relative pivot threshold.

Syntax

`amsDirect.simcntl scpivrel string "`*`threshold`*`"`

Values

`threshold`	The relative pivot threshold.

Description

This variable determines the value assigned to the `options pivrel` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scpivrel        string  "1e-7"
```

In response, the generated simulation control file contains

```
amsOptions options
+ pivrel = 1e-7
```

scquantities

Specifies the extent of the information to be returned about quantities.

Syntax

`amsDirect.simcntl scquantities cyclic "no" | "min" | "full"`

Values

`no`

`min`

`full`

Description

This variable determines the value assigned to the `options quantities` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scquantities    cyclic  "full"
```

In response, the generated simulation control file contains

```
amsOptions options
+ quantities = full
```

screadic

Specifies a file that contains initial conditions.

Syntax

`amsDirect.simcntl screadic string "`*`icfile`*`"`

Values

`icfile`	The path and name of a file containing initial conditions.

Description

This variable determines the value assigned to the `tran readic` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl screadic string  "/usr1/test6/SAR_A2D/tutorial_run/icfile"
```

In response, the generated simulation control file contains

```
amsAnalysis tran
+ stop = 0.003
+ readic = "/usr1/test6/SAR_A2D/tutorial_run/icfile"
```

screadns

Specifies a file that contains nodesets.

Syntax

`amsDirect.simcntl screadns string "`*`nsfile`*`"`

Values

`nsfile`	The path and name of a file that contains nodesets.

Description

This variable determines the value assigned to the `tran readns` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl screadns string  "/usr1/test6/SAR_A2D/tutorial_run/nsfile"
```

In response, the generated simulation control file contains

```
amsAnalysis tran
+ stop = 0.003
+ readns = "/usr1/test6/SAR_A2D/tutorial_run/nsfile"
```

screlref

Specifies the reference to use for the relative convergence criteria.

Syntax

```
amsDirect.simcntl screlref cyclic "sigglobal" | "allglobal" | "pointlocal" |
     "alllocal"
```

Values

`sigglobal`	Compares relative errors in each of the circuit signals to the maximum for all signals at any previous point in time.
`allglobal`	Same as `sigglobal` except that it also compares the residues (KCL error) for each node to the maximum of that node's past history.
`pointlocal`	Compares the relative errors in quantities at each node to that node alone.
`alllocal`	Compares the relative errors at each node to the largest values found for that node alone for all past time.

Description

This variable determines the value assigned to the `tran relref` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl      screlref       cyclic  "allglobal"
```

In response, the generated simulation control file contains

```
amsAnalysis tran
+ stop = 0.001
+ relref = allglobal
```

screltol

Specifies the maximum relative tolerance for values computed in the last two iterations of a solution.

Syntax

`amsDirect.simcntl screltol string "`*`tolerance`*`"`

Values

`tolerance`	The maximum relative tolerance for values computed in the last two iterations of a solution. Default: 1e-3

Description

This variable determines the value assigned to the `options reltol` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       screltol        string  "0.15"
```

In response, the generated simulation control file contains

```
amsOptions options
+ reltol = 0.15
```

scrforce

Specifies the resistance to be used when forcing nodesets and node-based initial conditions.

Syntax

`amsDirect.simcntl scrforce string "`*`resistance`*`"`

Values

`resistance`	The resistance to be used when forcing nodesets and node-based initial conditions.

Description

This variable determines the value assigned to the `options rforce` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scrforce        string  "1.5"
```

In response, the generated simulation control file contains

```
amsOptions options
+ rforce = 1.5
```

scscale

Specifies the scaling factor for device instances. This variable is obsolete: use the `scale` variable instead.

Syntax

`amsDirect.simcntl scscale int` *`factor`*

Values

`factor`	The scaling factor for device instances.

Description

This variable determines the value assigned to the `options scale` option in a generated simulation control file.

scscalem

Specifies the scaling factor for models. This variable is obsolete: use the `scalem` variable instead.

Syntax

`amsDirect.simcntl scscalem int` *`factor`*

Values

`factor`	The scaling factor for models.

Description

This variable determines the value assigned to the `options scalem` option in a generated simulation control file.

scscfincfile

Specifies a simulation control file to be included in the simulation control file generated from the options you specify in the GUI.

Syntax

`amsDirect.simcntl scscfincfile string "`*`sim_con_file`*`"`

Values

`sim_con_file`	The path and name of the simulation control file to be included.

Description

AMS Designer

Example

You set the variable

```
amsDirect.simcntl scscfincfile string  "/usr1/test6/SAR_A2D/tutorial_run/fpga.scs"
```

In response, the generated simulation control file contains

```
include "/usr1/test6/SAR_A2D/tutorial_run/fpga.scs"
```

scscftimestamp

A time stamp created by AMS Designer. Do not change this value manually.

Syntax

```
amsDirect.simcntl scscftimestamp string "timestamp"
```

Values

timestamp	A time stamp created by AMS Designer.

Description

AMS Designer uses this variable to track changes made in the simulation control file GUI.

Example

You use the GUI to create a simulation control file. You check the `ams.env` file and find that it contains a timestamp variable similar to the following.

```
amsDirect.simcntl       scscftimestamp  string  "1005580511000"
```

scscfusefileflag

Specifies that AMS simulator is to use an existing simulation control file, rather than a simulation control file created by the GUI.

Syntax

```
amsDirect.simcntl scscfusefileflag boolean t | nil
```

Values

`t`	The AMS simulator uses an existing simulation control file so the GUI for creating a new control file is disabled.
`nil`	The AMS simulator uses a simulation control file created by using the GUI, which is made active.

Description

AMS Designer

Example

You set the variable

```
amsDirect.simcntl       scscfusefileflag        boolean t
```

In response, the GUI for creating a new simulation control file is disabled, and a field is enabled that allows you to specify an existing control file.

scskipcount

Specifies a number of points and directs the simulator to save one point every time it calculates that number of points.

Syntax

`amsDirect.simcntl scskipcount int` *`skipcount`*

Values

`skipcount`	The number of points to be calculated for each saved point.

Description

This variable determines the value assigned to the `tran skipcount` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scskipcount     int     18
```

In response, the generated simulation control file contains

```
amsAnalysis tran
+ stop = 0.001
+ skipcount = 18
```

scskipdc

If `yes`, AMS Designer does not do any DC analysis for transient.

Syntax

```
amsDirect.simcntl scskipdc cyclic "no" | "yes" | "waveless" | "rampup" |
     "autodc"
```

Values

`no`	
`yes`	Skips the DC analysis entirely. The initial solution is the values given in the file you specify by the `screadic` variable, or, if that variable is not given, the values specified on `ic` statements.
`waveless`	
`rampup`	
`autodc`	

Description

This variable determines the value assigned to the `tran skipdc` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scskipdc        cyclic  "waveless"
```

In response, the generated simulation control file contains

```
amsAnalysis tran
+ stop = 0.001
+ skipdc = waveless
```

scskipstart

Specifies a time. The simulator saves all computed data before this time.

Syntax

`amsDirect.simcntl scskipstart string "`*`time`*`"`

Values

`time`	The time before which all computed data is saved.

Description

This variable determines the value assigned to the `tran skipstart` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scskipstart     string  "0.01"
```

In response, the generated simulation control file contains

```
amsAnalysis tran
+ stop = 0.003
+ skipstart = 0.01
```

scskipstop

Syntax

```
amsDirect.simcntl scskipstop string "0.0"
```

Values

The default is 0.0.

scspeed

Specifies the setting for the speed dial on the *Performance* pane of the AMS Options window. The *Speed dial* setting establishes the tradeoff between simulation performance and accuracy by writing the `options speed` parameter to the analog simulation control file. Generally, higher settings result in better performance but with some loss in accuracy.

Syntax

`amsDirect.simcntl scspeed int 0 | 1 | 2 | 3 | 4 | 5 | 6`

Values

`0`	The `options speed` parameter is not written to the simulation control file, effectively turning the speed dial off and allowing the underlying settings to take their default values (unless they are individually overridden). This is the default.
`1`	Writes `options speed = 1` to the analog simulation control file.
`2`	Writes `options speed = 2` to the analog simulation control file.
`3`	Writes `options speed = 3` to the analog simulation control file.
`4`	Writes `options speed = 4` to the analog simulation control file.
`5`	Writes `options speed = 5` to the analog simulation control file.
`6`	Writes `options speed = 6` to the analog simulation control file.

The `scspeed` variable sets values for the following fields in the AMS Options window. If you then change the value in one of these fields, the new value overrides the value set by the `scspeed` variable.

Pane	Field	For more information, see
Tran Analysis	*Error preset*	“scerrpreset” on page 492
Performance	*Node reduction threshold*	“scmaxrsd” on page 510
Convergence/Accuracy	*Reltol*	“screltol” on page 528

Pane	Field	For more information, see
Convergence/Accuracy	*Vabstol*	"scvabstol" on page 553
Convergence/Accuracy	*Iabstol*	"sciabstol" on page 498
Tran Convergence/Accuracy	*Lteratio*	"sclteratio" on page 505
Tran Convergence/Accuracy	*Relref*	"screlref" on page 527
Tran Convergence/Accuracy	*Integration method*	"scmethod" on page 514
Tran Convergence/Accuracy	*Maxstep*	"scmaxstep" on page 511

Example

```
amsDirect.simcntl scspeed int 3
```

Causes the environment to write the following statements to the analog simulation control file.

```
amsOptions options
+ speed = 6
```

In addition, the fields affected by the `scspeed` variable change to reflect the corresponding values.

Field	Value
Error preset	`moderate`
Node reduction threshold	`<Value defaulted>`
Reltol	`<Value defaulted>`
Vabstol	`1e-6`
Iabstol	`1e-12`
Lteratio	`<Value defaulted>`
Relref	`<Default value>`
Integration method	`<Default value>`
Maxstep	`<Value defaulted>`

scspscflag

An internal variable used by AMS Designer. Do not change this variable manually.

Syntax

```
amsDirect.simcntl scspscflag boolean t | nil
```

Values

`t`

`nil`

Description

An internal variable used by AMS Designer.

scstats

Prints analysis statistics.

Syntax

`amsDirect.simcntl scstats boolean t | nil`

Values

`t`	The simulator prints analysis statistics.
`nil`	The simulator does *not* print analysis statistics. This is the default.

Description

This variable determines the value assigned to the `tran stats` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scstats boolean t
```

In response, the generated simulation control file contains

```
amsAnalysis tran
+ stop = 0.003
+ stats = yes
```

scstep

Specifies the minimum time step to use.

Syntax

`amsDirect.simcntl scstep string "`*`minstep`*`"`

Values

`minstep`	The minimum time step to use.

Description

This variable determines the value assigned to the `tran step` option in a generated simulation control file. You might need to set this value to maintain the aesthetics of computed waveforms.

Example

You set the variable

```
amsDirect.simcntl       scstep  string  ".00001"
```

In response, the generated simulation control file contains

```
amsAnalysis tran
+ stop = 0.003
+ step = .00001
```

scstop

Specifies the stop time for the analysis.

Syntax

`amsDirect.simcntl scstop string "`*`stop_time`*`"`

Values

`stop_time`	The stop time.

Description

This variable determines the value assigned to the `stop` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scstop  string  "0.003"
```

In response, the generated simulation control file contains

```
amsAnalysis tran
+ stop = 0.003
```

scstrobedelay

Specifies an offset time relative to the time specified by `scskipstart`.

Syntax

`amsDirect.simcntl scstrobedelay string "`*`offset_time`*`"`

Values

`offset_time`	The offset time.

Description

This variable determines the value assigned to the `tran strobedelay` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scstrobedelay   string  "0.00001"
```

In response, the generated simulation control file contains

```
amsAnalysis tran
+ stop = 0.003
+ strobedelay = 0.00001
```

scstrobeperiod

Specifies an interval. The simulator calculates and saves a data point in each interval.

Syntax

`amsDirect.simcntl scstrobeperiod string "`*`interval`*`"`

Values

`interval`	The interval defining the strobe period.

Description

This variable determines the value assigned to the `tran strobeperiod` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scstrobeperiod  string  "0.0005"
```

In response, the generated simulation control file contains

```
amsAnalysis tran
+ stop = 0.003
+ strobeperiod = 0.0005
```

sctemp

Specifies the circuit temperature.

Syntax

`amsDirect.simcntl sctemp string "`*`temperature`*`"`

Values

`temperature`	The circuit temperature in degrees Celsius.

Description

This variable determines the value assigned to the `options temp` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       sctemp  string  "31.0"
```

In response, the generated simulation control file contains

```
amsOptions options
+ temp = 31.0
```

sctempeffects

Specifies what built-in primitive components are affected by circuit temperature.

Syntax

`amsDirect.simcntl sctempeffects cyclic "all" | "vt" | "tc"`

Values

`all`	All built-in temperature models are enabled.
`vt`	Only thermal voltage $V_t = \frac{kT}{q}$ can vary with temperature.
`tc`	In addition to thermal voltage, the component temperature coefficient parameters (parameters that start with `tc`, such as `tc1`, and `tc2`) are active. Use this setting when you want to disable the temperature effects for nonlinear devices.

Description

This variable determines the value assigned to the `options tempeffects` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       sctempeffects   cyclic  "vt"
```

In response, the generated simulation control file contains

```
amsOptions options
+ tempeffects = vt
```

sctitle

Specifies a title for the analysis.

Syntax

`amsDirect.simcntl sctitle string "`*`title`*`"`

Values

`title`	The title to be associated with the analysis.

Description

This variable determines the value assigned to the `tran title` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       sctitle string  "tran13"
```

In response, the generated simulation control file contains

```
amsAnalysis tran
+ stop = 0.003
+ title = "tran13"
```

sctnom

Specifies the measurement (nominal) temperature.

Syntax

`amsDirect.simcntl sctnom string "`*`temperature`*`"`

Values

`temperature`	The measurement (nominal) temperature in degrees Celsius.

Description

This variable determines the value assigned to the `tnom` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       sctnom  string  "31.0"
```

In response, the generated simulation control file contains

```
amsOptions options
+ tnom = 31.0

** UltraSim option settings **
*ultrasim: .usim_opt tnom = 31.0
```

sctopcheck

Specifies the extent of the error checking applied to the circuit topology.

Syntax

```
amsDirect.simcntl sctopcheck cyclic "full" | "min" | "no"
```

Values

`full`

`min`

`no`

Description

This variable determines the value assigned to the `options topcheck` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       sctopcheck      cyclic  "min"
```

In response, the generated simulation control file contains

```
amsOptions options
+ topcheck = min
```

scusemodeleval

Specifies whether the `mos_method` option is added to the `options` statement in generated analog simulation control files.

Syntax

`amsDirect.simcntl scusemodeleval boolean t | nil`

Values

`t`	AMS Designer adds the `mos_method` option to the `options` statement. For guidance on setting the value of the `mos_method` option, see "scmodelevaltype" on page 515.
`nil`	The simulator does *not* add the `mos_method` option to the `options` statement. This is the default.

Description

This variable, in conjunction with the `scmodelevaltype` variable, determines whether standard SPICE-like equations or table (accelerated) models are used to evaluate bsim3v3 and bsim4 models.

Example

You set the variables

```
amsDirect.simcntl       scusemodeleval boolean t
amsDirect.simcntl       scmodelevaltype cyclic "a"
```

In response, the generated simulation control file contains

```
amsOptions options
+ mos_method = a
```

scvabstol

Specifies the absolute tolerance for differences in the computed values of the voltages in the last two iterations of a solution.

Syntax

`amsDirect.simcntl scvabstol string "`*`tolerance`*`"`

Values

`tolerance`	The absolute tolerance for differences in the computed values of the voltages.

Description

This variable determines the value assigned to the `options vabstol` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scvabstol       string  "1e-8"
```

In response, the generated simulation control file contains

```
amsOptions options
+ vabstol = 1e-8
```

scwarn

Prints warning messages.

Syntax

`amsDirect.simcntl scwarn boolean t | nil`

Values

`t`	The simulator prints warning messages. This is the default.
`nil`	The simulator does *not* print warning messages.

Description

This variable determines the value assigned to the `options warn` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl       scwarn  boolean nil
```

In response, the generated simulation control file contains

```
amsOptions options
+ warn = no
```

scwrite

Specifies a file to which the simulator writes the initial transient solution.

Syntax

`amsDirect.simcntl scwrite string "`*`file`*`"`

Values

`file`	The path and name of a file to hold the initial transient solution.

Description

This variable determines the value assigned to the `tran write` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl scwrite string "/usr1/tutorial_run/writeinitial"
```

In response, the generated simulation control file contains

```
amsAnalysis tran
+ stop = 0.001
+ write = "/usr1/tutorial_run/writeinitial"
```

scwritefinal

Specifies a file to which the simulator writes the final transient solution.

Syntax

`amsDirect.simcntl scwritefinal string "`*`file`*`"`

Values

`file`	The path and name of a file to hold the final transient solution.

Description

This variable determines the value assigned to the `tran writefinal` option in a generated simulation control file.

Example

You set the variable

```
amsDirect.simcntl scwritefinal string "/usr1/tutorial_run/writefinal"
```

In response, the generated simulation control file contains

```
amsAnalysis tran
+ stop = 0.001
+ writefinal = "/usr1/tutorial_run/writefinal"
```

simcompat

Specifies whether simulation values are to be set for compatibility with Spectre syntax or with HSPICE syntax.

Syntax

`amsDirect.simcntl simcompat cyclic "spectre" | "hspice"`

Values

`spectre`	The simulation values are set for compatibility with Spectre syntax. This is the default.
`hspice`	The simulation values are set for compatibility with HSPICE syntax.

Description

This variable determines whether the `-simcompatible_ams hspice` option or the `-simcompatible_ams spectre` option is added to the `ncsim` command. Notice that the default is always `spectre`, regardless of the solver that you are using.

Example

You set the variable

```
amsDirect.simcntl simcompat cyclic "hspice"
```

In response, the generated `ncsim` command includes

```
ncsim -simcompatible_ams hspice
```

simRunDirLoc

Specifies the default directory to contain run directories.

Syntax

`amsDirect simRunDirLoc string "`*`location`*`"`

Values

`location`	The path and name of the default directory to contain run directories. The *`location`* string can contain shell environment variables. If location contains a relative path, the path is evaluated relative to the directory where the Cadence software (for example, `icms` or `cdsHierEditor`) is started. The default value for *`location`* is an empty string, which means that the current working directory is the default directory to contain run directories.

Description

The AMS Designer environment allows you to designate one or more run directories. You make those designations relative to the directory specified by the `simRunDirLoc` variable.

Example

```
amsDirect simRunDirLoc string "$PROJECT/$BLOCK"
```

If the `simRunDirLoc` variable is set as shown, and the shell variables `$PROJECT` and `$BLOCK` are set to `/newChip` and `comparator`, respectively, the default directory to contain run directories is set to `/newChip/comparator`.

simVisScriptFile

Specifies a script file to be run at the beginning of simulation.

Syntax

```
amsDirect.prep simVisScriptFile string "script_file"
```

Values

`script_file`	The script file to be run at the beginning of simulation. If *`script_file`* uses a relative path, the `ncsim` program looks for the file relative to the run directory. The default is an empty string.

Example

```
amsDirect.prep simVisScriptFile string "demo.tcl"
```

Tells the AMS simulator to run the `demo.tcl` script before starting the simulation.

status

Syntax

`amsDirect.vlog status boolean  t | nil`

Values

`t`	
`nil`	This is the default.

templateFile

Specifies a file whose contents are to be incorporated into the header of newly generated netlists.

Syntax

`amsDirect.vlog templateFile string "`*`text_file`*`"`

Values

`text_file`	Specifies the path and filename of a text file whose contents are to be used in netlist headers. The default is an empty string. The file contents are incorporated into the netlist header only when the `headerText` variable has the value `"file"`. For more information, see “headerText” on page 408.

Example

Specifying the variable

```
amsDirect.vlog templateFile string "./ASICheader"
```

where the file named `ASICheader` contains the following text

```
// Module produced by
// ASIC Team: Ocelot
// San Jose Development Center
```

inserts lines similar to the following at the top of each newly generated netlist.

```
// Verilog-AMS netlist generated by the AMS netlister, version 4.4.6.100.43.
// Cadence Design Systems, Inc.

// Module produced by
// ASIC Team: Ocelot
// San Jose Development Center
```

templateScript

Specifies a file whose contents are a script. The results produced when the script runs are to be incorporated into the header of newly generated netlists.

Syntax

`amsDirect.vlog templateScript string "`*`script_file`*`"`

Values

`script_file`	Specifies the path and filename of a script file. The results produced when the script runs are to be used in netlist headers. The default is an empty string. The results are incorporated into the netlist header only when the `headerText` variable has the value `"script"`. For more information, see "headerText" on page 408.

Example

Specifying the variable

```
amsDirect.vlog templateScript string "./CRheader"
```

where the file named `CRheader` contains the following script

```
echo '// Module produced by:'
echo '// ASIC Interactive, Ltd.'
printf '// (c) '
date '+DATE: %m/%d/%y%n'
```

inserts lines similar to the following at the top of each newly generated netlist.

```
// Verilog-AMS netlist generated by the AMS netlister, version 4.4.6.100.43.
// Cadence Design Systems, Inc.

// Module produced by:
// ASIC Interactive, Ltd.
// (c) DATE: 10/10/01
```

timescale

Specifies the default timescale for Verilog (digital) modules. The `timescale` variable has no effect on analog behavior.

Syntax

`amsDirect.prep timescale string "`*`time_unit`***`/`***`time_precision`***`"`**

Values

time_unit	The units of time to use. The default is `1ns`.
time_precision	The time precision required. The default is `1ns`.

Example

```
amsDirect.prep timescale string "2ns/2ns"
```

Tells the simulator to use `2ns` as the basic unit of time and to calculate time values with a precision of `2ns`.

update

Recompiles the design after design units, source files, or compiler directives are added, or if a design unit is changed in a way that introduces a new cross-file dependency.

Syntax

```
amsDirect.vlog update boolean  t | nil
```

Values

`t`	This is the default.
`nil`	

use5xForVHDL

Controls whether configurations apply to VHDL as well as Verilog-AMS.

Syntax

```
amsDirect.prep use5xForVHDL boolean t | nil
```

Values

`t`	Assumes that configurations apply to VHDL as well as Verilog-AMS. This is the default.
`nil`	Assumes that configurations do not apply to VHDL.

Description

If configurations apply to VHDL, the configurations take precedence over VHDL default binding and other searches. For more information, see the “-USe5x4vhdl Option” section of Chapter 7, in the *Virtuoso AMS Designer Simulator User Guide*.

useDefparam

Controls the netlisting of parameters passed onto instantiated modules.

Syntax

`amsDirect.vlog useDefparam boolean t | nil`

Values

`t`	Generates one `defparam` statement for each instance that requires passed parameters.
`nil`	Passes parameters by assigning instance parameter values. This is the default.

Description

The AMS netlister passes parameters by assigning instance parameter values; it passes parameters by name to child instances. Note digital simulators do not support passing parameters via instance parameter value assignments.

Another way to pass parameters is to use a `defparam` statement. If you specify `t`, the AMS netlister uses the `defparam` statement to pass parameters instead. One `defparam` statement is generated for each instantiation that requires passed parameters.

Example

```
amsDirect.vlog useDefparam boolean nil
```

Tells the AMS netlister to pass parameters by assigning instance parameter values. The following netlist results:

```
module mybuf (a, b);
    input a;
    output b;

    myinv #(.setup(10), .hold(5)) i0 (a, net10);
    myinv #(.setup(10), .hold(5)) i1 (net10, b);
endmodule
```

Example

```
amsDirect.vlog useDefparam boolean t
```

Tells the AMS netlister to pass parameters by using `defparam` statements. The following netlist results:

```
module mybuf (a, b);
    input a;
    output b;

    myinv i0(a, net10);
        defparam i0.setup = 10, i0.hold = 5;
    myinv i1(net10, b);
        defparam i1.setup = 10, i1.hold = 5;

endmodule
```

useEffectiveCDF

Controls whether the netlister uses effective or base CDF values.

The `useEffectiveCDF` variable has an effect only when the `netlistToRunDir` and `useRunDirNetlistsOnly` variables are both set to `t`.

Syntax

`amsDirect useEffectiveCDF boolean t | nil`

Values

`t`	Specifies that effective CDF values are to be used.
`nil`	Specifies that base CDF values are to be used. This is the default.

Example

```
amsDirect netlistToRunDir boolean t
amsDirect useRunDirNetlistsOnly boolean t
amsDirect useEffectiveCDF boolean t
```

All three necessary variables are set to `t` so the netlister uses the effective CDF values, which are the base CDF values updated with user CDF values.

useNcelabNowarn

Controls whether the list of suppressed warnings on the *Elaborator Messages/Errors* pane of the AMS Options window is used.

Syntax

`amsDirect.prep useNcelabNowarn boolean t | nil`

Values

`t`	Places a checkmark next to the *Suppress specific warnings* field, indicating that the listed warnings are to be suppressed. This is the default.
`nil`	Removes the checkmark, indicating that any listed warnings are not to be suppressed.

Example

```
amsDirect.prep useNcelabNowarn boolean nil
```

Removes the checkmark next to the *Suppress specific warnings* field. As a result, the `-nowarn` option of the `ncelab` command is not used and the listed warnings are not suppressed during elaboration.

useNcelabSdfCmdFile

Controls whether an SDF command file specified on the *SDF Annotation* pane of the AMS Options window is used.

Syntax

```
amsDirect.prep useNcelabSdfCmdFile boolean t | nil
```

Values

`t`	Places a checkmark next to the *Use SDF command file* field, indicating that the command file (if one is specified) is to be used during elaboration. This is the default.
`nil`	Removes the checkmark, indicating that any SDF command file that might be specified in the *Use SDF command file* field is not to be used.

Example

```
amsDirect.prep useNcelabSdfCmdFile boolean nil
```

Tells the elaborator to use an SDF command file if one is specified in the *Use SDF command file* field.

useNcsimNowarn

Controls whether the list of suppressed warnings on the *Simulator Messages/Errors* pane of the AMS Options window is used.

Syntax

```
amsDirect.prep useNcsimNowarn boolean t | nil
```

Values

`t`	Places a checkmark next to the *Suppress specific warnings* field, indicating that the listed simulation warnings are to be suppressed. This is the default.
`nil`	Removes the checkmark, indicating that any listed warnings are not to be suppressed.

Example

```
amsDirect.prep useNcsimNowarn boolean nil
```

Removes the checkmark next to the *Suppress specific warnings* field. As a result, the `-nowarn` option of the `ncsim` command is not used and the listed warnings are not suppressed during simulation.

useNowarn

Controls whether the list of suppressed warnings on the *Verilog-AMS Messages/Errors* pane of the AMS Options window is used.

Syntax

`amsDirect.vlog useNowarn boolean t | nil`

`amsDirect.vhdl useNowarn boolean t | nil`

Values

`t`	Places a checkmark next to the *Suppress specific warnings* field, indicating that the listed compilation warnings are to be suppressed. This is the default.
`nil`	Removes the checkmark, indicating that any listed warnings are not to be suppressed.

Example

```
amsDirect.vlog useNowarn boolean nil
```

Removes the checkmark next to the *Suppress specific warnings* field. As a result, the `-nowarn` option of the `ncvlog` command is not used and the listed warnings are not suppressed during compilation.

useProcessViewNamesOnly

Controls how the AMS netlister determines which cellviews to process.

Syntax

`amsDirect.vlog useProcessViewNamesOnly boolean t | nil`

Values

`t`	The AMS netlister determines which cellviews to process by consulting the `processViewNames` list.
`nil`	The AMS netlister determines which cellviews to process by consulting, in combination, the `amsEligibleViewTypes` list and the `excludeViewNames` list. This is the default.

Description

This variable determines which of two methods is used to select the views that are processed by the AMS netlister.

Example

Given the following values:

```
amsDirect.vlog amsEligibleViewTypes        string "schematic"
amsDirect.vlog excludeViewNames            string "sch[0-3]"
amsDirect.vlog processViewNames            string "sch1 sch[3-4]"
amsDirect.vlog useProcessViewNamesOnly     boolean nil
```

If the AMS netlister runs, for example, in response to a CDF save trigger on cell `mycell`, which has the six schematic views `sch0`, `sch1`, `sch2`, `sch3`, `sch4,` and `sch5`, only the `sch4` and `sch5` views are processed.

On the other hand, if `useProcessViewNamesOnly` is set to `t`, only the `sch1`, `sch3`, and `sch4` views are processed.

useScaddlglblopts

Controls whether the list of additional options on the *Analog Solver* pane of the AMS Options window is used.

Syntax

```
amsDirect.simcntl useScaddlglblopts boolean t | nil
```

Values

`t`	Places a checkmark next to the *Additional options* field, indicating that the listed options are to be passed to the analog solver. This is the default.
`nil`	Removes the checkmark, indicating that any listed additional options are to be ignored.

Example

```
amsDirect.simcntl useScaddlglblopts boolean nil
```

Removes the checkmark next to the *Additional options* field. As a result, any options listed in the field are not used by the analog solver.

useScaddltranopts

Controls whether the list of additional options on the *Tran Analysis* pane of the AMS Options window is used.

Syntax

```
amsDirect.simcntl useScaddltranopts boolean t | nil
```

Values

`t`	Places a checkmark next to the *Additional options* field, indicating that the listed options are to be used for transient analysis. This is the default.
`nil`	Removes the checkmark, indicating that any listed additional options are to be ignored

Example

```
amsDirect.simcntl useScaddltranopts boolean nil
```

Removes the checkmark next to the *Additional options* field. As a result, any options listed in the field are not used during transient analysis.

useScic

Controls whether the list of initial conditions on the *Tran Convergence/Accuracy* pane of the AMS Options window is used.

Syntax

```
amsDirect.simcntl useScic boolean t | nil
```

Values

`t`	Places a checkmark next to the *Set initial conditions* field, indicating that the listed initial conditions are to be used for transient analysis. This is the default.
`nil`	Removes the checkmark, indicating that any listed initial conditions are to be ignored.

Example

```
amsDirect.simcntl useScic boolean nil
```

Removes the checkmark next to the *Set initial conditions* field. As a result, any conditions listed in the field are not used during transient analysis.

useScreadic

Controls whether initial conditions are read from the file specified on the *Tran Convergence/ Accuracy* pane of the AMS Options window.

Syntax

```
amsDirect.simcntl useScreadic boolean t | nil
```

Values

`t`	Places a checkmark next to the *Read IC from file* field, indicating that initial conditions are to be read from the specified file. This is the default.
`nil`	Removes the checkmark, indicating that initial conditions are not to be read from the specified file.

Example

```
amsDirect.simcntl useScreadic boolean nil
```

Removes the checkmark next to the *Read IC from file* field. As a result, initial conditions that might be specified in the file are not used.

useScreadns

Controls whether nodesets are read from the file specified on the *Tran Convergence/ Accuracy* pane of the AMS Options window.

Syntax

`amsDirect.simcntl useScreadns boolean t | nil`

Values

`t`	Places a checkmark next to the *Read nodesets from file* field, indicating that nodesets s are to be read from the specified file. This is the default.
`nil`	Removes the checkmark, indicating that nodesets are not to be read from the specified file.

Example

```
amsDirect.simcntl useScreadns boolean nil
```

Removes the checkmark next to the *Read nodesets from file* field. As a result, nodesets that might be specified in the file are not used.

useScscfincfile

Controls whether a simulation control file specified on the *Analog Solver* pane of the AMS Options window is included when the analog simulation control file runs.

Syntax

`amsDirect.simcntl useScscfincfile boolean t | nil`

Values

`t`	Places a checkmark next to the *Include simulation control file* field, indicating that the file is to be included when the analog simulation control file runs. This is the default.
`nil`	Removes the checkmark, indicating that the specified simulation control file is not to be included when the analog simulation control file runs.

Example

```
amsDirect.simcntl useScscfincfile boolean nil
```

Removes the checkmark next to the *Include simulation control file* field. As a result, any simulation control file that might be specified in the field is not included when the analog simulation control file runs.

useScwrite

Controls whether the initial solution is written to the file specified on the *Tran Output* pane of the AMS Options window.

Syntax

```
amsDirect.simcntl useScwrite boolean t | nil
```

Values

`t`	Places a checkmark next to the *Write initial solution to file* field, indicating that the initial solution is to be written to the specified file. This is the default.
`nil`	Removes the checkmark, indicating that the initial solution is not to be written to the specified file.

Example

```
amsDirect.simcntl useScwrite boolean nil
```

Removes the checkmark next to the *Write initial solution to file* field. As a result, the initial solution is not written to the file.

useScwritefinal

Controls whether the final solution is written to the file specified on the *Tran Output* pane of the AMS Options window.

Syntax

`amsDirect.simcntl useScwritefinal boolean t | nil`

Values

`t`	Places a checkmark next to the *Write final solution to file* field, indicating that the final solution is to be written to the specified file. This is the default.
`nil`	Removes the checkmark, indicating that final solution is not to be written to the specified file.

Example

```
amsDirect.simcntl useScwritefinal boolean nil
```

Removes the checkmark next to the *Write final solution to file* field. As a result, the final solution is not written to the file.

useSimVisScriptFile

Controls whether a Tcl input script specified on the *Simulator* pane of the AMS Options window is used.

Syntax

```
amsDirect.prep useSimVisScriptFile boolean t | nil
```

Values

`t`	Places a checkmark next to the *Tcl input script* field, indicating that the script (if one is specified) is to be used to control the simulator. This is the default.
`nil`	Removes the checkmark, indicating that any script that might be specified in the *Tcl input script* field is not to be used.

Example

```
amsDirect.prep useSimVisScriptFile boolean nil
```

Tells the simulator not to ignore any script that might be specified in the *Tcl input script* field.

usimAbstoli through usimWFTres

These `ams.env` `usim*` variables correspond to UltraSim circuit simulator command options as follows:

ams.env Variable	UltraSim Option
`usimAbstoli`	`.usim_opt abstoli`
`usimAbstolv`	`.usim_opt abstolv`
`usimAddlOptions`	`.usim_opt` followed by the specified options.
`usimAnalog`	`.usim_opt analog`
`usimCapFile`	`.usim_opt capfile`
`usimCgnd`	`.usim _opt cgnd`
`usimCgndr`	`.usim_opt cgndr`
`usimDCMethod`	`.usim_opt dc`
`usimDcut`	`.usim_opt dcut`
`usimDcutField`	`.usim_opt dcut`
`usimDiodeMethod`	`.usim_opt diode_method`
`usimDpfFile`	`.usim_opt dpf`
`usimDumpStep`	`.usim_opt dump_step`
`usimenableNA`	None. Allows UltraSim Advanced Checks Node Activity Analysis information to be added to the control file.
`usimenablePA`	None. Allows UltraSim Advanced Checks Power Analysis information to be added to the control file.
`usimenableRA`	None. Allows UltraSim Advanced Checks Reliability Analysis information to be added to the control file.
`usimenableTA`	None. Allows UltraSim Advanced Checks Timing Analysis information to be added to the control file.
`usimLshort`	`.usim_opt lshort`
`usimLvshort`	`.usim_opt lvshort`
`usimMaxstep`	`.usim_opt maxstep`
`usimMaxstepStart`	`.usim_opt maxstep_start`

ams.env Variable	UltraSim Option
`usimMaxstepStop`	`.usim_opt maxstep_stop`
`usimMaxstepSubckt`	`.usim_opt maxstep`
`usimMosMethod`	`.usim_opt mos_method`
`usimNALimit`	`.usim_nact limit`
`usimNAOutputSort`	`.usim_nact max_vo`
`usimNASortIs`	`.usim_nact`
`usimOutputStart`	`tran outputstart`
`usimPostl`	`.usim_opt postl`
`usimRAAgeDomain`	`.agemethod`
`usimRAAgeMethod`	`.agemethod`
`usimRAAgeproc`	`.ageproc`
`usimRAAgingTime`	`.age`
`usimRADeltaD`	`.deltad`
`usimRADeltaDToggle`	`.deltad`
`usimRAMinAge`	`.minage`
`usimRAMode`	`.hci_only`, `.nbti_only`, `.nbtiageproc`
`usimRANBTIAgeproc`	`.nbtiageproc`
`usimRcrfmax`	`.usim_opt rcr_fmax`
`usimRshort`	`.usim_opt Rshort`
`usimRvshort`	`.usim_opt Rvshort`
`usimSimMode`	`.usim_opt sim_mode`
`usimSpeed`	`.usim_opt speed`
`usimSpefFile`	`.usim_opt spef`
`usimSpfFile`	`.usim_opt spf`
`usimTol`	`.usim_opt tol`
`usimTranAddlOptions`	`tran` followed by the specified options.
`usimUseAddlOptions`	None. Allows additional UltraSim options to be used.

ams.env Variable	**UltraSim Option**
`usimVcdFile`	`.vcd`
`usimVcdInfoFile`	`.vcd`
`usimVectorFile`	`.vec`
`usimWFAbstoli`	`.usim_opt wf_abstoli`
`usimWFAbstolv`	`.usim_opt wf_abstolv`
`usimWFFilter`	`.usim_opt wf_filter`
`usimWFReltol`	`.usim_opt wf_reltol`
`usimWFTres`	`.usim_opt wf_tres`

For information about UltraSim circuit simulator command options, see the *Virtuoso UltraSim Simulator User Guide*.

verboseUpdate

Controls whether the names of already up-to-date modules are included in the log file generated for an update compilation.

Syntax

```
amsDirect.vlog verboseUpdate boolean t | nil
```

Values

`t`	Places a checkmark next to the *Print verbose messages during update* field on the *Verilog-AMS* pane of the AMS Options window. This tells the compiler to print the names of already up-to-date cells in the log, while updating cells. This is the default.
`nil`	Removes the checkmark, indicating that the names of up-to-date cells are not to be printed in the log, while updating cells.

Example

```
amsDirect.vlog verboseUpdate boolean t
```

Example

```
amsDirect.simcntl useScaddltranopts boolean nil
```

Removes the checkmark next to the *Print verbose messages during update* field. As a result, the names of up-to-date cells do not appear in the log.

vlogGroundSigs

Specifies which signals are to be declared as ground.

Syntax

`amsDirect.prep vlogGroundSigs string "`*`signal_list`*`"`

Values

`signal_list`	A list of signals to be declared, by default, as ground. The default is `gnd!`.

Description

AMS Designer uses the value of this variable to determine which wires should be declared as ground.

Example

For example, if the variable is defined like

```
amsDirect.prep vlogGroundSigs string "gnd! gnd2!"
```

then AMS Designer declares any new global signals named `gnd!` and `gnd2!` to be a ground.

vloglinedebug

Enables support for setting line breakpoints and for single-stepping through code.

Syntax

`amsDirect.vlog vloglinedebug boolean  t | nil`

Values

`t`	
`nil`	This is the default.

Description

Example

```
amsDirect.vlog vloglinedubug boolean t
```

Tells AMS Designer to compile Verilog files with the `-linedebug` option. As a result, the generated command might look like this.

```
ncvlog -linedebug
```

vlogSupply0Sigs

Specifies which signals are to be declared as supply0 wire types.

Syntax

`amsDirect.prep vlogSupply0Sigs string "`*`signal_list`*`"`

Values

`signal_list`	A list of signals to be declared, by default, as supply0 wires. The default is an empty string.

Description

AMS Designer uses the value of this variable to determine which wires should be declared as supply0 wire types.

Example

For example, if the variable is defined like

```
amsDirect.prep vlogSupply0Sigs string "vss! vss2!"
```

then AMS Designer declares any new global signals named `vss!` and `vss2!` to be a supply0 wire type.

vlogSupply1Sigs

Specifies which signals are to be declared as supply1 wire types.

Syntax

```
amsDirect.prep vlogSupply1Sigs string "signal_list"
```

Values

signal_list	A list of signals to be declared, by default, as supply1 wires. The default is an empty string.

Description

AMS Designer uses the value of this variable to determine which wires should be declared as supply1 wire types.

Example

For example, if the variable is defined like

```
amsDirect.prep vlogSupply1Sigs string "vdd! vdd2!"
```

then AMS Designer declares any new global signal named `vdd!` and `vdd2!` to be a supply1 wire type.

wfDefaultDatabase

Specifies the name of the default database for waveform data produced by the simulator. This name appears in the *Default database name* field, on the *Waveforms* pane of the AMS Options window. It also appears as a database in the AMS Databases window and is used as the default

Syntax

`amsDirect.prep wfDefaultDatabase string "`*`database`*`"`

Values

`database`	The name of the default database for waveform data. The default name is `waves`.

Example

```
amsDirect.prep wfDefaultDatabase string "fast_db"
```

Specifies that the `fast_db` database is to be the default database for waveform data produced by the simulator. This name `fast_db` appears in the AMS Databases window and is the default database for new selections in the AMS Save/Plot window.

wfDefInstCSaveAll

Specifies whether current probes are to be created for all levels of the instances selected from the schematic or the navigator.

Syntax

```
amsDirect.prep wfDefInstCSaveAll boolean t | nil
```

Values

`t`	Specifies that current probes are to be created for all levels of the selected instances. This choice is indicated in the AMS Save/Plot list by the word `all` appearing in the *Depth* column for the selected instances.
`nil`	Specifies that current probes for the selected instances are to be created only for the number of levels specified by the `amsDirect.prep wfDefInstCSaveLvl` variable. This is the default.

Example

```
amsDirect.prep wfDefInstCSaveAll boolean t
```

Specifies that current probes are to be created for all levels of the selected instances.

wfDefInstCSaveLvl

Specifies that current probes for the specified number of levels are to be created for instances selected from the schematic or the navigator.

Syntax

`amsDirect.prep wfDefInstCSaveLvl int` *`level`*

Values

`level`	The number of levels of current probes to be created for the selected instances. The default value is 1.

Example

```
amsDirect.prep wfDefInstCSaveLvl int 2
```

Specifies that current probes are to be created for two levels of each of the selected instances. In the AMS Save/Plot list, the *Depth* column for the selected instances contains the value `2`.

wfDefInstSaveCurrents

Controls whether current probes are created for the objects selected from the schematic or the navigator.

Syntax

```
amsDirect.prep wfDefInstSaveCurrents boolean t | nil
```

Values

`t`	Places a checkmark next to the *Currents at terminals or ports* label in the *Waveforms* pane of the AMS Options window, indicating that current probes are to be created for the objects selected from the schematic or the navigator.
`nil`	Removes the checkmark next to the *Currents at terminals or ports* label in the *Waveforms* pane of the AMS Options window, indicating that current probes are not to be created. This is the default.

Example

```
amsDirect.prep wfDefInstSaveCurrents boolean t
```

Places a checkmark next to the *Currents at terminals or port* labels, indicating that current probes are to be created for objects selected from the schematic or the navigator.

wfDefInstSaveVoltages

Controls whether voltage probes are created for instances selected from the schematic or the navigator.

Syntax

`amsDirect.prep wfDefInstSaveVoltages boolean t | nil`

Values

`t`	Places a checkmark next to the *Voltages/Signals* label in the *Waveforms* pane of the AMS Options window, indicating that voltage probes are to be created for instances selected from the schematic or navigator. This is the default.
`nil`	Removes the checkmark next to the *Voltages/Signals* field in the *Waveforms* pane of the AMS Options window, indicating that voltages are not to be created for instances selected from the schematic or navigator.

Example

```
amsDirect.prep wfDefInstSaveVoltages boolean nil
```

Removes the checkmark next to the *Voltages/Signals* label so voltage probes are not created for objects selected from the schematic or navigator.

wfDefInstVSaveAll

Specifies whether voltage probes are to be created for all levels of the instances selected from the schematic or the navigator.

Syntax

`amsDirect.prep wfDefInstVSaveAll t | nil`

Values

`t`	Specifies that voltage probes are to be created for all levels of the selected instances. This choice is indicated in the AMS Save/ Plot list by the word `all` appearing in the *Depth* column for the selected instances
`nil`	Specifies that voltage probes for the selected instances are to be created only for the number of levels specified by the `amsDirect.prep wfDefInstVSaveLvl` variable. This is the default.

Example

```
amsDirect.prep wfDefInstVSaveAll nil
```

Specifies that voltage probes are to be created for all levels of the selected instances.

wfDefInstVSaveLvl

Specifies that voltage probes for the specified number of levels are to be created for instances selected from the schematic or the navigator.

Syntax

`amsDirect.prep wfDefInstVSaveLvl int` *`level`*

Values

`level`	The number of levels of voltage probes to be created for the selected instances. The default value is 1.

Example

```
amsDirect.prep wfDefInstVSaveLvl int 2
```

Specifies that voltage probes are to be created for two levels of each of the selected instances. In the AMS Save/Plot list, the *Depth* column for the selected instances contains the value `2`.

wfDefInstVSaveObjects

Specifies the objects for which voltages are to be saved when instances are selected from the schematic or navigator.

Syntax

```
amsDirect.prep wfDefInstVSaveObjects cyclic "Input_ports" | "Output_ports"
     | "All_ports" | "All_data"
```

Values

`Input_ports`	Indicates that only the input ports of the selected instances are to be probed for voltages.
`Output_ports`	Indicates that only the output ports of the selected instances are to be probed for voltages.
`All_ports`	Indicates that both the input and the output ports of the selected instances are to be probed for voltages.
`All_data`	Indicates that all ports and internal signals of the selected instances are to be probed for voltages. This is the default.

Example

```
amsDirect.prep wfDefInstVSaveObjects cyclic "Output_ports"
```

Specifies that only output ports are to be probed for voltages when instances are selected. Consequently, the row that appears in the AMS Save/Plot window when an instance is selected contains the *Output Ports* icon in the *Object* column. The Tcl `probe` command created from the row includes the `-outputs` option.

wfFilter

Controls whether domain filters are applied when probe data are saved.

Filtering by domain is not allowed when you probe currents. Filtering by domain has no effect on probes of nets, signals, or terminals.

Syntax

`amsDirect.prep wfFilter boolean t | nil`

Values

`t`	Specifies that the domain filters specified by the `wfFilterSpec` variable are to be applied.
`nil`	Specifies that the domain filters specified by the `wfFilterSpec` variable are not to be applied. This is the default.

Example

```
amsDirect.prep wfFilter boolean t
```

Specifies that the domain filters specified by the `wfFilterSpec` variable are to be applied when probe data are saved. In the *Waveforms* pane of the AMS Options window, the *Filtered by domain* field is check marked.

wfFilterSpec

Specifies the domains for which data are to be saved while probing.

Syntax

```
amsDirect.prep wfFilterSpec cyclic "none" | "digital" | "analog"
```

Values

`none`	Specifies that no filtering is to occur, which means that probe results are saved from both the digital and analog domains. This is the default.
`digital`	Saves probe results from only the digital domain.
`analog`	Saves probe results from only the analog domain.

Example

```
amsDirect.prep wfFilterSpec cyclic "digital"
```

Specifies that only probe results from the digital domain are to be saved. In the Waveforms pane of the AMS Options window, the *Digital* field is selected.

B

CIW Interface for AMS Designer

You can perform the following tasks using the command interpreter window (CIW) interface for Virtuoso® AMS Designer:

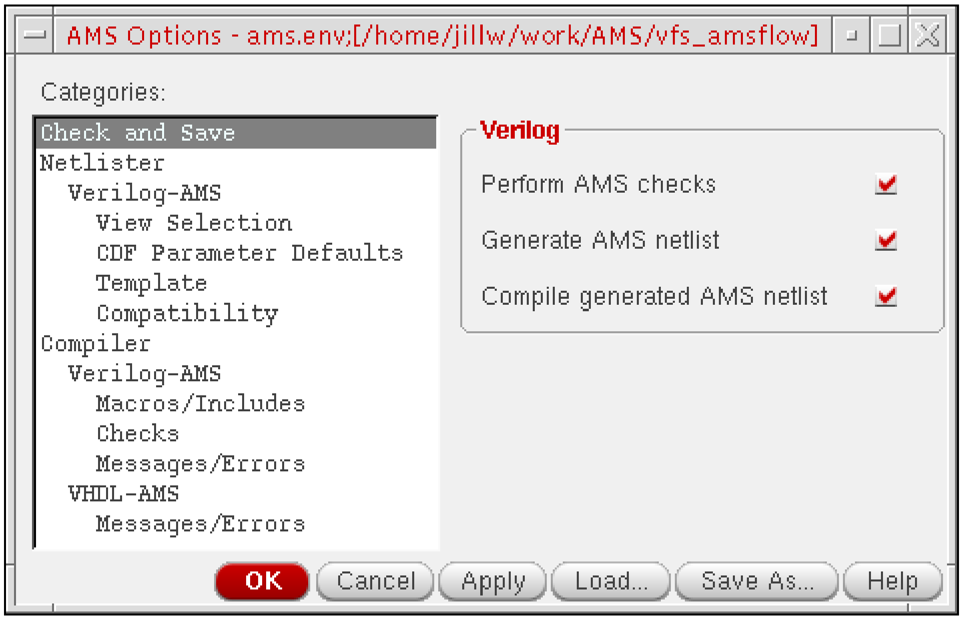
Specifying Automatic Netlisting from the CIW

You can set up AMS Designer for automatic cellview-based netlisting so that netlisting occurs automatically and transparently whenever you save a cellview that has valid connectivity. You can specify that you want both netlisting and compiling to take place automatically so that the AMS Designer simulator always has the required information and can run quickly.

To specify automatic netlisting, do the following:

1. In the CIW, choose *Tools – AMS – Options*.

   The AMS Options form appears.



AMS Options - ams.env;[/home/jillw/work/AMS/vfs_amsflow]

Categories:

```
Check and Save
Netlister
  Verilog-AMS
    View Selection
    CDF Parameter Defaults
    Template
    Compatibility
Compiler
  Verilog-AMS
    Macros/Includes
    Checks
    Messages/Errors
  VHDL-AMS
    Messages/Errors
```

Verilog

Perform AMS checks ✓

Generate AMS netlist ✓

Compile generated AMS netlist ✓

2. In the *Categories* list area, select *Check and Save*.
3. In the *Verilog* group box, select automated actions for the *Check and Save* operation:
   - ❑ *Perform AMS checks*
   - ❑ *Generate AMS netlist*

     Note: If you do not turn on *Generate AMS netlist*, when you check-and-save a cellview, the AMS netlister removes any previously-created netlist for the cellview, whether you have enabled AMS or not. This process of removing existing netlists ensures that you do not inadvertently simulate an out-of-date netlist.
   - ❑ *Compile generated AMS netlist*

   When you check-and-save a cellview, the software performs the tasks you select.

4. If necessary, select other items in the *Categories* list area and set the options that control the AMS netlister. For more information, see "Specifying AMS Netlister Options from the CIW" on page 607.
5. Click *OK* to save your settings and close the form.

Now, you can use the Virtuoso® Schematic Editor to create or edit schematic views, then click *Check and Save* to run the AMS netlister automatically. The netlister creates a Verilog-AMS netlist in the cellview directory of your saved schematic view. This netlist is available to all users of the block: None of the users needs to recreate the netlist unless the block changes.

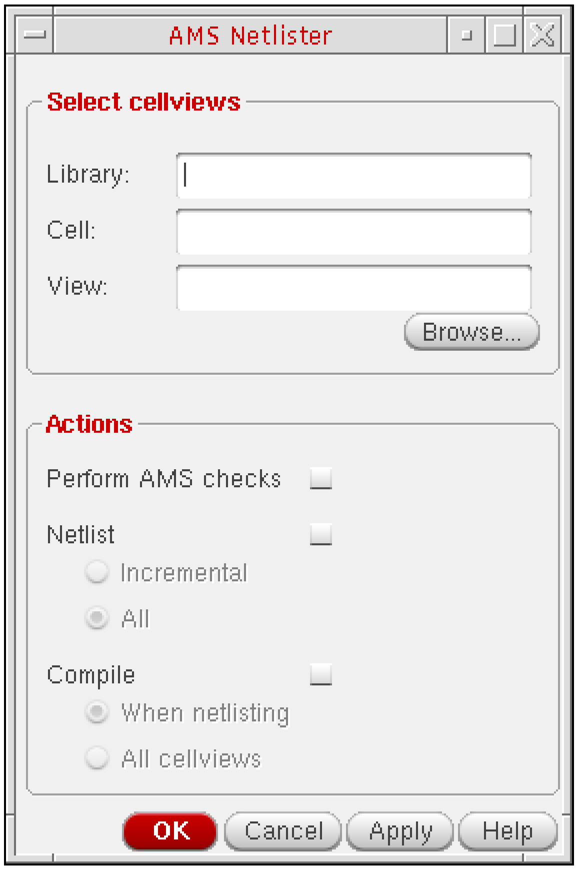
Library Netlisting from the CIW

Using this method, you can check, netlist, and compile an entire library, all the views of a cell, or a single cellview from the command interpreter window (CIW). You can also netlist and compile only new or revised cellviews.

To use library netlisting from the CIW, do the following:

1. In the CIW, choose *Tools – AMS – Netlist*.

The AMS Netlister form appears.


AMS Netlister

Select cellviews

Library:

Cell:

View:

Browse...

Actions

Perform AMS checks ☐

Netlist ☐

Incremental

[✓ All]

Compile ☐

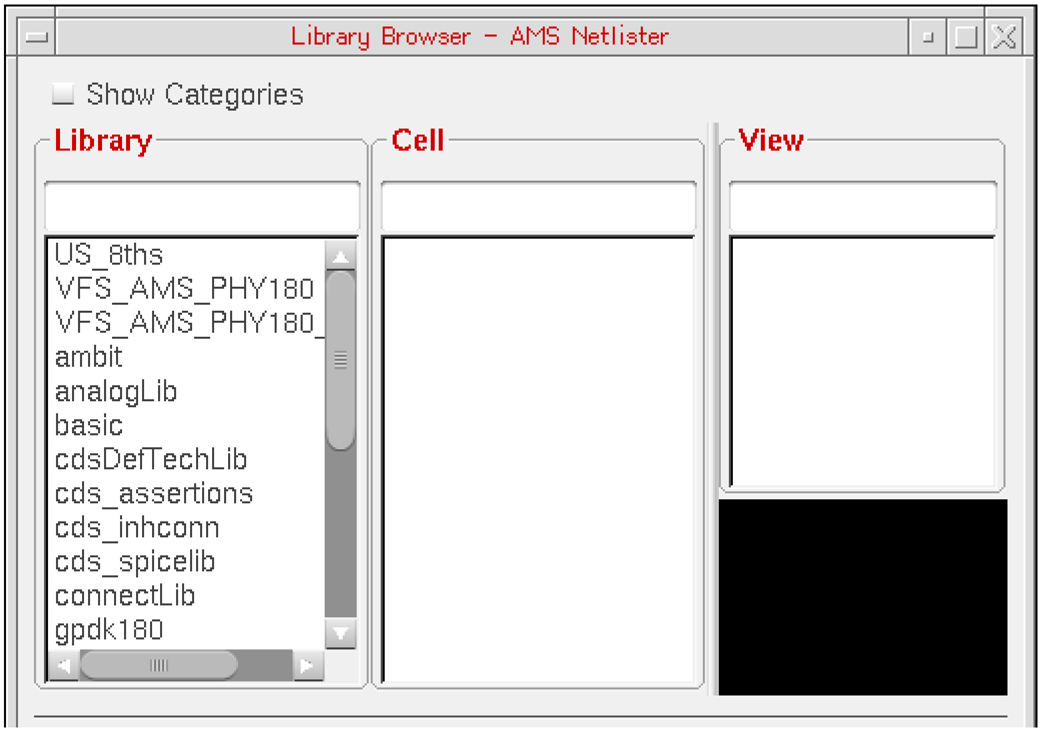
[✓ When netlisting]

All cellviews

OK Cancel Apply Help

2. Click *Browse*.

The Library Browser – AMS Netlister form appears.



3. In the *Library* column, select the library containing the cellviews you want to netlist.

   The library name appears in the *Library* field on the AMS Netlister form.

4. (Optional) In the *Cell* column, select a cell.

   The cell name appears in the *Cell* field on the AMS Netlister form.

   If you do not select a cell, the AMS netlister operates on eligible views for every cell in the library.

5. (Optional) In the *View* column, select a view.

   The view name appears in the *View* field on the AMS Netlister form.

   Note: You can further specify view names to process or to exclude. For more information, see "Eligible View Types and View Names to Exclude" on page 619 and "View Names to Process" on page 621.

6. In the *Actions* group box on the AMS Netlister form, select what you want the AMS netlister to do with the specified cellviews in the current run. You can choose incremental netlisting if you want to netlist only new or changed cellviews. Any selections you make on this form are for this run only and do not affect the settings on the AMS Options form (for automatic netlisting).

7. Click *OK* to begin the run.

   The AMS netlister creates Verilog-AMS netlists according to your specifications.

Specifying AMS Netlister Options from the CIW

You can specify the following AMS cellview-based netlister options from the CIW:

- Maximum Number of Errors on page 608
- Print Informational Messages on page 609
- Use Scaling Notation for Parameter Values on page 610
- Include Files on page 611
- Header Text on page 613
- Conditionally Include Verilog-AMS Language Extentions on page 617
- Eligible View Types and View Names to Exclude on page 619
- View Names to Process on page 621
- CDF Parameter Defaults on page 623
- Verilog-AMS Compatibility Exceptions on page 625

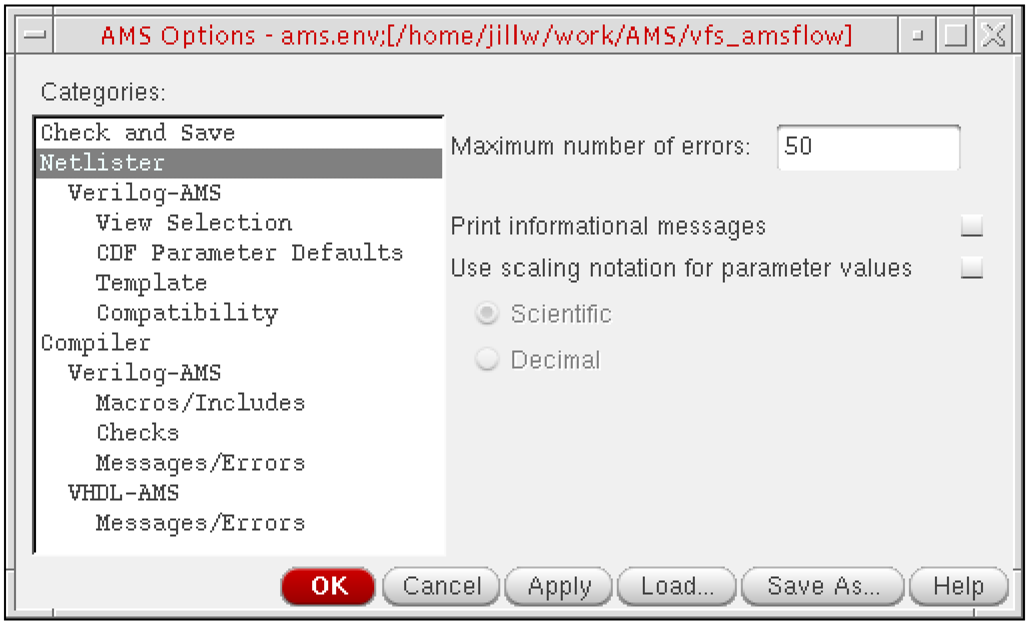
Maximum Number of Errors

To specify the maximum number of errors the AMS netlister can encounter before it stops processing the design, do the following:

1. In the command interpreter window (CIW), choose *Tools – AMS – Options*.

   The AMS Options form appears.

2. In the *Categories* list, select *Netlister*.



AMS Options - ams.env:[/home/jillw/work/AMS/vfs_amsflow]

Categories:

```
Check and Save
Netlister
  Verilog-AMS
    View Selection
    CDF Parameter Defaults
    Template
    Compatibility
Compiler
  Verilog-AMS
    Macros/Includes
    Checks
    Messages/Errors
  VHDL-AMS
    Messages/Errors
```

Maximum number of errors: 50

Print informational messages ☐

Use scaling notation for parameter values ☐

● Scientific

○ Decimal

3. In the *Maximum number of errors* field, type the maximum number of errors the AMS netlister can encounter before it stops processing the design.

4. Click *OK*.

   If the AMS netlister encounters more errors than the number you specified in the *Maximum number of errors* field, it stops processing the design.

   If the AMS netlister encounters any errors, it does not generate a netlist and it removes any existing netlist so that you cannot inadvertently simulate an out-of-date netlist.

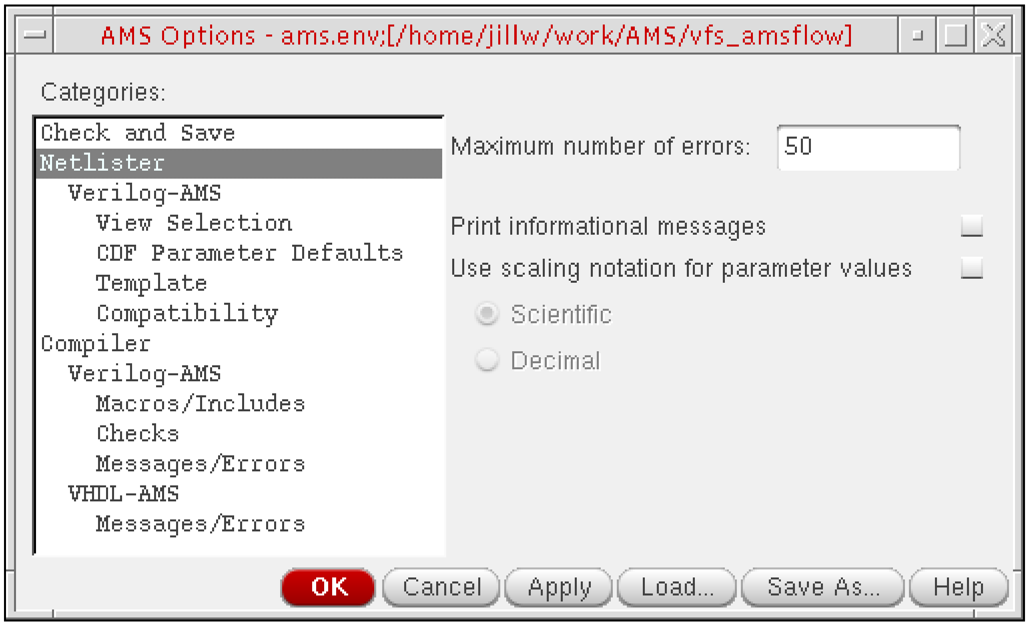
Print Informational Messages

To specify that you want the AMS netlister to print informational messages, do the following:

1. In the command interpreter window (CIW), choose *Tools – AMS – Options*.

   The AMS Options form appears.

2. In the *Categories* list, select *Netlister*.



AMS Options - ams.env:[/home/jillw/work/AMS/vfs_amsflow]

Categories:

```
Check and Save
Netlister
  Verilog-AMS
    View Selection
    CDF Parameter Defaults
    Template
    Compatibility
Compiler
  Verilog-AMS
    Macros/Includes
    Checks
    Messages/Errors
  VHDL-AMS
    Messages/Errors
```

Maximum number of errors: 50

Print informational messages ☐

Use scaling notation for parameter values ☐

[● Scientific] Decimal

3. Turn on *Print informational messages*.

4. Click *OK*.

   The AMS netlister will print more numerous and more extensive informational messages which can help you if you are trying to debug a netlisting problem.

Use Scaling Notation for Parameter Values

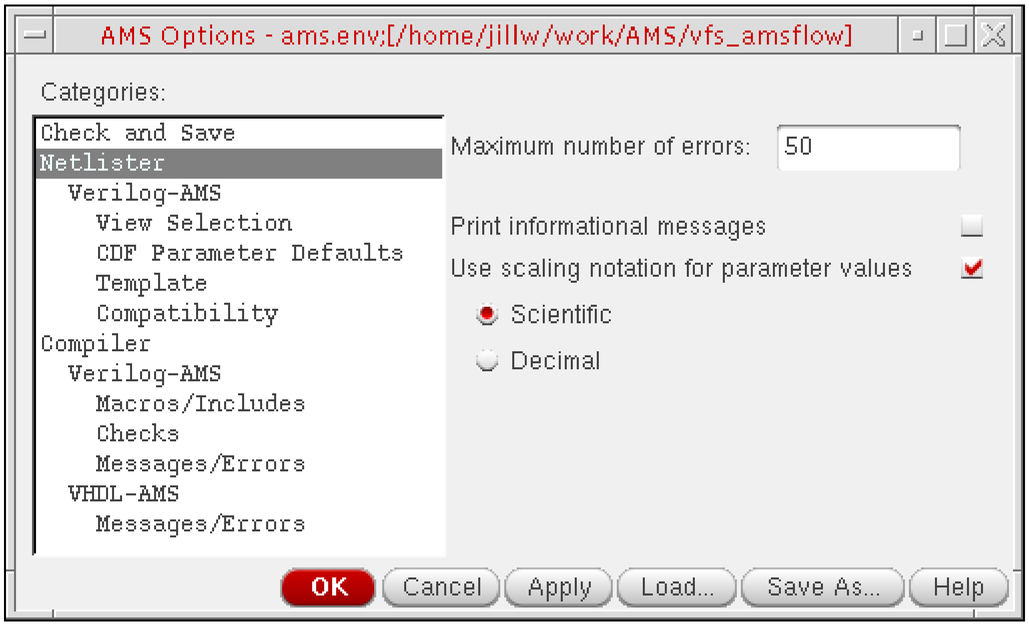
To specify that you want the AMS netlister to use scaling notation for parameter values, do the following:

1. In the command interpreter window (CIW), choose *Tools – AMS – Options*.

   The AMS Options form appears.

2. In the *Categories* list, select *Netlister*.

3. Turn on *Use scaling notation for parameter values*.



Maximum number of errors: 50
Print informational messages ☐
Use scaling notation for parameter values ✓
● Scientific
○ Decimal

   The *Scientific* and *Decimal* radio buttons become active.

4. Select one of the following radio buttons:

   ❑ *Scientific* expands scaling factor suffixes in scientific notation.

   For example, `5.46M` becomes `5.46e6`.

   ❑ *Decimal* expands scaling factor suffixes in decimal notation.

   For example, `5.46M` becomes `5,460,000`.

5. Click *OK*.

   The AMS netlister expands scaling factor suffixes according to your specifications.

Include Files

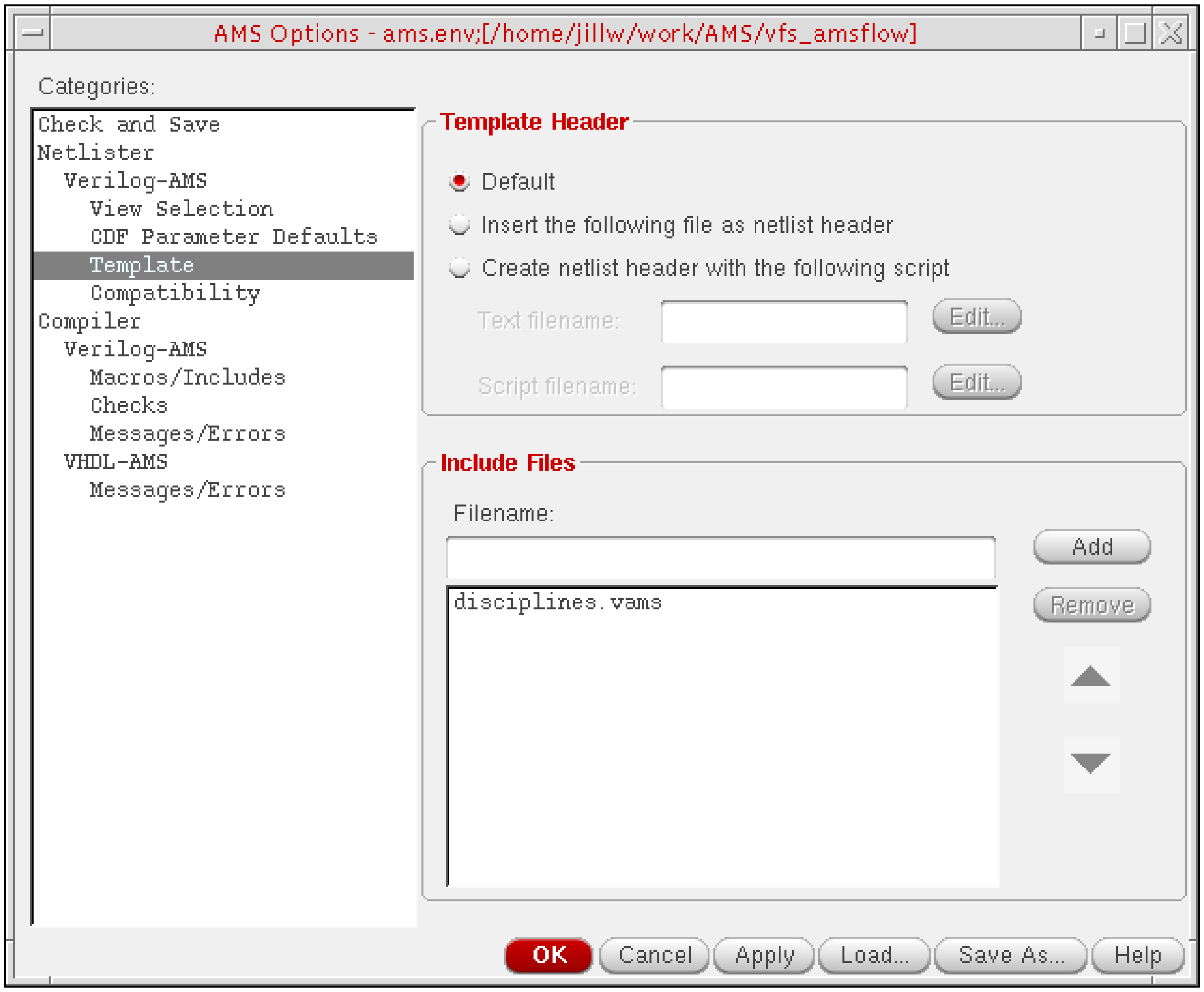
To specify files you want the AMS netlister to include in the Verilog-AMS netlist, do the following:

1. In the command interpreter window (CIW), choose *Tools – AMS – Options*.

   The AMS Options form appears.

2. In the *Categories* list, select *Netlister – Verilog-AMS – Template*.

   The *Template Header* and *Include Files* group boxes appear.


AMS Options - ams.env;[/home/jillw/work/AMS/vfs_amsflow]

Categories:

```
Check and Save
Netlister
  Verilog-AMS
    View Selection
    CDF Parameter Defaults
    Template
    Compatibility
Compiler
  Verilog-AMS
    Macros/Includes
    Checks
    Messages/Errors
  VHDL-AMS
    Messages/Errors
```

Template Header

[✓ Default]
Insert the following file as netlist header
Create netlist header with the following script
Text filename:
Script filename:

Include Files

Filename:

```
disciplines.vams
```

3. For each file you want to include in the netlist, do the following in the *Include Files* group box:

   a. In the *Filename* field, type the name of the file.

b. Click *Add*.

c. (Optional) Use the up and down arrows to move a file name up or down in the list.

The AMS netlister writes include files to the netlist in the order that they appear in this list. The order is important if you have files that use declarations in another file. For example, if `File2` uses a declaration from `File1`, `File1` must appear above `File2` in the list.

The file name you typed appears in the list area beneath the *Filename* field.

The AMS netlister writes a `` `include `` directive to every netlist to include each file you specify here. To specify the directories the AMS netlister searches for these files, see "Directories to Search for Verilog-AMS Include Files" on page 636.

Note: You can remove a file from the list by selecting it and clicking *Remove*.

Header Text

You can specify header text that you want the AMS netlister to insert in every Verilog-AMS netlist it generates using one of the following choices:

- Include Header Text from a Particular File on page 613
- Include Header Text That Results from a Script File on page 615

In either case, the AMS netlister inserts the header text you specify after the default header text, which is as follows:

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

Include Header Text from a Particular File

To include header text from a particular file, do the following:

1. In the command interpreter window (CIW), choose *Tools – AMS – Options*.

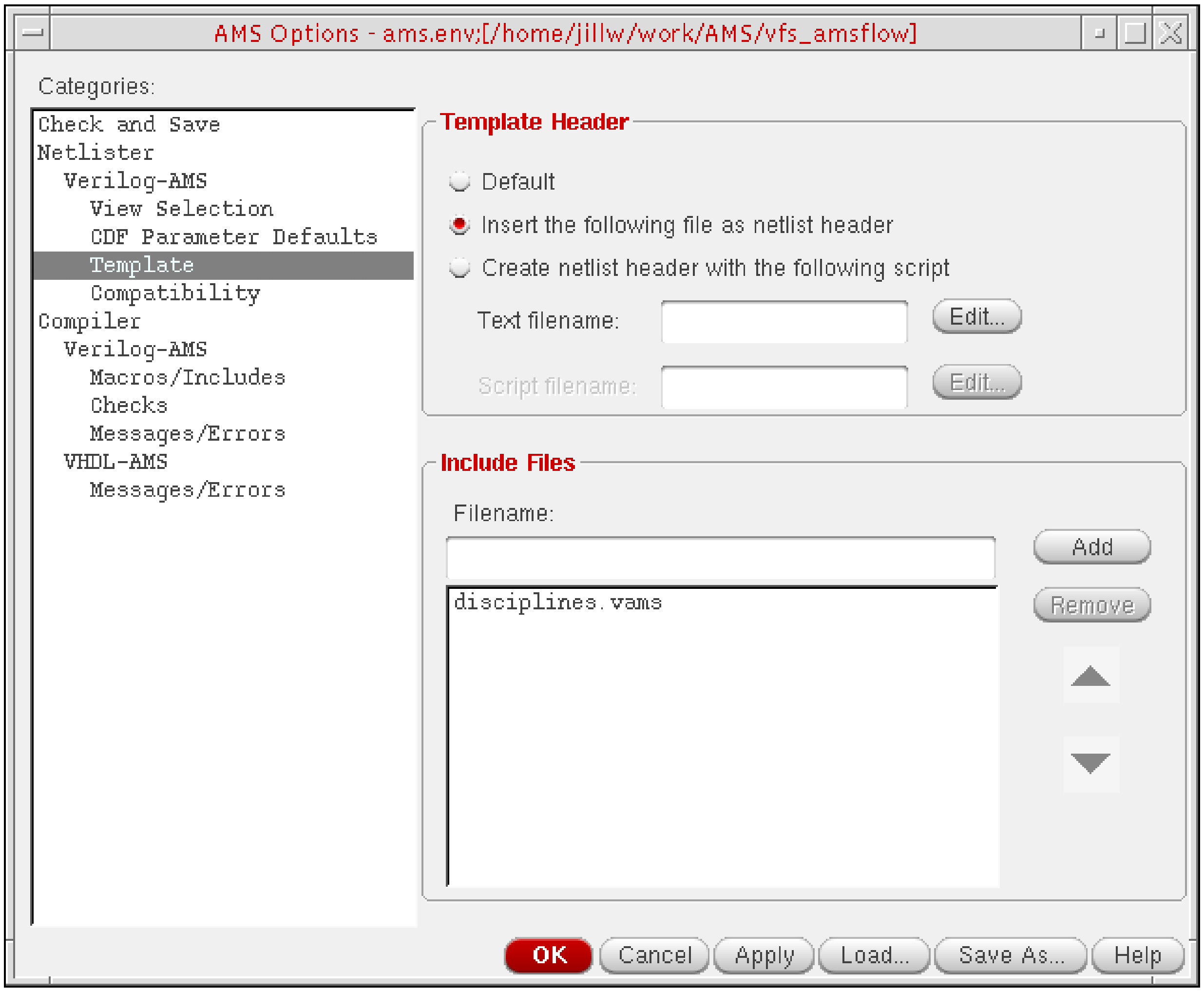
   The AMS Options form appears.

2. In the *Categories* list, select *Netlister – Verilog-AMS – Template*.

   The *Template Header* and *Include Files* group boxes appear.

3. In the *Template Header* group box, select *Insert the following file as netlist header*.

The *Text filename* field becomes active.


AMS Options - ams.env;[/home/jillw/work/AMS/vfs_amsflow]

Categories:

```
Check and Save
Netlister
  Verilog-AMS
    View Selection
    CDF Parameter Defaults
    Template
    Compatibility
Compiler
  Verilog-AMS
    Macros/Includes
    Checks
    Messages/Errors
  VHDL-AMS
    Messages/Errors
```

Template Header

Default [✓ Insert the following file as netlist header] Create netlist header with the following script

Text filename:

Script filename:

Include Files

Filename:

disciplines.vams

4. In the *Text filename* field, type the path and name of the text file that contains the header text you want to include. If you specify a relative path, the program resolves that path with respect to the directory where you started the AMS software.

   Note: If you type a new file name, you can click *Edit* to open a text editing window in which you can type the header text. Be sure to save the file before exiting the editor window.

5. Click *OK*.

   The AMS netlister inserts the default header text followed by the contents of the file you specify at the top of each netlist it generates. For example, if you have a file containing the following text:

```
// Module produced by
// ASIC Team: Ocelot
// San Jose Development Center
```

the AMS netlister inserts the following text at the top of each generated netlist:

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

// Module produced by
// ASIC Team: Ocelot
// San Jose Development Center
```

Include Header Text That Results from a Script File

To include header text that results from a script file, do the following:

1. In the command interpreter window (CIW), choose *Tools – AMS – Options*.

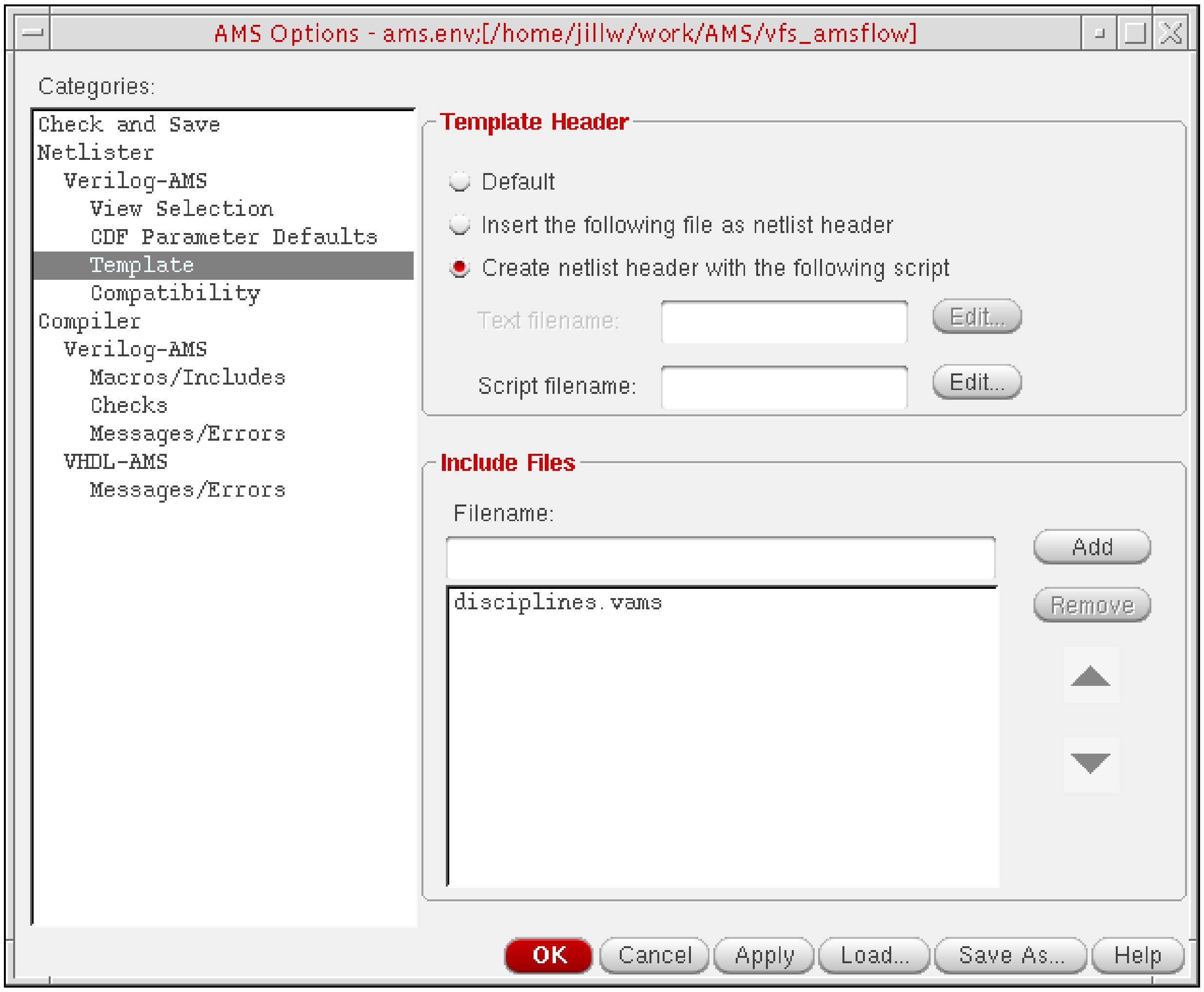
   The AMS Options form appears.

2. In the *Categories* list, select *Netlister – Verilog-AMS – Template*.

   The *Template Header* and *Include Files* group boxes appear.

3. In the *Template Header* group box, select *Create netlist header with the following script*.

The *Script filename* field becomes active.


AMS Options - ams.env;[/home/jillw/work/AMS/vfs_amsflow]

Categories:

```
Check and Save
Netlister
  Verilog-AMS
    View Selection
    CDF Parameter Defaults
    Template
    Compatibility
Compiler
  Verilog-AMS
    Macros/Includes
    Checks
    Messages/Errors
  VHDL-AMS
    Messages/Errors
```

Template Header

Default
Insert the following file as netlist header
[✓ Create netlist header with the following script]

Text filename: Edit...
Script filename: Edit...

Include Files

Filename: Add

disciplines.vams

Remove

4. In the *Script filename* field, type the path and name of the script file that contains the commands that will generate your header text. If you specify a relative path, the program resolves that path with respect to the directory where you started the AMS software.

   Note: If you type a new file name, you can click *Edit* to open a text editing window in which you can type the header text. Be sure to save the file before exiting the editor window.

5. Click *OK*.

   The AMS netlister inserts the default header text followed by the text from the script at the top of each netlist it generates. For example, if you have a file containing the following text:

```
    echo '// Module produced by:'
    echo '// ASIC Interactive, Ltd.'
    printf '// (c) '
    date '+DATE: %m/%d/%y%n'
```

the AMS netlister inserts the following text at the top of each generated netlist:

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

    // Module produced by:
    // ASIC Interactive, Ltd.
    // (c) DATE: 10/10/01
```

Conditionally Include Verilog-AMS Language Extentions

The AMS netlister adds attributes that are Cadence-specific extensions to the Verilog-AMS language to the netlist. Some third-party software and some Cadence applications interpret these attributes as illegal code and parsing fails. You can turn on the *Conditionally include language extensions* option to cause the AMS netlister to write these language extensions to the netlist using the `` `ifdef `` directive. This directive lets you control when language extensions are active for simulation.

With *Conditionally include language extensions* turned on, the following example

```
comparator
 (* integer library_binding = "amslib";
    integer cds_net_set[0:1]= {"xground","vdd"};
    integer xground[0:1] = {"new_ground","cds_globals.\\gnd5! "};
    integer vdd = "cds_globals.\\3.3v! ";  *)
I2 ( .inn( dacOut ), .inp( holdSig ), .out( compOut ) );
```

(which uses the Cadence-specific `cds_net_set` attribute) becomes

```
comparator
`ifdef INCA
 (* integer library_binding = "amslib";
    integer cds_net_set[0:1]= {"xground","vdd"};
    integer xground[0:1] = {"new_ground","cds_globals.\\gnd5! "};
    integer vdd = "cds_globals.\\3.3v! ";  *)
`endif
I2 ( .inn( dacOut ), .inp( holdSig ), .out( compOut ) );
```

`INCA` is a predefined compiler directive in the `ncvlog` compiler, so you do not need to define it.

Note: While omitting attributes allows compilation to succeed, the simulation results might be incorrect without the connection information that these attributes contains.

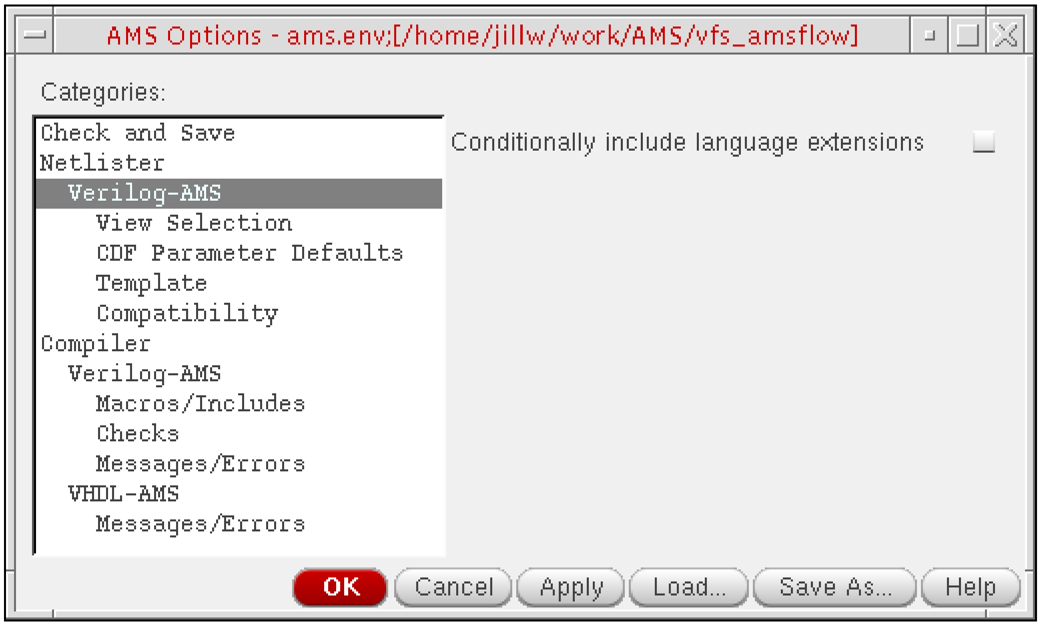
To include Verilog-AMS language extensions in the AMS netlist using a `` `ifdef `` directive (so that you can exclude these extensions for simulators that do not support them), do the following:

1. In the command interpreter window (CIW), choose *Tools – AMS – Options*.

   The AMS Options form appears.

2. In the *Categories* list, select *Netlister – Verilog-AMS*.

   The *Conditionally include language extensions* check box appears.



AMS Options - ams.env:[/home/jillw/work/AMS/vfs_amsflow]

Categories:

```
Check and Save
Netlister
  Verilog-AMS
    View Selection
    CDF Parameter Defaults
    Template
    Compatibility
Compiler
  Verilog-AMS
    Macros/Includes
    Checks
    Messages/Errors
  VHDL-AMS
    Messages/Errors
```

Conditionally include language extensions ☐

3. Turn on *Conditionally include language extensions*.

   The AMS netlister sets off language extensions in the netlist using `` `ifdef INCA `` directives so that you can exclude these language extensions for simulators that do not support them.

Eligible View Types and View Names to Exclude

The AMS netlister can translate four types of cellviews into Verilog-AMS netlists: schematic, symbolic, netlist, and maskLayout. You can specify which cellview types you want the AMS netlister to netlist. Further, you can specify particular view names you do not want the AMS netlister to netlist.

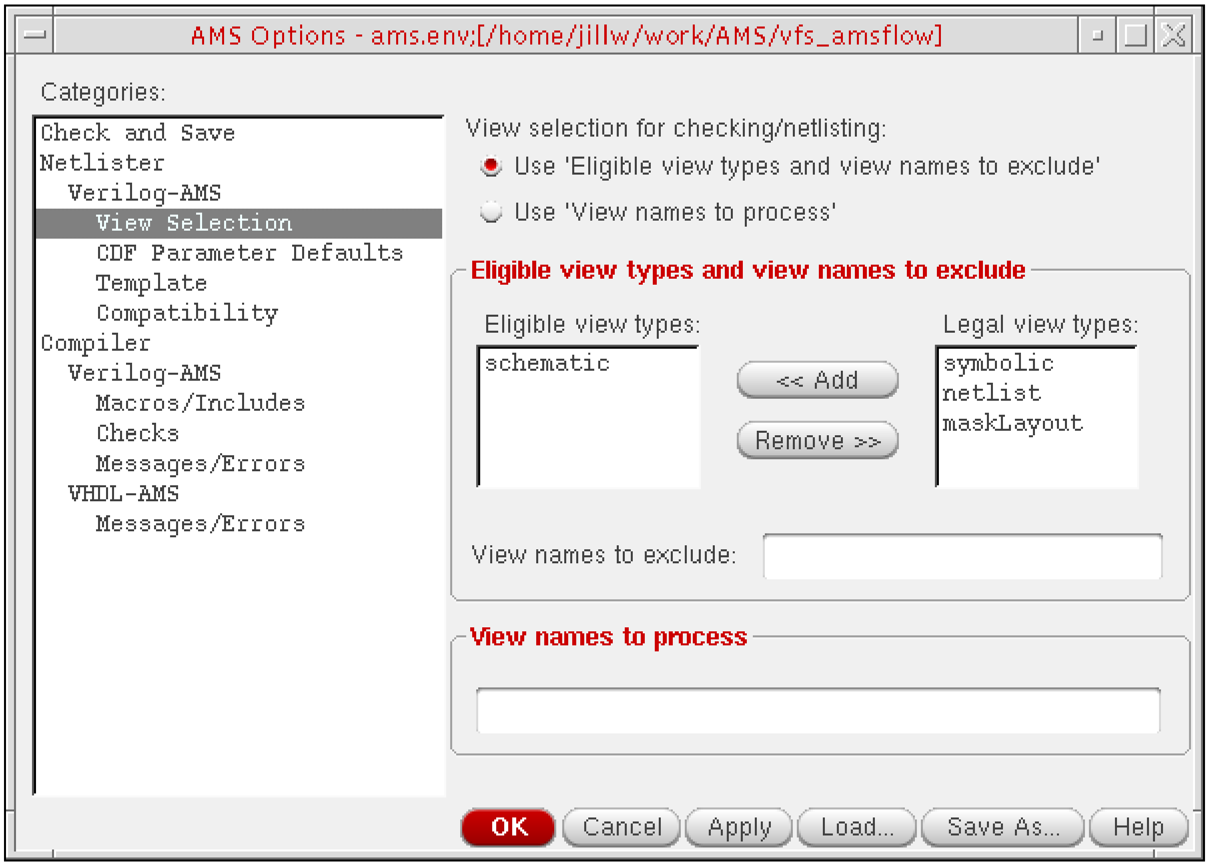
To specify eligible view types and (optionally) view names to exclude, do the following:

1. In the command interpreter window (CIW), choose *Tools – AMS – Options*.

   The AMS Options form appears.

2. In the *Categories* list, select *Netlister – Verilog-AMS – View Selection*.

   The selection options appear.



AMS Options - ams.env;[/home/jillw/work/AMS/vfs_amsflow]

Categories:

```
Check and Save
Netlister
  Verilog-AMS
    View Selection
    CDF Parameter Defaults
    Template
    Compatibility
Compiler
  Verilog-AMS
    Macros/Includes
    Checks
    Messages/Errors
  VHDL-AMS
    Messages/Errors
```

View selection for checking/netlisting:
[✓ Use 'Eligible view types and view names to exclude']
Use 'View names to process'

Eligible view types and view names to exclude

Eligible view types: schematic

Legal view types: symbolic, netlist, maskLayout

View names to exclude:

View names to process

3. Under *View selection for checking/netlisting*, select the *Use ‘Eligible view types and view names to exclude’* radio button.

4. For each cellview type you want the AMS netlister to netlist, do the following:

a. In the *Legal view types* list box, select a cellview type.

b. Click *Add*.

   The selected cellview type moves from the *Legal view types* list box to the *Eligible view types* list box.

   Note: If necessary, you can select a cellview type in the *Eligible view types* list box and click *Remove* to make that view type ineligible for netlisting.

5. (Optional) In the *View names to exclude* field, type a space-separated list of one or more view names you do not want the AMS netlister to netlist.

   You can specify a range of names using square brackets. For example, the following specification excludes all cellviews named `sch0`, `sch1`, `sch2`, and `sch3`:

   ```
   sch[0-3]
   ```

6. Click *OK*.

   The AMS netlister will netlist those cellview types you specified and will exclude from netlisting those cellview names you specified.

View Names to Process

To specify particular view names you want the AMS netlister to netlist, do the following:

1. In the command interpreter window (CIW), choose *Tools – AMS – Options*.

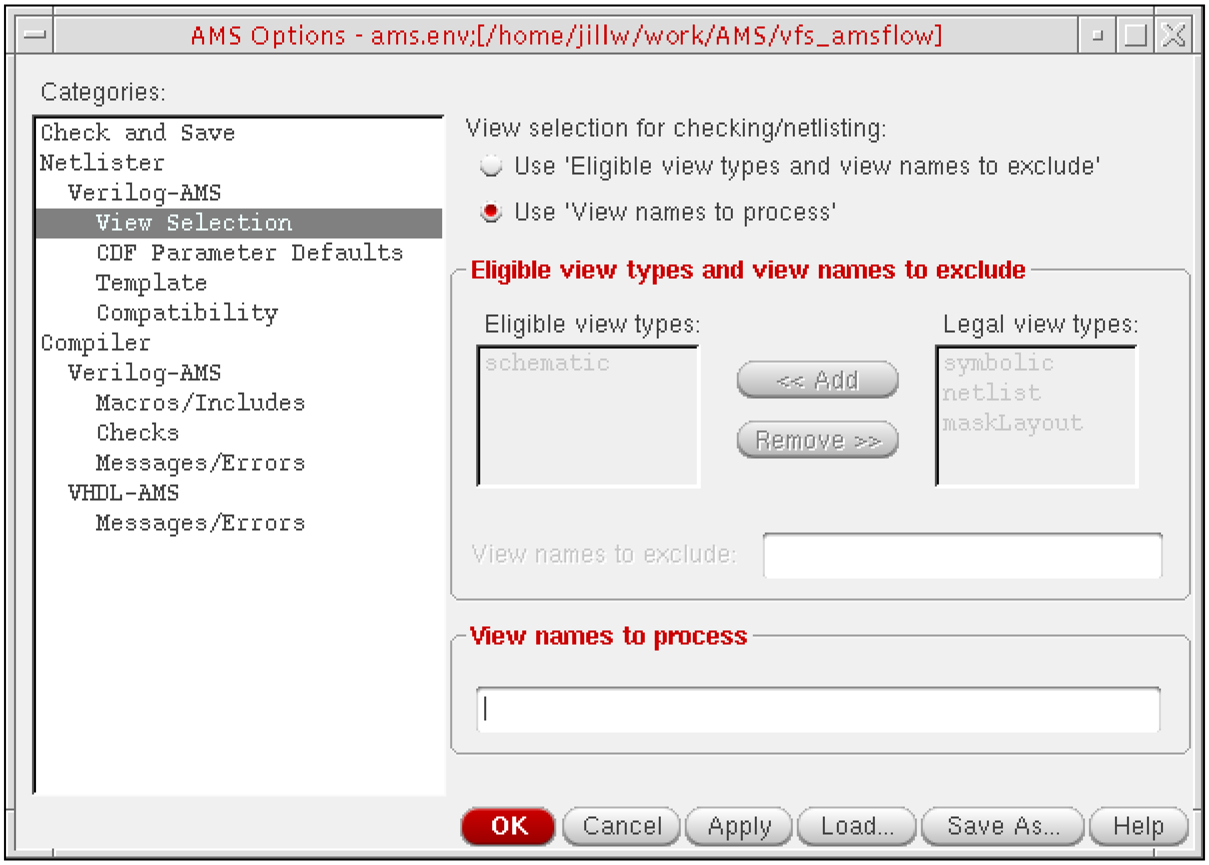
   The AMS Options form appears.

2. In the *Categories* list, select *Netlister – Verilog-AMS – View Selection*.

   The selection options appear.

3. Under *View selection for checking/netlisting*, select the *Use 'View names to process'* radio button.

   The *View names to process* field becomes active.



AMS Options - ams.env;[/home/jillw/work/AMS/vfs_amsflow]

Categories:

```
Check and Save
Netlister
  Verilog-AMS
    View Selection
    CDF Parameter Defaults
    Template
    Compatibility
Compiler
  Verilog-AMS
    Macros/Includes
    Checks
    Messages/Errors
  VHDL-AMS
    Messages/Errors
```

View selection for checking/netlisting:
- Use 'Eligible view types and view names to exclude'
- [✓ Use 'View names to process']

Eligible view types and view names to exclude

Eligible view types: schematic

Legal view types: symbolic, netlist, maskLayout

View names to exclude:

View names to process

4. In the *View names to process* field, type a space-separated list of one or more view names you want the AMS netlister to netlist.

You can specify a range of names using square brackets. For example, the following specification causes the AMS netlister to netlist only cellviews named `sch1`, `sch3`, and `sch4`:

```
sch1 sch[3-4]
```

5. Click *OK*.

The AMS netlister will netlist only cellviews whose names match the ones you specified.

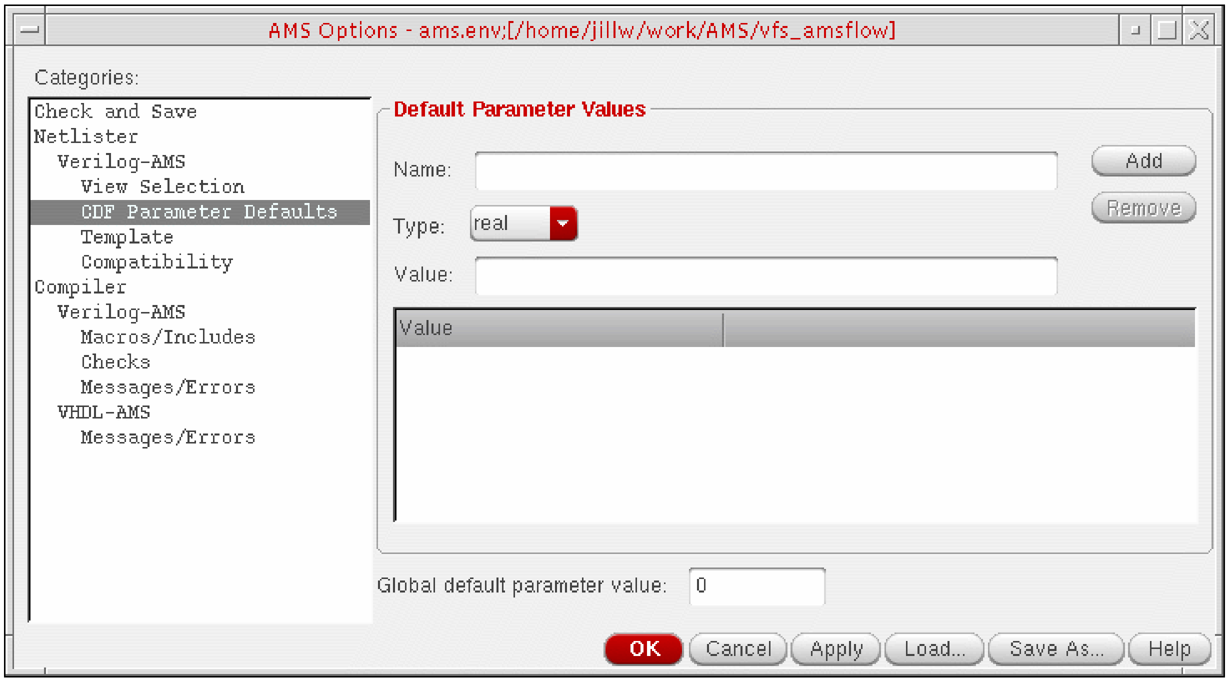
CDF Parameter Defaults

To specify default values for CDF parameters, do the following:

1. In the command interpreter window (CIW), choose *Tools – AMS – Options*.

   The AMS Options form appears.

2. In the *Categories* list, select *Netlister – Verilog-AMS – CDF Parameter Defaults*.



3. In the *Default Parameter Values* group box, specify default values for specific CDF parameters as follows:

   a. In the *Name* field, type the name of the parameter.

   b. In the *Type* field, select *integer* or *real* to indicate the type of the parameter.

   c. In the *Value* field, type a default value.

   d. Click *Add*.

      The newly-specified default value appears in the table. The AMS netlister uses this default value when the specified parameter does not already have one.

Note: You can remove a parameter from the table by selecting the parameter in the table and clicking *Remove*.

4. (Optional) In the *Global default parameter value* field, type a global default value for CDF parameters.

   The AMS netlister uses this default value for any parameter that does not otherwise have a default value.

Verilog-AMS Compatibility Exceptions

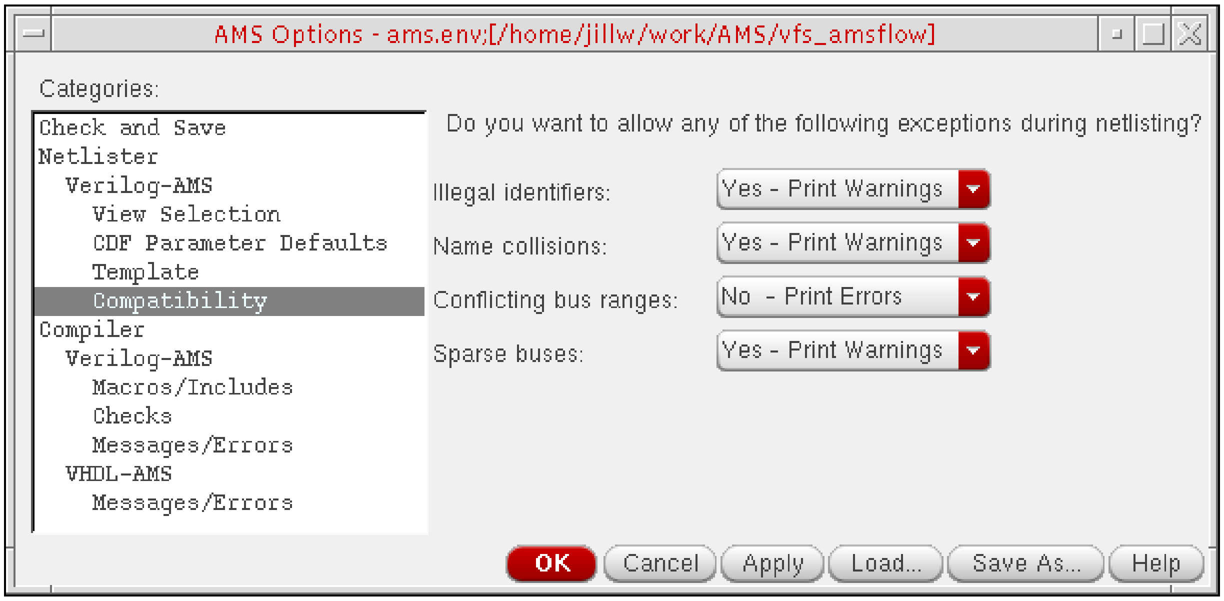
To specify how you want the AMS netlister to handle Verilog-AMS compatibility exceptions, do the following:

1. In the command interpreter window (CIW), choose *Tools – AMS – Options*.

   The AMS Options form appears.

2. In the *Categories* list, select *Netlister – Verilog-AMS – Compatibility*.

   The compatibility drop-down combo boxes appear.



Do you want to allow any of the following exceptions during netlisting?

Illegal identifiers: Yes - Print Warnings
Name collisions: Yes - Print Warnings
Conflicting bus ranges: No - Print Errors
Sparse buses: Yes - Print Warnings

3. For each exception, select one of the following choices from the drop-down combo box:

   ❑ *No – Print Errors*: The AMS netlister issues an error if the netlister encounters the design exception and does not generate a netlist.

   ❑ *Yes – Print Warnings*: The AMS netlister issues a warning if the netlister encounters the design exception. Netlisting continues after the netlister makes necessary changes (see below).

   ❑ *Yes – Silently*: The AMS netlister continues netlisting if it encounters the design exception, after making changes that allow netlisting to continue (see below), and does not issue a warning.

4. Click *OK*.

   If you selected *Yes* from any of the drop-down combo boxes, the AMS netlister makes the following changes when it encounters the specified design exception:

Exception	**AMS Netlister Action**
Illegal identifiers	Maps noncompliant identifiers to names that are legal in the target language. The associated warning, if issued, tells you how the name is mapped.
Name collisions	Maps noncompliant names to system-generated names that are legal in the target language.
Conflicting bus ranges	Netlisting continues if it is possible to create a valid netlist. The associated warning, if issued, tells you how the noncompliant bus data is transformed. **Note:** The generated netlist is likely to be less readable than one created from compliant bus data.
Sparse buses	Overdeclares any sparse buses.

   Note: For schematic-based designs, see also "Specifying Schematic Rules Checking for AMS Designer" on page 224.

Netlisting User-Defined Functions

The AMS netlister does not support calls to Spectre or SPICE user-defined functions (UDFs) from a schematic, but it does support macro references. You can convert UDF calls into macro references by doing the following:

Important

You must make sure that the referenced macros are defined and that the definitions are accessible during netlisting.

1. Add the variable `netlistUDFAsMacro` to the `ams.env` file in your working directory, with the value set to `t`.

   ```
   amsDirect.vlog netlistUDFAsMacro boolean t
   ```

   This variable setting tells the netlister to convert UDF calls into macro references.

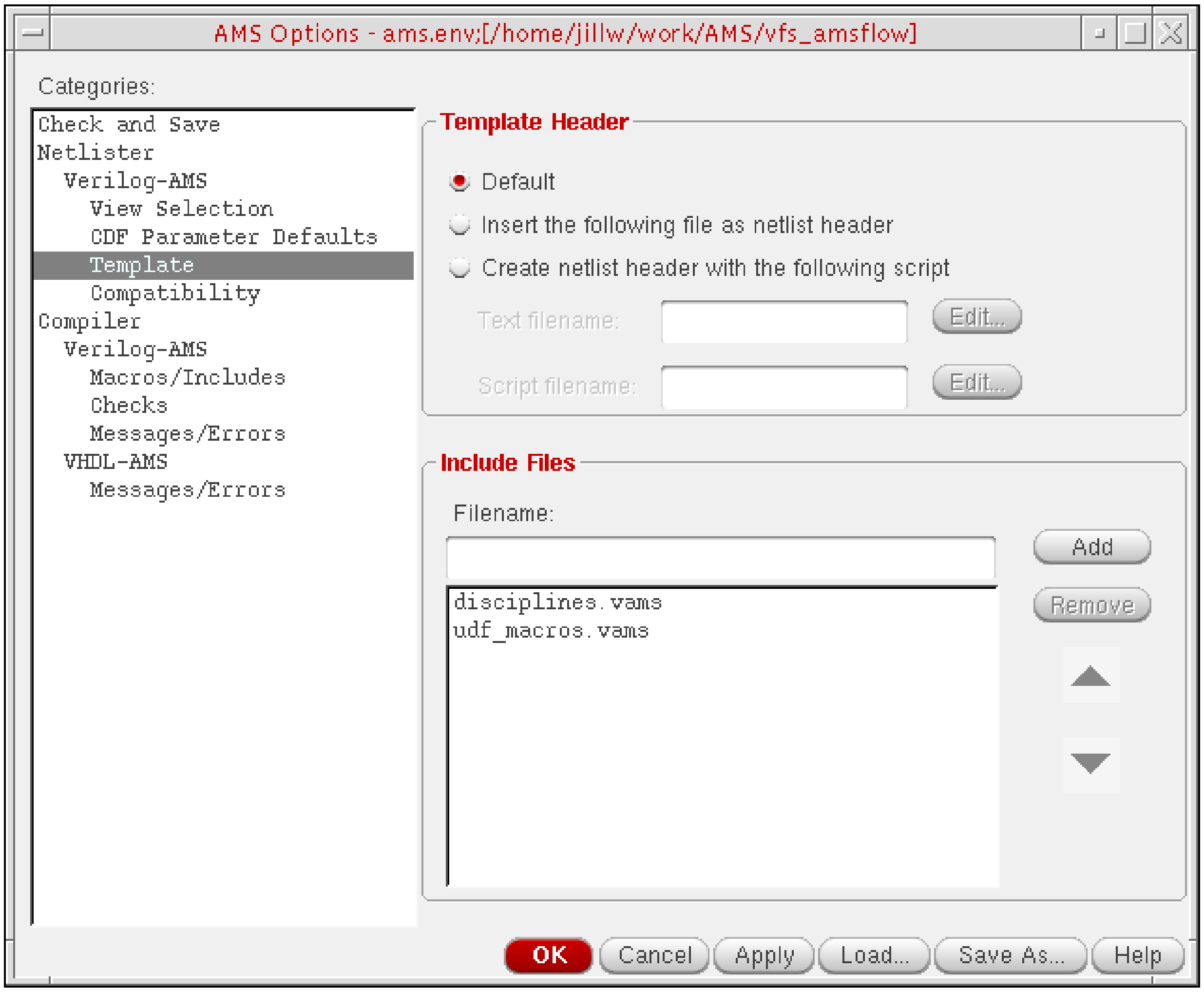
2. Create a file containing the macro definitions.

Begin each definition with `` `define `` and enclose the actual function in parentheses. For example:

```
`define mod(a,b) ( (a) - (b)*ceil(((a)+0.5)/(b)) )
`define int(a) ( (abs(a)==(a)) ? floor(a) : ceil(a) )
```

3. Add the definitions file in the *Include Files* group box on the AMS Options form for *Netlister – Verilog-AMS – Template*.

   For example, if your definitions file is `udf_macros.vams`, your AMS Options form might look like this once you add it:



AMS Options - ams.env;[/home/jillw/work/AMS/vfs_amsflow]

Categories:

```
Check and Save
Netlister
  Verilog-AMS
    View Selection
    CDF Parameter Defaults
    Template
    Compatibility
Compiler
  Verilog-AMS
    Macros/Includes
    Checks
    Messages/Errors
  VHDL-AMS
    Messages/Errors
```

Template Header

[✓ Default]
Insert the following file as netlist header
Create netlist header with the following script
Text filename:
Script filename:

Include Files

Filename:

```
disciplines.vams
udf_macros.vams
```

4. Add the directory containing your definitions file in the *Include Directories* group box on the AMS Options form for *Compiler – Verilog-AMS – Macros/Includes*.

5. On the AMS Options form, click *OK*.

Specifying Compiler Options from the CIW

You can specify the following compiler options from the CIW:

- hdl.var File on page 629
- Verilog-AMS Compiler Options on page 631
- Verilog-AMS Macros to Use during Compilation on page 634
- Directories to Search for Verilog-AMS Include Files on page 636
- Checks for Verilog-AMS Modules on page 638
- Verilog-AMS Compiler Message Options on page 640
- VHDL-AMS Compiler Options on page 642
- VHDL-AMS Compiler Message Options on page 645

hdl.var File

An `hdl.var` file contains variables and settings for the compiler, elaborator, and simulator. For information about the `hdl.var` file, see “The hdl.var File” in the *Virtuoso AMS Designer Simulator User Guide*.

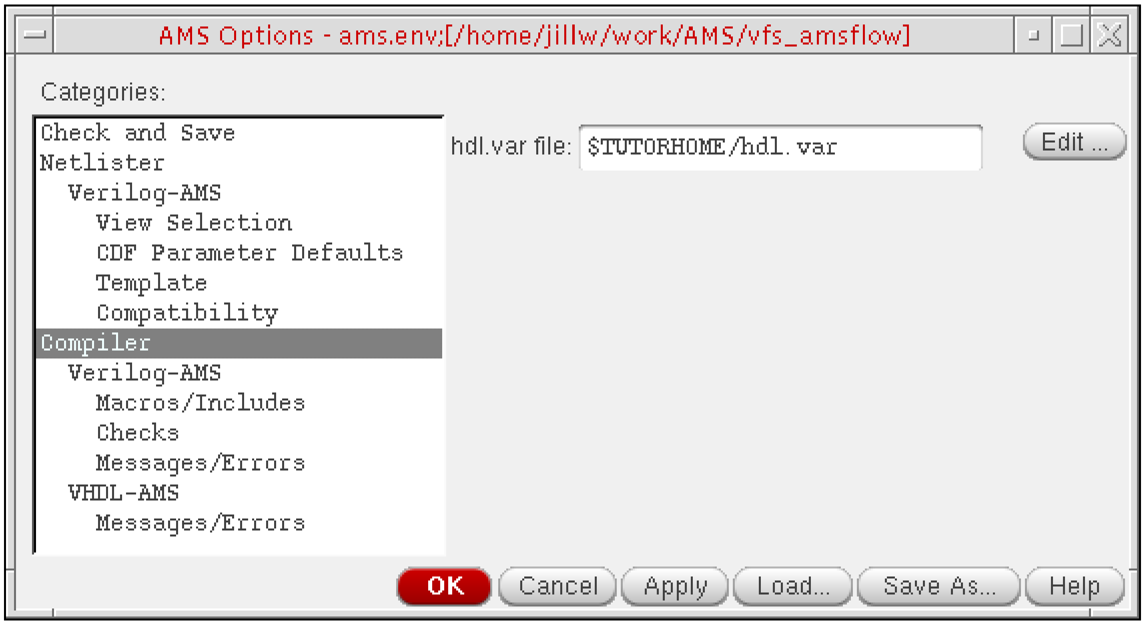
To specify an `hdl.var` file in the AMS Designer environment, do the following:

1. In the command interpreter window (CIW), choose *Tools – AMS – Options*.

   The AMS Options form appears.

2. In the *Categories* list, select *Compiler*.

   The *hdl.var file* field appears.



AMS Options - ams.env:[/home/jillw/work/AMS/vfs_amsflow]

Categories:

```
Check and Save
Netlister
  Verilog-AMS
    View Selection
    CDF Parameter Defaults
    Template
    Compatibility
Compiler
  Verilog-AMS
    Macros/Includes
    Checks
    Messages/Errors
  VHDL-AMS
    Messages/Errors
```

hdl.var file: `$TUTORHOME/hdl.var`

3. In the *hdl.var file* field, type the fully-qualified path and name of the `hdl.var` file you want to use.

   If you specify a relative path, the program resolves that path with respect to the directory where you started the AMS software. The compiler runs from the directory where you started the AMS software, but the elaborator and simulator start from the run directory, so you should use an absolute path or an environment variable that expresses a fully-qualified path to be sure that the compiler, elaborator, and simulator all use the same `hdl.var` file.

   Note: If you type a new file name, you can click *Edit* to open a text editing window in which you can type the content of the `hdl.var` file. Be sure to save the file before exiting the editor window.

4. Click *OK*.

   The AMS compilers use the specified `hdl.var` file during compilation.

Verilog-AMS Compiler Options

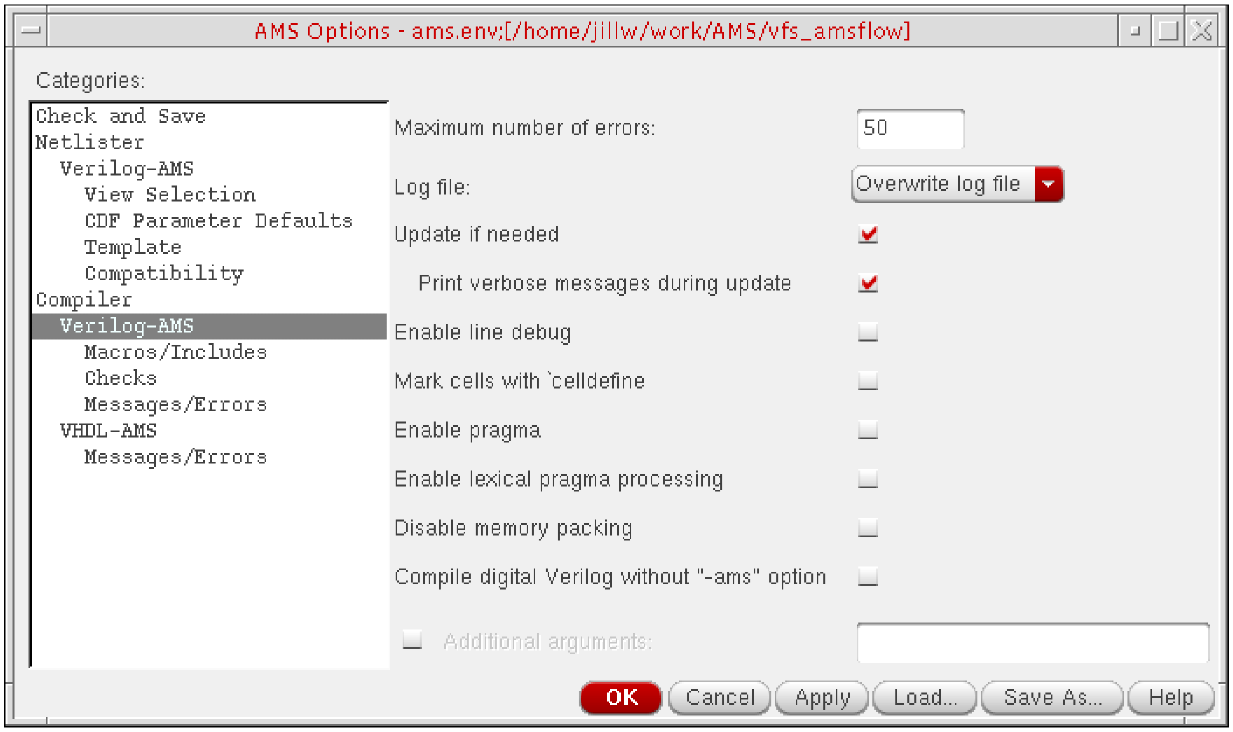
To specify options for the Verilog-AMS compiler, do the following:

1. In the command interpreter window (CIW), choose *Tools – AMS – Options*.

   The AMS Options form appears.

2. In the *Categories* list, select *Compiler – Verilog-AMS*.

   The Verilog-AMS compiler options appear.



AMS Options - ams.env:[/home/jillw/work/AMS/vfs_amsflow]

Categories:

```
Check and Save
Netlister
  Verilog-AMS
    View Selection
    CDF Parameter Defaults
    Template
    Compatibility
Compiler
  Verilog-AMS
    Macros/Includes
    Checks
    Messages/Errors
  VHDL-AMS
    Messages/Errors
```

Maximum number of errors:	50
Log file:	Overwrite log file
Update if needed	✓
Print verbose messages during update	✓
Enable line debug	☐
Mark cells with `celldefine	☐
Enable pragma	☐
Enable lexical pragma processing	☐
Disable memory packing	☐
Compile digital Verilog without "-ams" option	☐
☐ Additional arguments:	

Note: The options you can specify on this form correspond to certain `ncvlog` command-line options as indicated. For information about these `ncvlog` command-line options, see "ncvlog Command Options" in "Compiling Verilog Source Files with ncvlog" in *Cadence NC-Verilog Simulator Help*.

3. (Optional) In the *Maximum number of errors* field, type a number to specify the maximum number of errors the compiler can encounter before it stops compiling.

   Note: This option corresponds to the `ncvlog -errormax` option.

4. (Optional) Using the *Log file* drop-down combo box, select one of the following choices:

Overwrite log file	Overwrites any existing log file; the default log file name is `ncvlog.log`
Append log file	Appends log information to the existing log file; corresponds to the `ncvlog -append_log` option
No log file	Does not generate any log file; corresponds to the `ncvlog -nolog` option

5. (Optional) Turn on one or more of the following options (some of which might already be turned on by default):

Update if needed	Recompiles the design after design units, source files, or compiler directives are added, or if a design unit is changed in a way that introduces a new cross-file dependency; corresponds to the `ncvlog -update` option
Print verbose messages during update	Prints the names of up-to-date modules that otherwise are not printed in the log file; corresponds to the `ncvlog -uptodate_messages` option
Enable line debug	Enables support for setting line breakpoints and for single-stepping through code; corresponds to the `ncvlog -linedebug` option
Mark cells with ‘celldefine	Inserts `` `celldefine `` and `` `endcelldefine `` compiler directives to tag module instances as cell instances; corresponds to the `ncvlog -libcell` option
Enable pragma	Parses pragmas contained in HDL source files; corresponds to the `ncvlog -pragma` option **Note:** This check box becomes inactive if you turn on *Enable lexical pragma processing*.

Enable lexical pragma processing

Parses pragmas contained in HDL source files and treats `translate off` and `translate on` as if they are Verilog `` `ifdef 0 `` and `` `endif `` so that the code between them is not included during compilation; corresponds to the `ncvlog -lexpragma` option

Disable memory packing

Prepares design units for access by the PLI routine `tf_nodeinfo`; corresponds to the `ncvlog -nomempack` option

Compile digital Verilog without "-ams" option

Omits the `-ams` command line option when running `ncvlog` on files named `verilog.v`. If a file named `verilog.v` is actually a link, the decision to use or omit the `-ams` option is based on the extension of the name of the physical file that is the target of the link; corresponds to the `ncvlog -ams` option

6. (Optional) In the *Additional arguments* field, type any additional arguments you want the Verilog-AMS compiler to use.

   Note: You must not specify a `-log` argument because the compiler automatically writes the default log file, `ncvlog.log`, to the run directory (unless you select *No log file*).

7. Click *OK*.

   The Verilog-AMS compiler uses the options you specified.

Verilog-AMS Macros to Use during Compilation

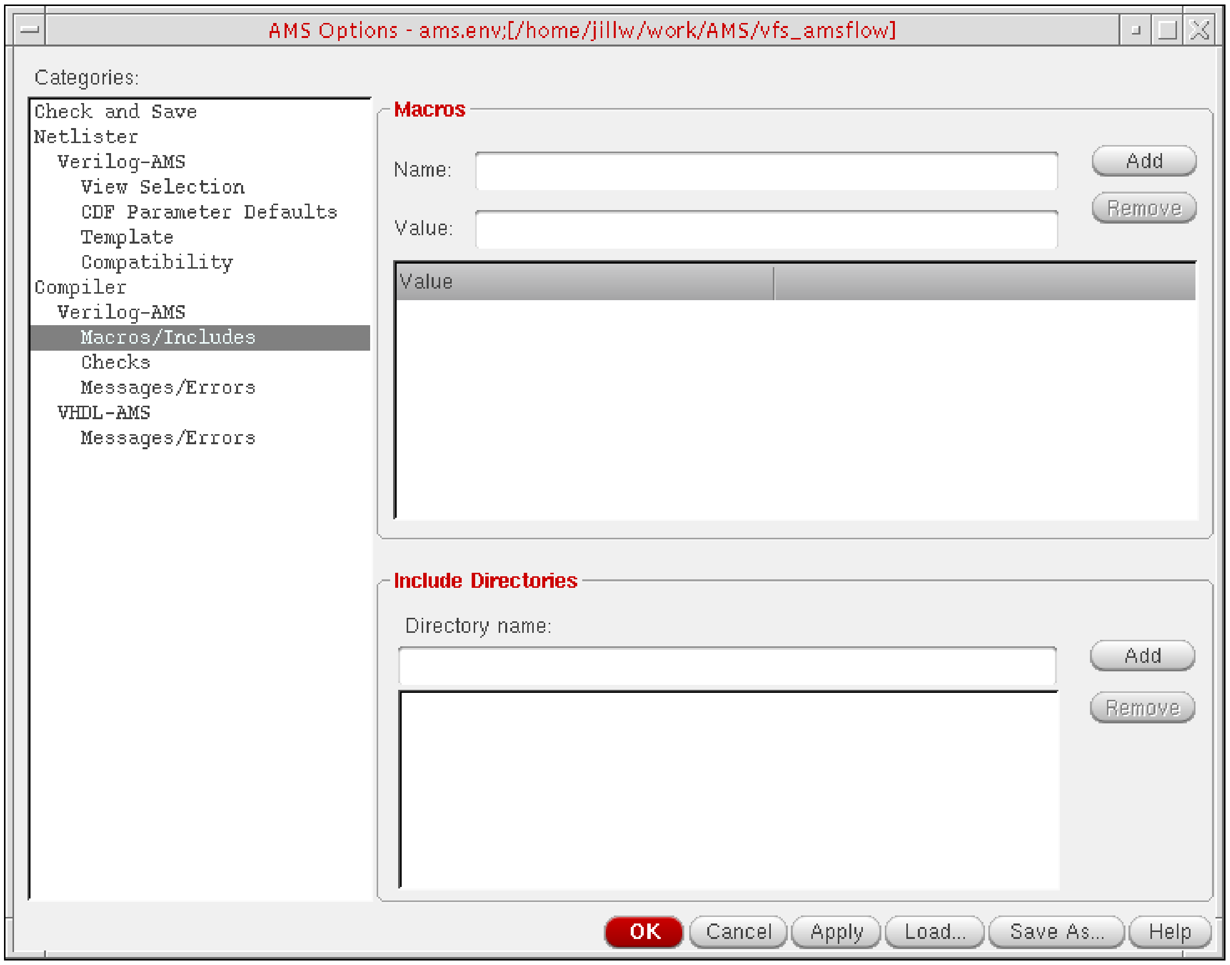
To specify Verilog-AMS macros to use during compilation, do the following:

1. In the command interpreter window (CIW), choose *Tools – AMS – Options*.

   The AMS Options form appears.

2. In the *Categories* list, select *Compiler – Verilog-AMS – Macros/Includes*.

   The *Macros* and *Include Directories* group boxes appear.



3. For each macro you want to add, do the following in the *Macros* group box

   a. In the *Name* field, type a macro name.

   b. Click *Add*.

This option corresponds to the `ncvlog -define` option. See “ncvlog Command Options” in “Compiling Verilog Source Files with ncvlog” in *Cadence NC-Verilog Simulator Help* for more information.

4. Click *OK*.

   The Verilog-AMS compiler uses these macros.

Directories to Search for Verilog-AMS Include Files

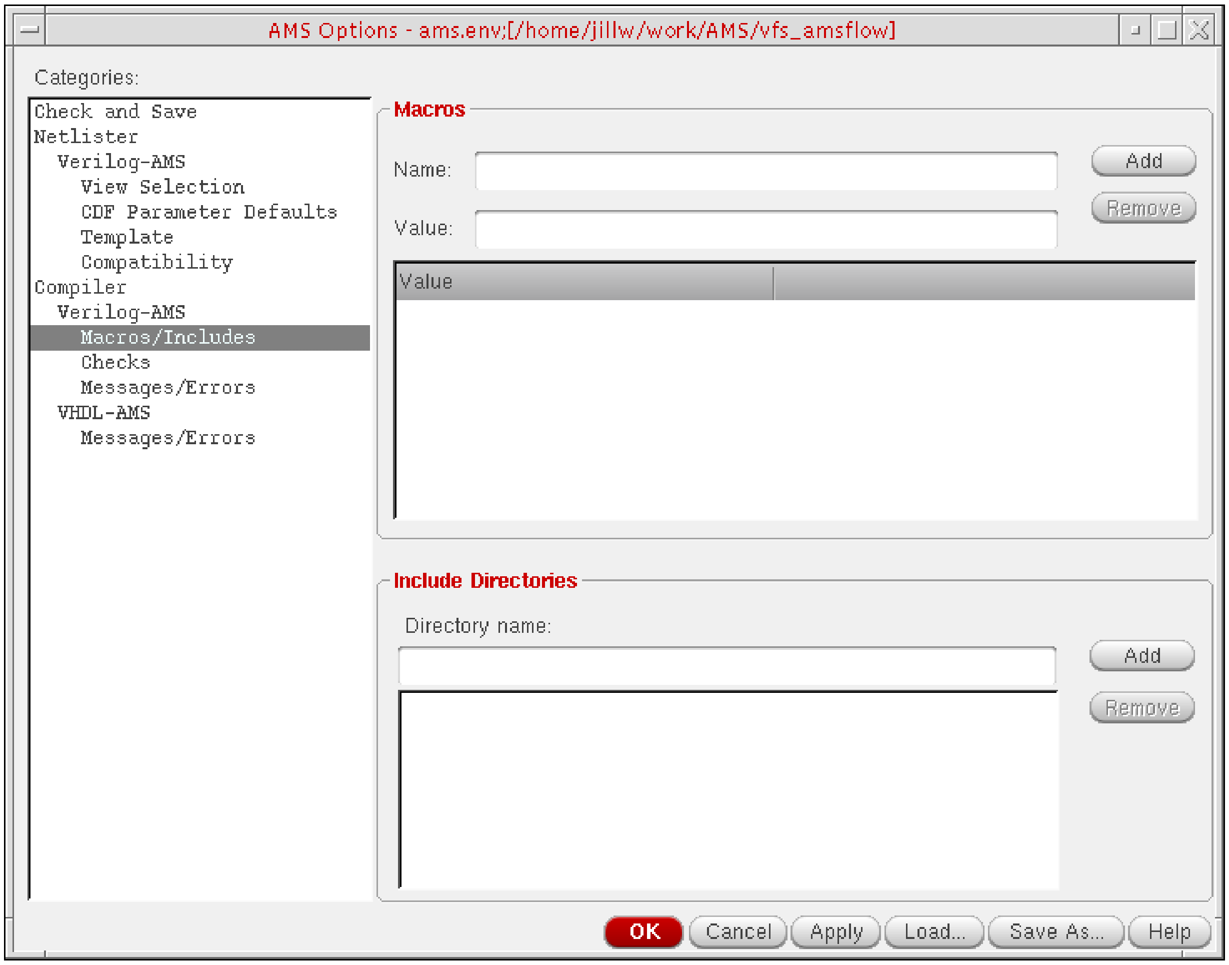
To specify directories to search for Verilog-AMS include files during compilation, do the following:

1. In the command interpreter window (CIW), choose *Tools – AMS – Options*.

   The AMS Options form appears.

2. In the *Categories* list, select *Compiler – Verilog-AMS – Macros/Includes*.

   The *Macros* and *Include Directories* group boxes appear.



3. For each directory path you want to specify, do the following in the *Include Directories* group box:

   a. In the *Directory name* field, type a directory path.

b. Click *Add*.

This option corresponds to the `ncvlog -incdir` option. For more information, see “ncvlog Command Options” in “Compiling Verilog Source Files with ncvlog” in *Cadence NC-Verilog Simulator Help*.

4. Click *OK*.

The Verilog-AMS compiler searches these directories for include files.

Checks for Verilog-AMS Modules

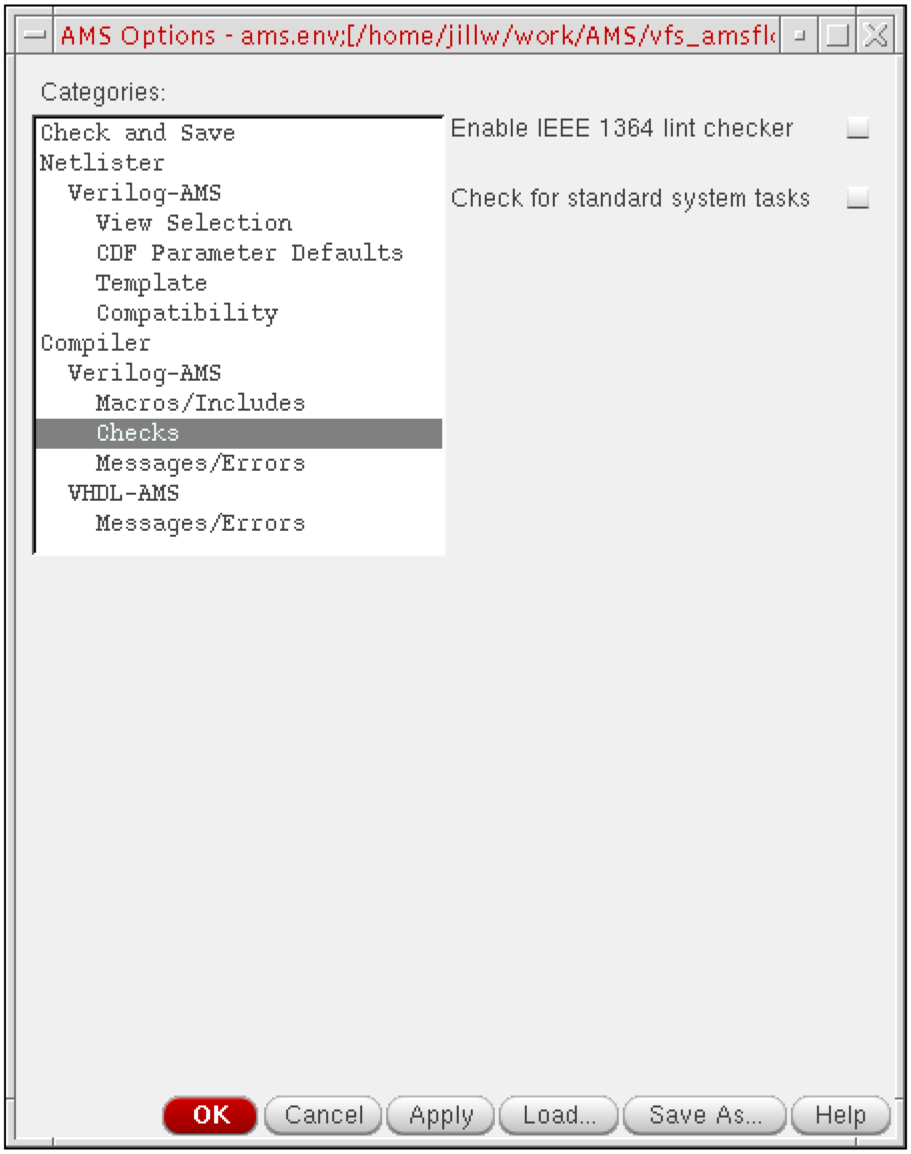
To specify what checks you want the compiler to perform on Verilog-AMS modules, do the following:

1. In the command interpreter window (CIW), choose *Tools – AMS – Options*.

   The AMS Options form appears.

2. In the *Categories* list, select *Compiler – Verilog-AMS – Checks*.

   The check options appear.



Enable IEEE 1364 lint checker ☐

Check for standard system tasks ☐

3. Turn on one or both of the following checks:

Enable IEEE 1364 lint checker

Checks the source code for compatibility with the IEEE standard described in *IEEE-1364 Verilog Hardware Description Language Reference Manual*; corresponds to the `ncvlog -ieee1364` option

Check for standard system tasks

Checks for the presence of any non-predefined system tasks or functions in the source code; corresponds to the `ncvlog -checktasks` option

For information about these `ncvlog` options, see “ncvlog Command Options” in “Compiling Verilog Source Files with ncvlog” in *Cadence NC-Verilog Simulator Help*.

4. Click *OK*.

The Verilog-AMS compiler performs the specified checks.

Verilog-AMS Compiler Message Options

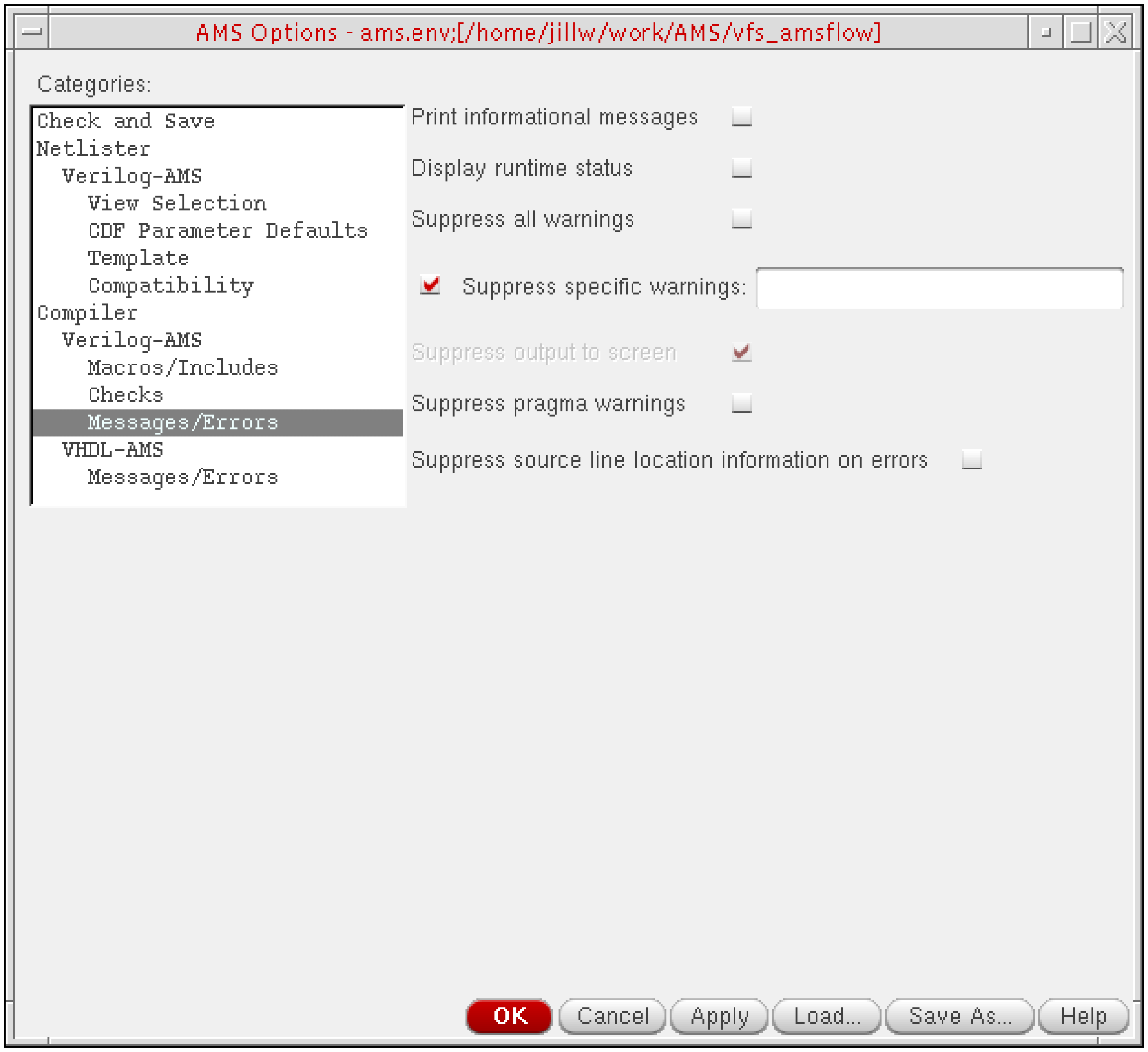
To control the output of Verilog-AMS compiler messages, do the following:

1. In the command interpreter window (CIW), choose *Tools – AMS – Options*.

   The AMS Options form appears.

2. In the *Categories* list, select *Compiler – Verilog-AMS – Messages/Errors*.

   The message options appear.


AMS Options - ams.env;[/home/jillw/work/AMS/vfs_amsflow]

Categories:

```
Check and Save
Netlister
  Verilog-AMS
    View Selection
    CDF Parameter Defaults
    Template
    Compatibility
Compiler
  Verilog-AMS
    Macros/Includes
    Checks
    Messages/Errors
  VHDL-AMS
    Messages/Errors
```

Print informational messages ☐
Display runtime status ☐
Suppress all warnings ☐
✓ Suppress specific warnings:
Suppress output to screen ✓
Suppress pragma warnings ☐
Suppress source line location information on errors ☐

3. Turn on one or more of the following options:

Print informational messages

Prints informational messages as the compiler runs; corresponds to the `ncvlog -messages` option

Display runtime status

Prints statistics on memory and CPU usage after compilation; corresponds to the `ncvlog -status` option

Suppress all warnings

Suppresses all warning messages; corresponds to the `ncvlog -neverwarn` option

Suppress specific warnings

Suppresses warning messages according to the comma- or space-separated list of codes you type in the field; corresponds to the `ncvlog -nowarn` option

Suppress output to screen

Suppresses output to the screen but does not change what is written to the log file; corresponds to the `ncvlog -nostdout` option

Suppress pragma warnings

Suppresses warning messages related to pragmas; corresponds to the `ncvlog -nopragmawarn` option

Suppress source line location information on errors

Tells the compiler not to locate the source line of errors, potentially improving performance; corresponds to the `ncvlog -noline` option

For information about these `ncvlog` options, see “ncvlog Command Options” in “Compiling Verilog Source Files with ncvlog” in *Cadence NC-Verilog Simulator Help*.

4. Click *OK*.

The Verilog-AMS compiler outputs messages according to your specifications.

VHDL-AMS Compiler Options

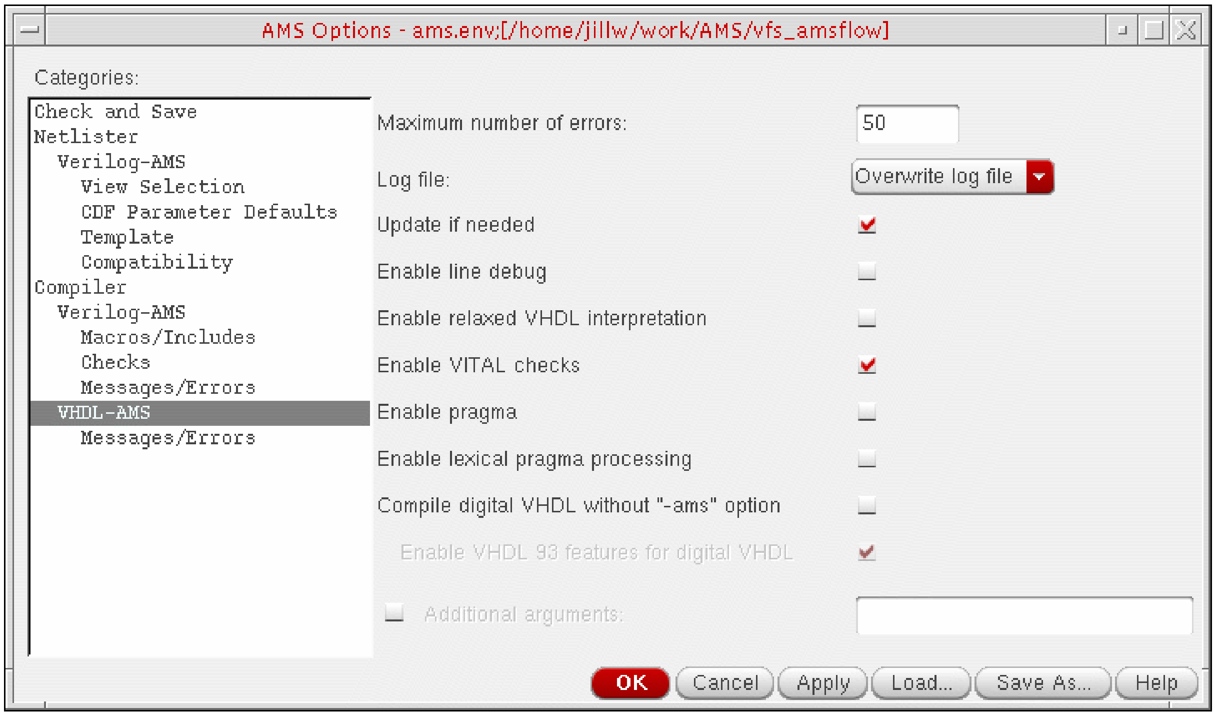
To specify options for the VHDL-AMS compiler, do the following:

1. In the command interpreter window (CIW), choose *Tools – AMS – Options*.

   The AMS Options form appears.

2. In the *Categories* list, select *Compiler – VHDL-AMS*.

   The VHDL-AMS compiler options appear.



AMS Options - ams.env:[/home/jillw/work/AMS/vfs_amsflow]

Categories:

```
Check and Save
Netlister
  Verilog-AMS
    View Selection
    CDF Parameter Defaults
    Template
    Compatibility
Compiler
  Verilog-AMS
    Macros/Includes
    Checks
    Messages/Errors
  VHDL-AMS
    Messages/Errors
```

Maximum number of errors:	50
Log file:	Overwrite log file
Update if needed	✓
Enable line debug	☐
Enable relaxed VHDL interpretation	☐
Enable VITAL checks	✓
Enable pragma	☐
Enable lexical pragma processing	☐
Compile digital VHDL without "-ams" option	☐
Enable VHDL 93 features for digital VHDL	✓
☐ Additional arguments:	

   Note: The options you can specify on this form correspond to certain `ncvhdl` command-line options as indicated. For information about these `ncvhdl` command-line options, see “ncvhdl Command Options” in “Compiling VHDL Source Files with ncvhdl” in *Cadence NC-VHDL Simulator Help*.

3. (Optional) In the *Maximum number of errors* field, type a number to specify the maximum number of errors the compiler can encounter before it stops compiling.

   Note: This option corresponds to the `ncvhdl -errormax` option.

4. (Optional) Using the *Log file* drop-down combo box, select one of the following choices:

Overwrite log file	Overwrites any existing log file; the default log file name is `ncvhdl.log`
Append log file	Appends log information to the existing log file; corresponds to the `ncvhdl -append_log` option
No log file	Does not generate any log file; corresponds to the `ncvhdl -nolog` option

5. (Optional) Turn on one or more of the following options (some of which might already be turned on by default):

Update if needed	Recompiles the design after design units, source files, or compiler directives are added, or when a design unit is changed in a way that introduces a new cross-file dependency; corresponds to the `ncvhdl -update` option **Note:** You should turn on this option for almost all circumstances. Not turning on this option can result in timestamp mismatches between entities and their corresponding architectures.
Enable line debug	Enables support for setting line and process breakpoints, and for single-stepping through code; corresponds to the `ncvhdl -linedebug` option
Enable VITAL checks	Turns on VITAL compliance checking; corresponds to the `ncvhdl -novitalcheck` option
Enable relaxed VHDL interpretation	Relaxes the interpretation of some VHDL rules; corresponds to the `ncvhdl -relax` option
Enable pragma	Parses pragmas contained in HDL source files; corresponds to the `ncvhdl -pragma` option **Note:** This check box becomes inactive if you turn on *Enable lexical pragma processing*.
Enable lexical pragma processing	

Parses pragmas contained in HDL source files and treats `translate off` and `translate on` as if they are Verilog `` `ifdef 0 `` and `` `endif `` so that the code between them is not included during compilation; corresponds to the `ncvhdl -lexpragma` option

Compile digital VHDL without "-ams" option

Omits the `-ams` command line option when running `ncvhdl` on files named `verilog.vhd`. If a file named `verilog.vhd` is actually a link, the decision to use or omit the `-ams` option is based on the extension of the name of the physical file that is the target of the link; corresponds to the `ncvhdl -ams` option

Note: Using the `-ams` option for a VHDL cellview forces `ncvhdl` to use the `-v93` option also, whether or not the cellview contains any analog features.

Enable VHDL 93 features for digital VHDL

Enables supported VHDL-93 features; corresponds to the `ncvhdl -v93` option

Note: The compiler uses the `-v93` option automatically whenever you specify `-ams` for a VHDL cellview.

6. (Optional) In the *Additional arguments* field, type any additional arguments you want the VHDL-AMS compiler to use.

   Note: You must not specify a `-log` argument because the compiler automatically writes the default log file, `ncvhdl.log`, to the run directory (unless you select *No log file*).

7. Click *OK*.

   The VHDL-AMS compiler uses the options you specified.

VHDL-AMS Compiler Message Options

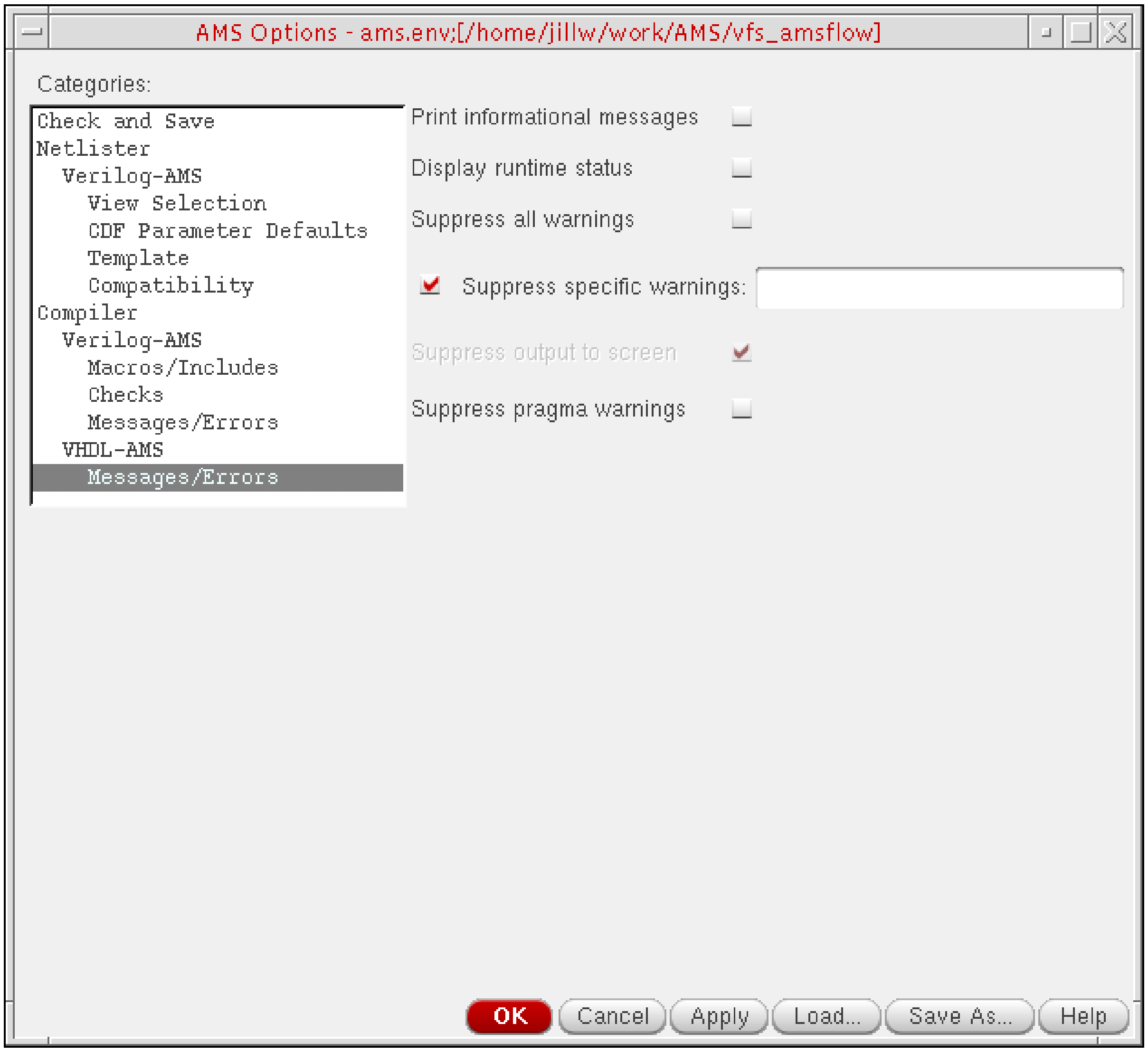
To control the output of VHDL-AMS compiler messages, do the following:

1. In the command interpreter window (CIW), choose *Tools – AMS – Options*.

   The AMS Options form appears.

2. In the *Categories* list, select *Compiler – VHDL-AMS – Messages/Errors*.

   The message options appear.


AMS Options - ams.env;[/home/jillw/work/AMS/vfs_amsflow]

Categories:

```
Check and Save
Netlister
  Verilog-AMS
    View Selection
    CDF Parameter Defaults
    Template
    Compatibility
Compiler
  Verilog-AMS
    Macros/Includes
    Checks
    Messages/Errors
  VHDL-AMS
    Messages/Errors
```

Print informational messages ☐
Display runtime status ☐
Suppress all warnings ☐
✓ Suppress specific warnings:
Suppress output to screen ✓
Suppress pragma warnings ☐

3. Turn on one or more of the following options:

Print informational messages

Prints informational messages as the compiler runs; corresponds to the `ncvhdl -messages` option

Display runtime status

Prints statistics on memory and CPU usage after compilation; corresponds to the `ncvhdl -status` option

Suppress all warnings

Suppresses all warning messages; corresponds to the `ncvhdl -neverwarn` option

Suppress specific warnings

Suppresses warning messages according to the comma- or space-separated list of codes you type in the field; corresponds to the `ncvhdl -nowarn` option

Suppress output to screen

Suppresses output to the screen but does not change what is written to the log file; corresponds to the `ncvhdl -nostdout` option

Suppress pragma warnings

Suppresses warning messages related to pragmas; corresponds to the `ncvhdl -nopragmawarn` option

For information about these `ncvhdl` options, see “ncvhdl Command Options” in “Compiling VHDL Source Files with ncvhdl” in *Cadence NC-VHDL Simulator Help*.

4. Click *OK*.

The VHDL-AMS compiler outputs messages according to your specifications.

C

Updating Legacy SimInfo for Analog Primitives

Cadence® cellview-based netlisters format instances of analog devices according to the instructions specified in the simulation information (simInfo) of the device's CDF. The simInfo is composed of one or more sets of directions, parameters, and terminal names. Each set of simInfo information represents formatting instructions for that device for a given simulator. The information for each simulator is unique. The purpose of the information is the same: To provide the appropriate netlister with the necessary information to netlist a particular primitive device instance for a specific simulator.

For example, if you want to simulate an NMOS device using the Spectre® circuit simulator, you will have information in the `spectre` section of the simInfo. If you also want to simulate that NMOS device using the AMS Designer simulator, you will have AMS-specific information in the `ams` section of the simInfo.

Virtuoso® AMS Designer uses the AMS netlister, which generates Verilog®-AMS netlists targeted for the AMS Designer simulator. To support the AMS simulator, the simInfo for analog primitives contains an `ams` section. This appendix describes the information contained in the `ams` section and describes how the information affects the formatting of primitive devices for Verilog-AMS netlists.

Note: The AMS netlister does not consider whether the view you are netlisting is a stop view before applying the formatting instructions in the `ams` simInfo.

The ams Fields

For the AMS netlister a netlisting procedure that formats instances can be named in the *netlistProcedure* field for a device. In addition, fields provided in the `ams` section provide a mechanism for specializing device instantiation formatting. The retrieval and storage of these fields happens mostly via SKILL, so the values must conform to SKILL types. This appendix describes the fields within the `ams` section of the CDF simInfo and explains how the values within those fields affect the instantiations in the Verilog-AMS netlist.

Some of the fields in the `ams` section require that the values be names that are recognized by the targeted simulator. For example, you might want to specify a delay value for instances of the vpwl device in your design. If you are targeting the Spectre simulator, you specify this value with a parameter called `delay`. In each case, the parameter name is recognized by the targeted simulator. You might use a different parameter name, like `delay1`, in your design for the delay value and then map it to the appropriate simulator name through the *propMapping* field of the `ams` section of the simInfo.

Examples in this appendix are taken from the analogLib library of analog primitives, which is provided in the Cadence hierarchy.

See the following cross-references for information about the fields in the `ams` section.

- otherParameters on page 648
- instParameters on page 649
- enumParameters on page 649
- referenceParameters on page 650
- stringParameters on page 651
- arrayParameters on page 651
- excludeParameters on page 653
- componentName on page 654
- termOrder on page 654
- termMapping on page 655
- propMapping on page 656
- extraTerminals on page 657
- isPrimitive on page 658

otherParameters

This field contains a list of the parameters that the AMS netlister needs to be able to fully process the array parameters specified in the *arrayParameters* field (see "arrayParameters" on page 651 for details about array parameters). For example, if the range of the array in an array parameter is specified with parameters rather than numbers, those parameters must be listed in the *otherParameters* field. Additionally, the generated parameter names that comprise the elements of the array must be specified in this field.

A portion of the *otherParameters* entry for the vpwl device is

```
tvpairs t1 v1 t2 v2 t3 v3 t4 v4 t5 v5 t6
v6 t7 v7 t8 v8 t9 v9 t10 v10 t11 v11 t12
v12 t13 v13 t14 v14 t15 v15 t16 v16 t17 v17
t18 v18 t19 v19 t20 v20 t21 v21 t22 v22 t23
v23 t24 v24 t25 v25 t26 v26 t27 v27 t28 v28
t29 v29 t30 v30 t31 v31 t32 v32 t33 v33 t34
v34 t35 v35 t36 v36 t37 v37 t38 v38 t39 v39
t40 v40 t41 v41 t42 v42 t43 v43 t44 v44 t45
v45 t46 v46 t47 v47 t48 v48 t49 v49 t50 v50
```

This entry instructs the AMS netlister to look for *tvpairs*, which are used to determine the upper range of the array (see "arrayParameters" on page 651) and the list of array elements that are possible in the array.

instParameters

This field contains a list of parameters to be included when writing an instance of the device to the netlist, unless there is no value for the parameter on the instance as a property or in the CDF for the instance master. The parameter names must be names recognized by the targeted simulator. This list of parameters, along with those specified in the other *ams* simInfo **Parameters* fields, are the set of parameters for instances of this device.

The format of the field is

```
instParameters ::= { parameterName }
```

where `parameterName` is a symbol or a string.

A portion of the *instParameters* entry for the vpwl device is

```
dc mag phase xfmag pacmag pacphase delay
```

This entry instructs the AMS netlister to look for values for these parameters and pass them to the instantiation of the vpwl. Note that instance properties and CDF parameters that are not in any of the *ams* simInfo **Parameters* lists are not written for instances of this device.

Note: If the device is a primitive that supports model passing semantics, you must list the associated `model`, `modelname`, or `modelName` parameter in the *instParameters* field. This requirement holds even though `model*` parameters are not passed parameters and AMS Designer treats them in a special way.

enumParameters

This field contains a list of parameters that have meaning for the targeted simulator that handles SPICE and Spectre primitives. For example, consider the `type` parameter of `vsource` in Spectre. The possible values are `dc`, `pulse`, `pwl`, `sine`, or `exp`, which are enumerations. Because Verilog-AMS does not allow enumerated types, the AMS netlister writes the values of these parameters in quotation marks (`""`). The parameter names must be names recognized by the targeted simulator.

The format of the field is

```
enumParameters ::= { parameterName }
```

where `parameterName` is a symbol or a string.

The *enumParameters* entry for the vpwl device is

```
type
```

When these parameters are found on the instance as properties or in the CDF of the device master, they are written as parameters on the instantiation of the device in the netlist, and their values are surrounded by quotation marks (`""`). If the parameters are not found, or are found but do not have values specified, they are not written as parameters. Although these parameters are handled by the AMS netlister in the same manner that *stringParameters* are handled, they have been separated to allow for the possibility of targeting an HDL that does support enumerated types in the future.

referenceParameters

This field contains a list of parameters that have instance names as their values. The parameter names must be names recognized by the targeted simulator. Each parameter is written to the netlist with the value (the name of the instance being referenced) in quotation marks (`""`).

You must identify these parameters for the AMS netlister because sometimes an instance name must be mapped to conform to the requirements of the target language. When this occurs, the value of the reference parameter must be mapped to match the instance name written to the netlist. For example, if a parameter references an instance called `in1` and the module has a net called `in1`, the instance name is mapped by the AMS netlister to `in1_instclash`. The parameter must then have a value of `in1_instclash` rather than `in1`.

The format of the field is

```
referenceParameters ::= { parameterName }
```

where `parameterName` is a symbol or a string.

The *referenceParameters* entry for the cccs device is

```
probe
```

When the AMS netlister encounters a value for this parameter, it verifies that there is an instance in the cellview with a name that matches the value. If it finds such an instance, it writes the name of the instance, in quotation marks, as the value of the parameter. The name of the instance is mapped if necessary.

stringParameters

This field contains a list of parameters to be treated as strings when they are written to the netlist. The AMS netlister writes the values of these parameters in quotation marks (`""`). The parameter names must be names recognized by the targeted simulator.

The format of the field is

```
stringParameters ::= { parameterName }
```

where *`parameterName`* is a symbol or a string.

The *stringParameters* entry for the vpwl device is

```
noisefile fundname
```

When these parameters are found on the instance as properties or in the CDF of the device master, they are written as parameters on the instantiation of the device in the netlist, and their values are enclosed in quotation marks. If the parameters are not found, or are found but do not have values specified, they are not written to the netlist.

arrayParameters

This field specifies parameters that must be written to the netlist as arrays. The data specifies the name of the parameter, the array range, the prefix and suffix to be used for the array elements, and a condition that determines whether the parameter is written to the netlist. The parameter names must be names recognized by the targeted simulator.

- A different prefix and suffix can be specified for each set of elements.
- Omitting the conditional portion of the specification by setting it to `nil` causes the netlister to always write the array parameter to the netlist.

The format of the field is

```
arrayParameters ::=
        nil { paramName arrayDPL }
arrayDPL ::=
        ( nil range ( start stop )
              format ( { ( prefix suffix ) } )
              [condition ( nil [ propname propertyName
                                 value propertyValue ] ) ]
```

paramName	Name of the arrayed parameter (must be a symbol or a string)
start	The beginning of the array index range *start* can be an integer or the name of a property specifying the integer (must be a symbol or a string)
stop	The end of the array index range *stop* can be an integer or the name of a property specifying the integer (must be a symbol or a string). *start* and *stop* cannot both name a property. One of the two must be an integer. Moreover, the value specified in *stop* must be greater than or equal to the value of *start*.
prefix	A string to prepend to the element counter *prefix* can be `nil` if no prefix is needed. If specified, it must be a symbol or a string. *prefix* and *suffix* cannot both be `nil`.
suffix	A string to append to the element counter *suffix* can be `nil` if no suffix is needed. If specified, it must be a symbol or a string. *prefix* and *suffix* cannot both be `nil`.
propertyName	The name of a property whose value is used to determine whether the array is written or not (must be a symbol or a string)
propertyValue	The value that must be matched by *propertyName* if the arrayed parameter is to be written (must be a symbol or a string)

A portion of the *arrayParameters* entry for the `vpwl` device is

```
nil wave (nil range (1 tvpairs) format ((t nil) (v nil)))
```

This entry instructs the AMS netlister to write an array parameter called `wave`. The array is constructed from values of properties `t`*counter* and `v`*counter*, where *counter* ranges from 1 to the value of the `tvpairs` property. Assuming that `tvpairs` is set to 3, the resulting parameter is

```
.wave({ t1, v1, t2, v2, t3, v3 })
```

In this example, names in italics denote the values for those properties. For instance, *`t1`* means the value of property `t1`. If `tvpairs` property is not specified on the instance, the array parameter is not written to the netlist.

In general, if `arrayDPL` is

```
(nil range (1 n) format ( (p1 s1) (p2 s2) ... (pm sm) ) )
```

the arrayed parameter that is written is

```
.name( { p11s1, p21s2, ..., pm1sm,
         p12s1, p22s2, ..., pm2sm,
         ...,
         p1ns1, p2ns2, ..., pmnsm } )
```

The arrayed parameter in this example does not have a conditional clause. However, consider the following example.

```
nil wave (nil range (1 tvpairs) format ((t nil) (v nil))
condition (nil propname printwave value "yes"))
```

For this example, the netlister checks the value of the `printwave` property to see if the value matches the value specified in the condition (`"yes"`). If the values match, the arrayed parameter, `wave`, is written to the netlist just as described above. If they do not match, `wave` is omitted from instances of the device.

The *arrayParameters* field works in conjunction with the *otherParameters* field. The names of properties that are in *arrayParameters*, or can be generated from the *range* and *format* specification, must be listed in the *otherParameters* field. See “otherParameters” on page 648 for details.

excludeParameters

This field contains a list of parameters associated with a particular cell that are to be excluded from netlisting.

You do not need to specify values in the *excludeParameters* field for cells that have valid information in one or more of the *arrayParameters*, *otherParameters*, *instParameters*, *enumParameters*, *stringParameters*, or *referenceParameters* fields because, then, parameters that do not appear in these fields are automatically excluded from netlisting. Avoid specifying the same parameter in both the *excludeParameters* field and in one of the other fields listed above.

The format of the field is

```
excludeParameters ::= { parameterName }
```

You can use the *excludeParameters* simInfo field in conjunction with the `amsExcludeParams` CDF parameter to specify more precisely parameters at the cell, design, and library levels that you do not want to netlist.

componentName

This field, which contains the type of the component being instantiated, overrides the device master cell name. The `model`, `modelname`, or `modelName` parameter also overrides the device master cell name. The precedence for determining the Verilog-AMS module name for the instance is

1. Value of `model` , `modelname`, or `modelName` parameter, if available
2. Value of *componentName* field in the `ams` section of the simInfo, if available
3. Name of device master cell

The format of the field is

```
componentName ::= componentName
```

where *`componentName`* is a symbol or a string.

The *componentName* entry for the `vpwl` device is

```
vsource
```

This entry instructs the AMS netlister to use the name `vsource` rather than the master cell name, `vpwl`, when instantiating the vpwl device.

Note: Because the AMS netlister uses the value you type in this field verbatim (that is, it does not employ any name mapping), you can have a SPICE or Spectre primitive that has the same name as a Verilog-AMS built-in primitive. You must specify formatting instructions in the *termOrder* field.

termOrder

This field contains a list of terminals. The terminals define the port connection for instantiations of the device. The AMS netlister specifies the connections by order, in the order specified, rather than by name. Only the terminals listed in this field are included in the port connection list. If a terminal exists on the instance to which no connection is made, the netlister generates an empty entry in the connection list, ensuring that subsequent connections in the list are made accurately.

If no data is provided in the *termOrder* field, the AMS netlister specifies the port connections for instances of the device by name and in an undetermined order. If a terminal exists on the instance to which no connection is made, that port is omitted from the connection list.

The format of the field is

```
termOrder ::= { terminalName }
```

where `terminalName` is a symbol or a string.

The *termOrder* entry for the nmos device is

```
D G S B
```

This entry instructs the AMS netlister to include four entries in the port connection list when instantiating an nmos device. The port connections appear in the order specified.

The *termOrder* field can specify more terminals than are present in the symbol view. In such cases, the *extraTerminals* field must be used. The AMS netlister issues an error for a `terminalName` that exists in *termOrder* but does not exist in either the placed master or in the *extraTerminals* field.

Using the example above, assume that the signals `sig1` and `sig2` are connected to terminals `D` and `S`, respectively, and that the *extraTerminals* field specifies that `B` be connected to `gnd!`. Further assume that the `G` terminal is unconnected. The port connection list for the instance is written to the netlist as

```
(sig1,,sig2,cds_globals.\gnd! )
```

If the nmos device did not have a *termOrder* entry, the port connection list for the instance might be written to the netlist as

```
(.S(sig2),.D(sig1),.B(cds_globals.\gnd! ),.G())
```

Programmable nodes that are specified in the *termOrder* entry as an argument to `Progn` (i.e., `Progn (nodeName)`) are not supported by the AMS netlister. Programmable nodes must be represented as *extraTerminals* in the *ams* simInfo. If `Progn` is encountered by the conversion program, the program issues a warning that manual editing of the *ams* simInfo is necessary. For information about the conversion program, see "Converting an Existing Analog Primitive Library" on page 660.

termMapping

This field specifies the mapping between the terminal names used in the cell schematic and the terminal names used by the simulator, when those names are different.

The format of the field is

```
termMapping ::= nil { symbolTermName simTermName }
```

where *symbolTermName* is a terminal name on the symbol and *simTermName* is the corresponding terminal name used by the simulator. Because the value of the field is implemented as a DPL, it must always begin with `nil`.

For example, in the `nbsim` cell in `analogLib`, the terminals names `D`, `G`, and `S` are specified in uppercase. Although uppercase terminal names are appropriate for SPICE, the AMS simulator expects the terminal names `d`, `g`, and `s` in lowercase. To map between the two sets of terminal names, the following code appears in the *termMapping* field for the AMS simulator:

```
nil D \:d G \:g S \:s B \:b
```

Note: Make sure all non-alphanumeric characters are preceded by a backslash.

propMapping

This field specifies a one-to-one mapping between a given simulator property name and a corresponding CDF property name. This allows you to place values on instances using one set of property names and have the netlisters use the property names that are specific to the simulators they are targeting.

The format of the field is

```
propMapping ::= nil { simParamName cdfParamName }
```

where both *simParamName* and *cdfParamName* are symbols or strings.

A portion of the *propMapping* entry for the vpwl device is

```
nil dc vdc mag acm phase acp delay td
```

Because the value of the field is implemented as a DPL, it must always begin with `nil`.

The entry above instructs the AMS netlister to replace parameter names when writing instances of the vpwl as follows:

When the parameter `vdc` is found...	Use `dc` as the parameter name.
When the parameter `acm` is found...	Use `mag` as the parameter name.
When the parameter `acp` is found...	Use `phase` as the parameter name.
When the parameter `td` is found...	Use `delay` as the parameter name.

extraTerminals

This field contains information for writing inherited connection terminals on instances. This is necessary when the simulator view of an instance contains more terminals than are present on the symbol view. An example is the *B* terminal of *nmos* in analogLib. The symbol view of *nmos* contains only three terminals. The spectre view contains a fourth terminal, with a net expression on the fourth terminal. This fourth terminal is a programmable node to which a connection is made through a property specification rather than through wiring to the pin.

Because the AMS netlister is a single cellview netlister and does not read any views other than the one it is netlisting, information such as the net expression in the spectre view must be specified in the *extraTerminals* field.

The format of the field is

```
extraTerminals ::=
        { ( nil name termName
                direction directionType
                netExpr netExpression ) }
```

termName	The name of the terminal (must be a string)
directionType	The direction type of the terminal as specified in `cv~>terminals~>direction` *directionType* can be `"input", "output"`, or `"inputOutput"`
netExpression	The net expression that specifies what the connection to the terminal should be (must be a string)

The *extraTerminals* entry for the nmos device is

```
(nil name "B" direction "inputOutput" netExpr "[@bulk_n:%:gnd!]")
```

This entry instructs the AMS netlister to create a connection for a terminal `B` in the instance connection port list for all instances of the nmos device. The terminal is considered to be an input/output terminal. The netlist expression indicates that a property called `bulk_n` is to be consulted for the name of the net to which terminal `B` is to be connected. In addition, if `bulk_n` is not found, the `gnd!` net is to be used.

For details on net expressions, see the *Virtuoso Schematic Editor User Guide*.

Each terminal specified in the *extraTerminals* field is enclosed in a set of parenthesis and each *termName* specified must also appear in the *termOrder* field. See “termOrder” on page 654 for details.

isPrimitive

This field tells the AMS netlister that the device is an analog primitive.

The format of the field is

```
isPrimitive ::= t | nil
```

Using the value `t` for this field indicates that the component is a primitive. A primitive is an analog component that is understood directly by the analog solver.

This field instructs the AMS netlister how to handle the model parameters: `model`, `modelName`, and `modelname`. If the *isPrimitive* field is set to `t` and if any of the model parameters are listed in the AMS simulation information for this device, then the AMS netlister treats the device as a primitive and supports model semantics for the AMS simulator. In this case, the values of model parameters found on instances of this device are used as cellnames (or `analogmodel` instantiations) when the instances are added to the netlist. For more information, see "Special Handling of model, modelName, modelname, and componentName" on page 660.

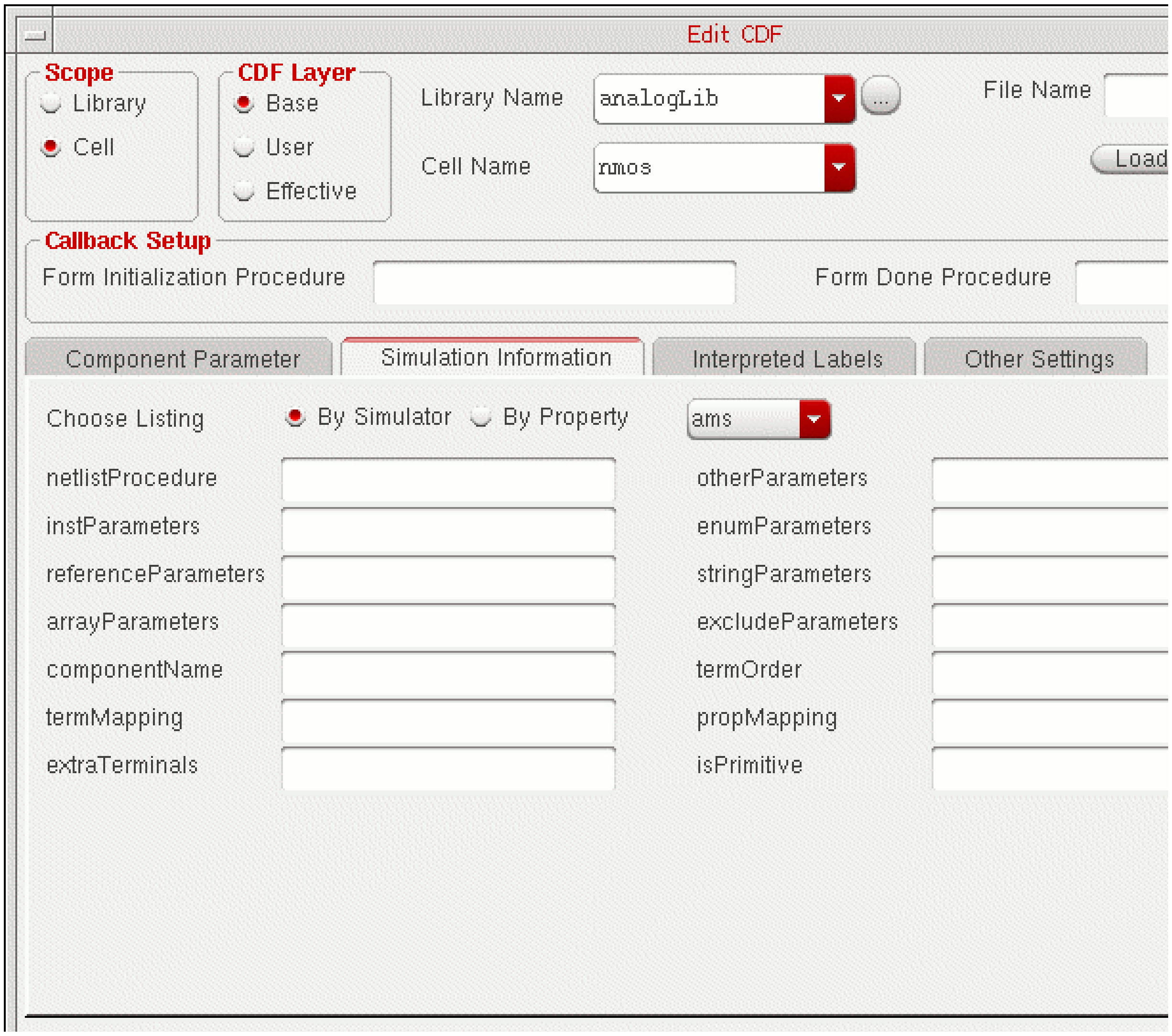
To add a `t` to the `isPrimitive` field for a cell,

1. From the CIW, choose *Tools – CDF – Edit*.

   The Edit Component CDF window opens.

2. Fill in the library and cell information for the cell to be marked as a primitive.
3. Set *CDF Type* to `Base`.
4. Click the *Edit* button in the *Simulation Information* part of the form.

The *Edit Simulation Information* form appears.



Edit CDF

Scope: Library [✓ Cell]

CDF Layer: [✓ Base] User Effective

Library Name	analogLib	File Name	
Cell Name	nmos		

Callback Setup

Form Initialization Procedure		Form Done Procedure	

Choose Listing [✓ By Simulator] By Property ams

netlistProcedure		otherParameters	
instParameters		enumParameters	
referenceParameters		stringParameters	
arrayParameters		excludeParameters	
componentName		termOrder	
termMapping		propMapping	
extraTerminals		isPrimitive	

5. Select *ams* in the *Choose Simulator* cyclic field.
6. Type a `t` in the `isPrimitive` field.
7. Click *OK*.
8. Click *OK* in the Edit Component CDF window.

Special Handling of model, modelName, modelname, and componentName

Although you can specify the `model`, `modelname`, `modelName`, and *componentName* parameters in the *ams* simInfo **Parameters* fields, they are not written to the Verilog-AMS netlists. Instead, they change the module name of the instance to which they apply.

Another important difference in the treatment of `model`, `modelname`, `modelName`, and *componentName* is the mapping of the name. Names specified as values of the `model`, `modelname`, and `modelName` parameters are checked for legality in the Verilog-AMS language. If necessary, these names are escaped. For example, if an instance has the property `model=4nmos`, `4nmos` is mapped to Verilog-AMS as `\4nmos`, because Verilog-AMS identifiers cannot begin with a digit.

Note: If the device is a primitive that supports model passing semantics, the associated *model*, *modelName*, or *modelname* parameter must be listed in the *instParameters* field. This requirement holds even though *model*, *modelName*, and *modelname* parameters are not passed parameters and are treated specially.

Names specified as values of the *ams* simInfo *componentName* field are not mapped. They are written to the netlist verbatim. This behavior supports the handling of SPICE and Spectre primitives that have the same names as Verilog-AMS built-in primitives. SPICE and Spectre primitives such as these can be supported by providing the complete set of formatting instructions via the *termOrder* field.

For additional information about these parameters, see "Netlisting Model Names from Parameter Values" on page 216.

Converting an Existing Analog Primitive Library

Cadence provides a conversion utility that you can use to add *ams* simInfo to an existing analog primitive library. The utility derives the new *ams* simInfo from existing *spectre* simInfo data.

Note: If you do not have existing *spectre* simInfo, you can

- ❑ Use another conversion program to prepare the primitives for direct simulation with the Spectre circuit simulator and then convert the *spectre* simInfo to *ams* simInfo
- ❑ Create *ams* simInfo for the primitives according to the definitions in this appendix, then use the steps in "Placing SPICE and Spectre Design Units on a Schematic" on page 269 to edit the CDF simulation information

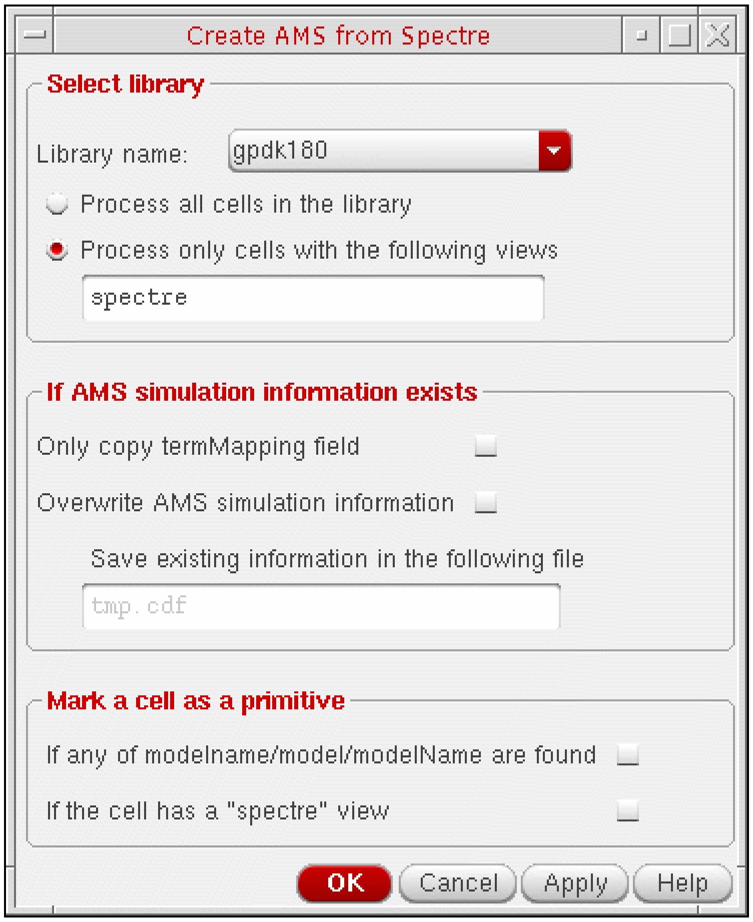
To create *ams* simInfo from existing *spectre* simInfo, do the following:

1. In the command interpreter window (CIW), choose *Tools – Conversion Tool Box*.

   The Conversion Tool Box appears.

2. Click *AMS simInfo from Spectre*.

   The Create AMS from Spectre form appears.



Create AMS from Spectre

Select library

Library name: gpdk180

○ Process all cells in the library

● Process only cells with the following views

spectre

If AMS simulation information exists

Only copy termMapping field ☐

Overwrite AMS simulation information ☐

Save existing information in the following file

tmp.cdf

Mark a cell as a primitive

If any of modelname/model/modelName are found ☐

If the cell has a "spectre" view ☐

3. From the *Library name* cyclic field, select the analog primitive library that you want to update.

4. Choose one of the following radio buttons:

   ❑ *Process all cells in the library*

   ❑ *Process only cells with the following views*

   If you choose this option, type one or more view names in the corresponding field. The utility processes only those cells that have at least one of the specified views.

The utility does not process any cells that have none of the specified views and does not do anything with any existing AMS simulation information.

5. (Optional) If you are updating libraries that have existing *ams* simInfo that you do not want the program to alter, turn on *Only copy termMapping field*.
6. (Optional) If the library already has *ams* simInfo data, do the following:
   a. Turn on *Overwrite AMS simulation information*.

      The *Save existing information in the following file* field becomes active.
   b. In the *Save existing information in the following file* field, type the name of a backup file for the conversion utility to create.

      If necessary, you can use the backup file to restore the original *ams* simInfo.
7. (Optional) Specify conditions for the conversion utility to use when determining whether a cell is a primitive.
   a. Turn on *If any of modelname/model/modelName are found* if you want the netlister to use this condition for determining whether a cell is a primitive.
   b. Turn on *If the cell has a "spectre" view* to have the netlister use this condition.

   The following conditions often indicate that a cell is a primitive and ought to be marked as such by having the *isPrimitive* field of the `ams` section of the CDF simInfo set to `t`. However, this automatic determination is not always correct, so Cadence recommends that you check the accuracy. You might need to set *isPrimitive* manually to the opposite value.
8. Click *OK*.

The *AMS simInfo from Spectre* conversion traverses the library, reporting the cells for which it creates *ams* simInfo data and those which it skips. When it skips a cell, the *AMS simInfo from Spectre* conversion reports the reason.

The conversion program first copies data from the fields of the *spectre* simInfo that are common to the *ams* simInfo:

- *otherParameters*
- *instParameters*
- *componentName*
- *termOrder*
- *propMapping*

It then examines the *netlistProcedure* field in the *spectre* simInfo data. The conversion program recognizes and converts the following analogLib procedures:

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

The conversion program segregates parameters referenced by these procedures into the various *ams* fields: *instParameters*, *enumParameters*, *referenceParameters*, *stringParameters*, and so on. If there is no *netlistProcedure* for a cell or if the *netlistProcedure* is not one of those listed above, the conversion program leaves the copied *spectre* simInfo data in the corresponding common *ams* simInfo fields. In such cases, you must manually edit the information to ensure that the parameters are in the correct fields.

The *AMS simInfo from Spectre* conversion program can fill in the `isPrimitive` field of the AMS simulation information according to rules you provide. However, because these rules can sometimes produce an incorrect setting of the field, be sure to validate the results.

Finally, if the cell has a spectre view, the conversion program examines it to determine if it has more terminals than the symbol view. If so, an entry is made in the *extraTerminals* field of the *ams* simInfo that represents the additional terminals.

To manually edit *ams* simInfo data,

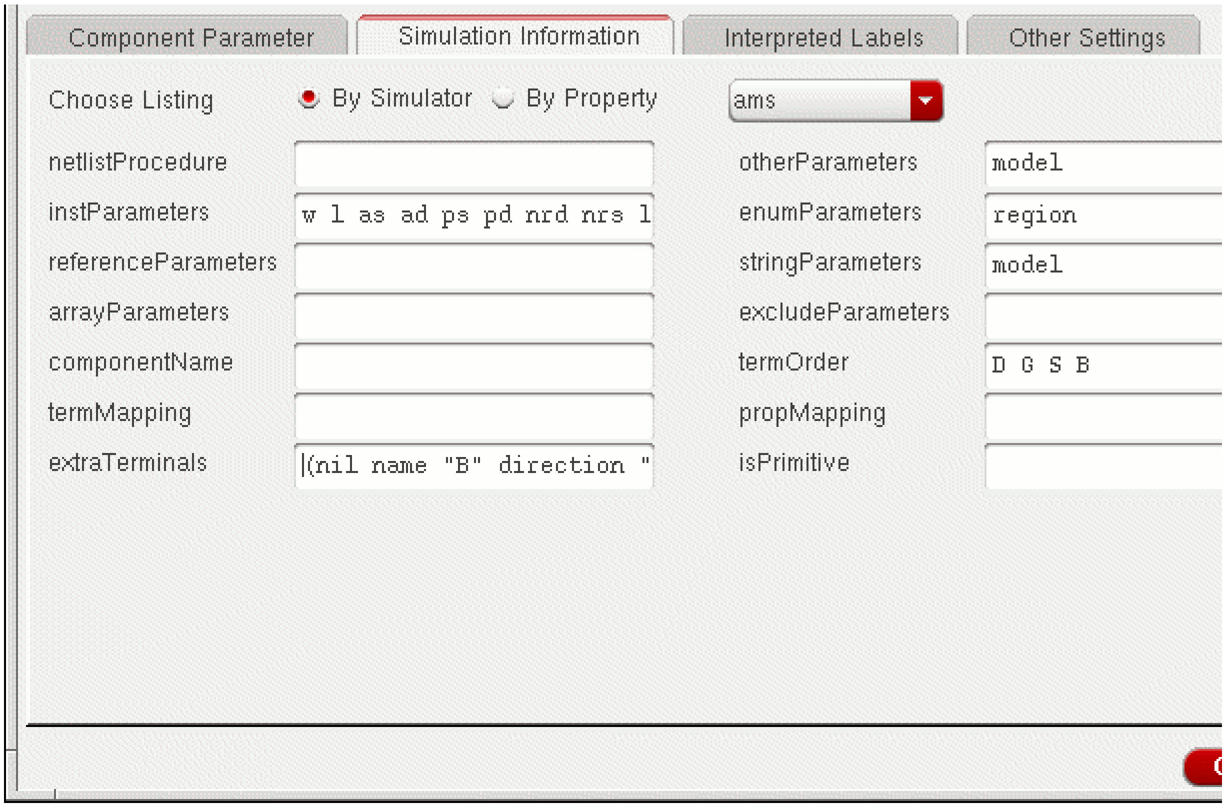
1. From the CIW, choose *Tools – CDF – Edit*.

   The Edit Component CDF window opens.

2. Fill in the library and cell information.
3. Set *CDF Type* to `Base`.

   Except for the `amsExcludeParams` parameter, the AMS Designer environment does not use *Library* or *User* CDF information.
4. Click the *Edit* button in the *Simulation Information* part of the form.

   The *Edit Simulation Information* form appears.



Choose Listing: ● By Simulator ○ By Property — ams

netlistProcedure		otherParameters	model
instParameters	w l as ad ps pd nrd nrs l	enumParameters	region
referenceParameters		stringParameters	model
arrayParameters		excludeParameters	
componentName		termOrder	D G S B
termMapping		propMapping	
extraTerminals	|(nil name "B" direction "	isPrimitive	

If *spectre* simInfo data existed when the conversion program ran and the *netlistProcedure* field was empty or contained a procedure other than those listed above, the *ams* fields that correspond to *spectre* simInfo fields contain data copied directly.

5. Examine the *instParameters* and *otherParameters* fields for parameters that should be moved to the *arrayParameters, enumParameters, stringParameters,* or *referenceParameters* fields and make the appropriate changes.

D

Designing for Virtuoso AMS Compliance

This appendix describes guidelines for creating schematic designs that the Virtuoso® AMS Designer environment can handle efficiently. The consequences of not complying with these guidelines vary. In some cases, not complying with a guideline results in a netlist that is less readable or in the failure of downstream processes, such as cross-probing. In other cases, not complying with a guideline causes netlisting to fail.

The guidelines presented here are arranged in the following categories.

Identifiers

So that your design works efficiently with the Virtuoso AMS Designer environment, ensure that the identifiers you use

- Follow the recommended syntax
- Map cleanly to the netlist languages you plan to use
- Are unique within your design

These guidelines are discussed in greater detail in the following sections.

Follow the Recommended Syntax for Identifiers

Use the following syntax for the `basic_identifier` when you create identifiers.

```
basic_identifier ::=
        letter {[_] letter_or_digit}

letter_or_digit ::=
        letter
    |   digit

letter ::=
        a-z

digit ::=
        0-9
```

For example, the following identifiers comply with this syntax.

```
an_identifier_name
a_2nd_name
a_name2
```

However, the following identifiers do *not* comply with the syntax.

```
2identifier      // Should begin with a letter.
My_identifer     // Should not use uppercase letters.
an_identifier_   // Should end with a letter or digit.
an__identifier   // Should not use multiple adjacent underscores.
```

Ensure that Identifiers Map Cleanly to Netlist Languages

In addition to complying with the `basic_identifier` syntax, your identifiers should also map cleanly to the netlist languages that you plan to use. (In this release, the only supported netlist language is Verilog®-AMS.) To meet the goal of mapping cleanly, follow these additional guidelines.

- Use only characters that are legal in the netlist languages you plan to use.

  The AMS Designer environment escapes illegal characters, resulting in a less readable netlist. For example, the identifier `a&b` appears in the netlist as `\a&b` followed by a space if your netlist language is Verilog-AMS; it appears as `\a&b\` if your netlist language is VHDL-AMS.

- Do not use names that are reserved words in the netlist languages you plan to use.

  The AMS Designer environment escapes any reserved word used as an identifier, resulting in a less readable netlist. For example, when used as an identifier, the reserved word `nature` appears in the netlist as `\nature` followed by a space if your netlist language is Verilog-AMS; it appears as `\nature\` if your netlist language is VHDL-AMS.

Ensure That Identifiers Are Unique within Your Design

Ensure that every one of the instances, cells, terminals, parameters, and nets in your design has a unique identifier. As noted in the following tables, the consequences of not complying with this guideline vary from netlist failure to reduced cross-probing capabilities because the name in the netlist no longer matches the name in the schematic. The consequences also depend on the netlist language that you use.

VHDL-AMS: Handling of Non-Unique Identifiers

When these objects share a name	Then
terminal, cell	Netlisting fails
parameter, terminal	Netlisting fails
parameter, cell	Netlisting fails
net, parameter	Net identifier maps to *`netName`*`_netclash`
net, terminal	Net identifier maps to *`netName`*`_netclash`. (However, no mapping occurs when the net and terminal are connected to each other.)
net, cell	Net identifier maps to *`netName`*`_netclash`
instance, net	Instance identifier maps to *`instName`*`_instclash`

VHDL-AMS: Handling of Non-Unique Identifiers, *continued*

When these objects share a name	Then
instance, parameter	Instance identifier maps to `*instName*_instclash`
instance, terminal	Instance identifier maps to `*instName*_instclash`
instance, cell	Instance identifier maps to `*instName*_instclash`

Verilog-AMS: Handling of Non-Unique Identifiers

When these objects share a name	Then
terminal, cell	No mapping occurs and netlisting proceeds normally
parameter, terminal	Netlisting fails
parameter, cell	No mapping occurs and netlisting proceeds normally
net, parameter	Net identifier maps to `*netName*_netclash`
net, terminal	Net identifier maps to `*netName*_netclash`. (However, no mapping occurs when the net and terminal are connected to each other.)
net, cell	Net identifier maps to `*netName*_netclash`
instance, net	Instance identifier maps to `*instName*_instclash`
instance, parameter	Instance identifier maps to `*instName*_instclash`
instance, terminal	Instance identifier maps to `*instName*_instclash`
instance, cell	Instance identifier maps to `*instName*_instclash`

Terminals

Your designs should comply with the following guidelines for terminals.

■ Every cellview of a cell should use the same set of terminals.

Following this general guideline facilitates cellview switching. However, the minimum requirement is that at least every connected terminal in a symbol must be defined in the switched view. In the switched view, you can have additional defined terminals that do not appear in the symbol view.

- For the VHDL-AMS netlist language, each terminal identifier should match the identifier of the external net to which the terminal connects.

  If your design does not comply with this guideline, the netlister attempts to declare an alias of the terminal, where the alias has the name of the terminal.

  In some cases, it is not possible to use an alias, such as when only a part of a bus is connected to a terminal or when a bus is connected to multiple terminals. In situations like this, the netlister attempts to create a VHDL block and to resolve the connection by using block port maps. The netlister warns you when it creates a VHDL block because all cross-probing capabilities inside the block are lost.

Buses

To ensure that your design can proceed smoothly through the steps in the Virtuoso AMS Designer environment flow, follow these guidelines dealing with buses.

- Use simple buses when you declare vector terminals or nets.

  Avoiding the use of concatenated non-consecutive bits, ranges with increment values other than one, prefix repeat operators, and suffix repeat operators is especially important when declaring terminals. For example, you have a terminal on a schematic with the identifier `<*2>term`, connected to a 2-bit wide net. The netlister, however, writes the identifier as a single-bit port called `term`. Attempting to connect the single-bit port described in the netlist to the 2-bit wide net results in failure.

  For nets, an identifier like `net1,net1` is written to the netlist as a concatenation, in this example, `{net1,net1}`. You can successfully simulate with this concatenation, but you lose the ability to cross-probe the net.

- Use a consistent range direction when declaring and using each bus. Choose either MSB:LSB or LSB:MSB.

  Using a bus or a subsection of a bus with a range direction different from the declared range direction for that bus forces the netlister to write the bus instance as a concatenation of bits. Because the concatenation does not match the original declaration, you lose the ability to cross-probe the net that includes the bus.

- Do not declare sparse buses.

  Using sparse buses hinders or prevents cross-probing. For example, you declare a bus in the schematic as `busname<15:0:2>`, which is an 8-bit net. Because sparse buses are overdeclared, the netlister writes the bus to the netlist as `busname[15:0]`, which is a 16-bit net. As a result, the connections have to be written as a concatenation of the eight odd-numbered bits of `busname`. This concatenation does not match the original declaration of `busname<15:0:2>`, so you lose the ability to cross-probe the net that includes the bus.

Component Description Format

Do not use library component description format (CDF) information. In the AMS Designer environment, library CDF information has no effect on netlisting.

(You can, however, use cell CDF. For information, see "Using Cell Parameters" on page 673.)

Parameters

This section describes the guidelines for using inherited parameters, cell parameters, and parameter formats in the AMS Designer environment.

Using Inherited Parameters

If your design uses inherited parameters, comply with the following guidelines to ensure that you get the results you expect. (For background information, see the *Analog Expression Language Reference Manual* and the *Component Description Format User Guide*.)

- Do not use atPar (`[@`) or dotPar (`[.`) expressions. If you ignore this guideline, atPar expressions are interpreted as pPar expressions, and dotPar expressions are interpreted as iPar expressions.
- Ensure that the parameters for any iPar (`[~`) expressions are defined, with defaults, in the CDF for the master of the instance. Ensure that the parameters for any pPar (`[+`) expressions are defined, with defaults, in the CDF for the instantiating cell.
- Define all parameters used as the argument of a pPar expression in the CDF.

  If a parameter that is used as the argument of a pPar expression on an instance is not declared in the CDF, the parameter statement is written near the instantiation statement in the netlist file rather than at the top of the module. This behavior makes the netlist less readable.
- Do not use the `{` *`param`* `}` expression to pass parameters between levels of hierarchy. This form is not supported by the AMS Designer environment. Parameters can be passed from one level of the design hierarchy to a lower level by using pPar expressions.

Using Cell Parameters

Follow these guidelines for cell parameters as you develop your design.

- Ensure that numerical parameters are defined with appropriate types, such as `int` or `float`.

  Pay particular attention to this guideline if your design was originally created for the Spectre simulator. Sometimes numerical parameters created for the Spectre simulator are defined as strings (because that is the default parameter type). Such parameters are successfully interpreted by the Spectre simulator as numerical values.

  However, the AMS netlister interprets such parameters as they are defined, as strings, even though the design requires numerical values. This conflict in data types leads to unpredictable type translations and incorrect simulation results.

  For additional information, see “Forcing Schematic Parameter Values to Netlist as Floating Point Values” on page 219.

- Ensure that cell parameters are defined in the CDF and that they have defined default values.

  If cell parameters are not defined in the CDF, the netlister looks in environment files for a specified value. If no value is found, the netlister uses 0 as the default value for integer type parameters and uses 0.0 as the default value for all other parameters, including string parameters.

- If you develop a Verilog® cellview outside of the AMS Designer environment and then use the cellview within the environment, be sure that the default parameter values in the original Verilog view are the same as the default values for the same parameters in the cell CDF. Following this guideline ensures that the values in the original Verilog cellview are consistent with the values used in cellviews generated by the AMS Designer environment.

Using Efficient Formats for Parameter Values

You can speed up netlisting by entering parameter values in the format that you want them to be used in the netlist. For example, if you want the value `5.46u` to appear as `0.00000546` in the netlist, use the expanded form to define the parameter.

This guideline is especially pertinent for the VHDL-AMS netlist language, which does not support the use of scaling factors.

Parameterized Cells

In the AMS Designer environment, follow this restriction on using parameterized cells.

In schematic and layout views, do not use parameterized cells that change internal connectivity. For example, do not use parameterized cells that change the number or width of terminals or instances. You can, however, use parameterized cells to vary shapes and sizes, such as the width or shape of a transistor.

Be sure that you define the parameters used in parameterized cells either in the cell CDF or as instance properties.

VHDL-AMS Component Declarations

You can simplify and speed up netlisting by using only one kind of component declaration methodology within your design. Mixing the package and inline methodologies within a design can force the AMS Designer environment to reanalyze and renetlist cells that use inconsistent methodologies.

Properties

Your designs function most efficiently in the AMS Designer environment if you use properties according to the guidelines in the following sections.

Properties to Avoid Completely

The properties listed in this section are not supported in the AMS Designer environment and should not be used. If you use these properties, the AMS Designer environment omits them from the netlist. This process increases the run time.

Property to avoid	Other Cadence netlisters that use the property
`hnlVerilogCDFdefparamList`	Verilog
`hnlVerilogHandleRCdata`	Verilog
`verilogFormatProc`	Verilog
`verilogView`	Verilog

Property to avoid	Other Cadence netlisters that use the property
`vhdlArchitectureName`	VHDL
`vhdlNetlistType`	VHDL
`vhdlPortType`	VHDL
`vhdlSignalKind`	VHDL

Avoid the portOrder Property Unless Required by Special Circumstances

Unless you need the `portOrder` property to ensure that ports and buses are netlisted in a particular way, Cadence recommends not using the property.

Property to avoid unless required	Other Cadence netlisters that use the property
`portOrder`	VHDL and Verilog

Properties to Use Only in AMS Compatibility Mode

The properties listed in this section are supported only when you use the AMS compatibility mode. The AMS compatibility mode facilitates migration by instructing the netlister to support some of the properties used by other Verilog and VHDL netlisters.

Using the following properties when the AMS compatibility mode is not in effect slows down processing by the amount of time required to filter the properties out of the netlist.

Property to use only in AMS compatibility mode	Other Cadence netlisters that use the property
`vhdlAttributeDefList`	VHDL
`vhdlComponentDecl`	VHDL
`vhdlFormalPortFuncName`	VHDL
`vhdlPackageComponents`	VHDL
`vhdlPackageNames`	VHDL

Properties That Have No Special Meaning in the AMS Designer Environment

The properties in this section have meaning for some netlisters, but have no special meaning in the AMS Designer environment.

Property	**Other Cadence netlisters that use the property**
`algorithm`	Verilog
`c`	Verilog
`chargeDecay`	Verilog
`chargeStrength`	Verilog
`driveStrength`	Verilog
`High_Strength`	Verilog
`highThreshold`	Verilog
`length`	Verilog
`Low_Strength`	Verilog
`lowThreshold`	Verilog
`td`	Verilog
`technology`	Verilog
`tf`	Verilog
`tr`	Verilog
`tz`	Verilog
`width`	Verilog

Properties Fully Supported by the AMS Designer Environment

The properties in this section are fully supported in both the compatibility mode and the non-compatibility mode.

Property	**Other Cadence netlisters that use the property**
`hnlVerilogCellAuxData`	Verilog
`modelName`	Verilog
`netType`	Verilog
`nlAction="ignore"`	VHDL and Verilog
`vhdlActualPortFuncName`	VHDL
`vhdlComment`	VHDL
`vhdlDataType`	VHDL
`vhdlGenericDefList`	VHDL
`vhdlInitialValue`	VHDL
`vhdlResolveFunction`	VHDL
`vhdlScalarType`	VHDL
`vhdlVectorType`	VHDL

E

Customization Variables

This appendix describes the Cadence customization variables associated with the AMS Designer environment.

Customization Variables

You can use variables to customize the operation of the AMS Designer environment. For example, you can put these variables in a `.cdsinit` file or set their values in the CIW. The variables apply when you do one of the following:

- Type in or edit a Verilog-AMS cellview
- Create a Verilog-AMS cellview from another cell using one of the *Design – Create Cellview* commands from the schematic or symbol editor
- Create another cellview from a Verilog-AMS cellview
- Call the function `vmsUpdateCellViews`

For details, see the cross-references below.

Variable	**For information, see**
`schHdlNotCreateDB`	schHdlNotCreateDB on page 681
`schHdlParseUsingNcvhdl`	"schHdlParseUsingNcvhdl" in Appendix A of the "Virtuoso Schematic Editor VHDL Interface User Guide"
`schHdlUseVamsForVerilog`	schHdlUseVamsForVerilog on page 682
`vhdlCrossViewCheck`	vhdlCrossViewCheck on page 682
`vhdlKeepCaseAsNC`	vhdlKeepCaseAsNC on page 683
`vhdlUpdateSymbol`	vhdlUpdateSymbol on page 684

Variable	**For information, see**
`vmsAnalysisType`	vmsAnalysisType on page 685
`vmsCreateMissingMasters`	vmsCreateMissingMasters on page 686
`vmsCrossViewCheck`	vmsCrossViewCheck on page 686
`vmsDoNotCheckMasterFileWritable`	vmsDoNotCheckMasterFileWritable on page 687
`vmsNcvlogExecutable`	vmsNcvlogExecutable on page 688
`vmsPortProcessing`	vmsPortProcessing on page 689
`vmsRunningInUI`	vmsRunningInUI on page 690
`vmsTemplateScript`	vmsTemplateScript on page 691
`vmsUpdateSymbolAfterEdit`	vmsUpdateSymbolAfterEdit on page 692
`vmsVerboseMsgLevel`	vmsVerboseMsgLevel on page 693

schHdlNotCreateDB

Specifies a list of HDL view types for which database data is not to be created.

```
schHdlNotCreateDB_variable ::=
        schHdlNotCreateDB = '( { "view_type" } ) | nil
```

The parameters are the following:

view_type	The environment does not create database data for these HDL view types. In this release, the only values supported for *view_type* are `VerilogAMSText` and, when `schHdlUseVamsForVerilog` is set to `t`, `VerilogText`.
`nil`	The environment creates database data for all HDL view types. This is the default.

This variable allows you to control whether database data is created for specified view types. You might want to turn off database data creation to avoid creating library bindings in the database that prevent you from using the library list in the Virtuoso® Hierarchy Editor.

Example 1

You use the following command in the CIW to identify the existing view types.

```
ddMapGetDataTypeList()
```

The returned information looks similar to

```
("AHDLNetlist" "AHDLText" "AsciiText" "CDBAGraphics" "CDBANetlist"
    "CDBAStranger" "ComposerSchematic" "ComposerSymbol"
    "HierarchyDescription" "MaskLayout" "MaskLayoutGNS"
    "SPECTRENetlist" "SPECTREText" "VERILOGANetlist"
    "VERILOGAText" "VHDLText" "VerilogAMSNetlist"
    "VerilogAMSText" "VerilogNetlist" "VerilogText"
    "VirtuosoSimView" "dfIICategoryFiles" "dfIIPropXxFiles"
)
```

Having confirmed that one of the view types is `VerilogAMSText`, you specify that database data is not to be created for that type.

```
schHdlNotCreateDB = '("VerilogAMSText")
```

Example 2

The following example specifies that database data is to be created for every view type.

```
schHdlNotCreateDB = nil
```

schHdlUseVamsForVerilog

Controls, from the command interpreter window (CIW), the compilation of Verilog (digital-only) text views.

```
schHdlUseVamsForVerilog ::=
        schHdlUseVamsForVerilog = t | nil
```

The parameters are the following:

`t`	The syntax of a Verilog text view is checked by the AMS compiler, which generates a Verilog syntax tree (VST) for the view. All SKILL variables applicable for Verilog-AMS text processing are also active for Verilog text processing.
`nil`	The syntax of a Verilog text view is checked by the Verilog Analyzer (VAN), which does not generate a VST for the view. This is the default value.

vhdlCrossViewCheck

Controls whether symbol views are checked for consistency with other views in a cell when the `vmsUpdateCellViews` function is run.

The parameters are the following:

`t`	Checks whether the symbol views that are created are consistent with all other views in a cell.
`nil`	Disables the consistency check.

vhdlKeepCaseAsNC

By default, names of VHDL identifiers (such as entity, port and architecture names) are lower cased when the `vmsUpdateCellViews` function is run. Use this variable to preserve the case of entity and port names when the `vmsUpdateCellViews` function is run.

Note the following:

- Architecture names are always lowercased.
- Cadence recommends that you do not use the following environment variable to preserve the case of entity and port names:

  ```
  CDS_ALT_NMP=match
  ```

For example, consider the following VHDL entity:

```
entity myEntity is
port(
VIn : In bit
Vout : out bit
);
```

When the `vmsUpdateCellViews` function is run, by default, the symbol view contains lower cased port names `vin` and `vout`. The entity name is also converted to lowercase and saved as `myentity`. Note that, in this case, the original VHDL text view does not get modified. Instead, a new VHDL entity view named `myentity`, is created.

The parameters are the following:

`t`	Case of entity and port names are preserved. For escaped names, case is preserved for all identifiers. **Note:** If `t`, you must also add the following entry in the hdl.var file: `define ncvhdlopts -keepcase4use5x`
`nil`	Names of VHDL identifiers are lowercased.

vhdlUpdateSymbol

Controls whether symbol views are automatically created for cells that don't have a symbol view when the `vmsUpdateCellViews` function is run.

The parameters are the following:

`t`	The symbol view is automatically created.
`nil`	Disables creation of symbol views.
`query`	Displays a pop-up asking for confirmation whether the symbol view should be created.

vmsAnalysisType

Specifies the kind of syntax checks to be applied to text views.

```
vmsAnalysisType_variable ::=
        vmsAnalysisType = "Analog" | "Digital" | "Mixed"
```

The parameters are the following:

`Analog`	The syntax in text views is checked for compliance with the Verilog-A language specification.
`Digital`	The syntax in text views is checked for compliance with the Verilog (digital-only) language specification. This is the default value for verilog text views.
`Mixed`	The syntax in text views is checked for compliance with the Verilog-AMS language specification. This is the default value for verilog-ams text views.

vmsCreateMissingMasters

Specifies whether the environment is to create skeleton descriptions for undefined modules.

```
vmsCreateMissingMasters_variable ::=
        vmsCreateMissingMasters = t | nil
```

The parameters are the following:

`t`	The environment creates skeleton Verilog-AMS descriptions and symbols for undefined modules by using explicit port names (when the instances are connected explicitly) or by using connecting module port names (when the instances are connected implicitly). If these approaches fail, the environment uses dummy names for ports. The direction assigned to each port is based on the direction of the connecting net, if a direction is set.
`nil`	The environment does not create skeleton descriptions or symbols for undefined modules. This is the default value.

vmsCrossViewCheck

Controls whether symbol views are checked for consistency with other views in a cell when the `vmsUpdateCellViews` function is run.

The parameters are the following:

`t`	Checks whether the symbol views that are created are consistent with all other views in a cell.
`nil`	Disables the consistency check.

Default value: `t`

vmsDoNotCheckMasterFileWritable

Specifies whether the program should check the master file for write privileges before performing the vmsUpdateCellViews function.

```
vmsDoNotCheckMasterFileWritable_variable ::=
        vmsDoNotCheckMasterFileWritable = t | nil
```

The parameters are the following:

`t`	In order to update cellviews using the vmsUpdateCellViews function when the source file is read-only, set this variable to `t`.
`nil`	If the master file is read-only, the vmsUpdateCellViews function will not perform the update. This is the default value.

vmsNcvlogExecutable

Specifies which ncvlog executable is to be used to parse the Verilog-AMS text file.

```
vmsNcvlogExecutable_variable ::=
        vmsNcvlogExecutable = "path_and_executable"
```

The parameter is the following:

path_and_executable

The executable used to parse the text file.

vmsPortProcessing

Determines how port concatenations are handled when the environment generates a Verilog-AMS text view from another cellview.

```
vmsPortProcessing_variable ::=
        vmsPortProcessing = "Analog" | "Digital" | "Mixed"
```

The parameters are the following:

`Analog`	Port concatenations remain as concatenations in the generated cellviews.
`Digital`	Port concatenations remain as concatenations in the generated cellviews. This is the default value for verilog text views when `schHdlUseVamsForVerilog` is set to `t`.
`Mixed`	Port concatenations in generated cellviews are translated to the format expected by the AMS netlister. This is the default value for verilog-ams text views. This is the default value for verilog text views when `schHdlUseVamsForVerilog` is set to `nil`.

Example

You have a symbol with two terminals named `a<2:3>,b,c<1>` and `c<2:3>,b`. If `vmsPortProcessing` is set to `Analog` or `Digital` and the terminals are of the inout type, the AMS Designer environment creates the following skeletal text module from the symbol.

```
module <name> ( {a[2:3], b, c[1]}, {c[2:3], b} );
    inout  [1:3] c;
    inout  b;
    inout  [2:3] a;
endmodule
```

If `vmsPortProcessing` is set to `Mixed`, the AMS Designer environment creates the following skeletal module, which is in the format expected by the AMS netlister.

```
module <name> ( a, b, c )
    inout  [1:3] c;
    inout  b;
    inout  [2:3] a;
endmodule
```

vmsRunningInUI

Controls whether messages are displayed in dialog boxes.

```
vmsRunningInUI_variable ::=
        vmsRunningInUI = t | nil
```

The parameters are the following:

`t`	Messages are displayed in dialog boxes rather than in the CIW.
`nil`	Messages are displayed in the CIW.

vmsTemplateScript

Specifies the name of a script used to customize the header information for new Verilog-AMS cellviews.

```
vmsTemplateScript_variable ::=
        vmsTemplateScript = "template_script" | nil
```

The parameters are the following:

template_script	The specified script is used to generate headers.
nil	A default header is used. It has the form `//Verilog-AMS HDL for` *libname*, *cellname viewname*

Example

Assume that `vmsTemplateScript` is set to `"template_script"` and `template_script` contains

```
#!/bin/csh -f
echo "// Verilog-AMS HDL for " \"$1\", \"$2\" \"$3\"
echo ""
echo "// Module    : $2"
echo "// Description   :"
echo "// First Created :" `date`
echo "//"
echo "//"
echo "// MODULE DESCRIPTION :"
echo "//"
echo "// EVENTS DESCRIPTION :"
echo ""
exit 0
```

Now assume that a new cell called `test`, with the view `vams`, is created in the library `vams_test_lib`. A new Verilog-AMS cell is generated with the following information:

```
// Verilog-AMS HDL for "vams_test_lib", "test" "vams"


// Module        : test
// Description   :
// First Created : Wed Apr 5 15:01:26 IST 2000
//
//
// MODULE DESCRIPTION :
//
// EVENTS DESCRIPTION :


module test ( );


endmodule
```

vmsUpdateSymbolAfterEdit

Controls whether symbol views are automatically created for cells that don't have a symbol view when the `vmsUpdateCellViews` function is run.

The parameters are the following:

`t`	The symbol view is automatically created.
`nil`	Disables creation of symbol views.
`query`	Displays a pop-up asking for confirmation whether the symbol view should be created.

Default value: `query`

vmsVerboseMsgLevel

Specifies the highest message level to be printed. Higher numbers result in more messages being printed; lower numbers result in fewer messages being printed.

```
vmsVerboseMsgLevel_variable ::=
        vmsVerboseMsgLevel = message_level
```

The parameter is the following:

message_level	An integer equal to or greater than zero, which is the highest message level to be printed.

Level 0 (zero) messages are printed as is. Messages of level 1 or higher are prefixed with `VAMS Diagnostics:`

Messages are categorized as fatal (`F`), warning (`W`), or error (`E`) and each is displayed with a mnemonic. For example,

```
\o VAMS *W, MNEERR: Inherited Expressions for multiple member terms for port "zz"
\o       cannot be formatted. Declaring it without any net expression
\o       attribute.
```

F

Compiling Cadence-Provided Libraries

Cadence provides the `amsLibCompile` program to compile Verilog modules in the following Cadence libraries:

- ahdlLib
- basic
- bmslib
- rfLib
- sample

The amsLibCompile program runs automatically during the post-load phase of the amsEnv kit installation process. Before the program runs, either automatically or manually, ensure that:

- The kits containing the referenced libraries are installed so that the libraries exist in the hierarchy
- The Verilog compiler, ncvlog, is in the executable search path (`$PATH`)

If libraries fail to compile during the installation process, or if the libraries need to be recompiled for use with a different version of the NC programs, you can run `amsLibCompile` manually. The following guidelines apply:

- You must have write permission to the libraries
- You must start the amsLibCompile program from the root of the DFII installation hierarchy. In other words, you must start the program in the directory that contains the `tools` directory

Consider the following example where there are two LDV installations available:

- `/ams/ldv50/s1`
- `/usr2/ldv51`

and your `$PATH` includes only `/ams/ldv50/s1`.

To use `amsLibCompile`, do the following:

1. Run the `amsLibCompile` command.

   ```
   amsLibCompile
   ```

   The program prompts you to select one of the following responses:

   ```
   1) Add an LDV installation directory
   2) Compile with /ams/ldv50/s1
   3) Quit
   ```

2. The LDV hierarchy listed is not the one you want to use, so select the first option to add another hierarchy:

   ```
   1
   ```

   The program prompts you as follows:

   ```
   Enter path (as returned by ncroot command):
   ```

3. Type the fully-qualified path to the hierarchy containing the ncvlog executable that you want to use to compile the libraries.

   ```
   /usr2/ldv51
   ```

   The program adds this path to the list of LDV hierarchies.

4. When asked to do so, press *Enter* to continue:

   The program returns to the numbered menu, which now includes your newly-entered path to the LDV hierarchy:

   ```
   1) Add an LDV installation directory
   2) Compile with /ams/ldv50/s1
   3) Compile with /usr2/ldv51
   4) Quit
   ```

5. Choose the number that corresponds to the LDV hierarchy that you want to use to compile the libraries:

   ```
   3
   ```

   The program processes each of the libraries in turn, printing status messages about the success or failure of each compilation.

6. When asked to do so, press *Enter* to continue:

7. When you are done compiling libraries, type the number corresponding to `Quit`

   ```
   4
   ```

Note: Compiling the libraries with ncvlog from different versions of the LDV hierarchy creates separate, version-specific `.pak` files in your libraries so you can use them later with the different versions of AMS Designer.

G

Migrating from Previous Versions of the AMS Designer Environment

If you are familiar with using the AMS Designer environment in releases prior to IC 6.1.2, the following information is for you.

- The ams.env File on page 697
- AMS Design Prep Form on page 697
- AMS Options for Global Design Data on page 698
- AMS Direct Plot Form on page 698
- AMS Designer Simulations on page 698

The ams.env File

The Virtuoso® AMS Designer environment now works with ADE state files and the `.cdsenv` default mechanism. When you specify a run directory that contains an `ams.env` file (from a previous version), the program converts it into state files automatically and stores state information in the `.amsd_state` directory in the run directory:

```
runDirectory/.amsd_state
```

After converting it, the program no longer uses it.

AMS Design Prep Form

The AMS Design Prep form had options for netlisting and compiling. You will find these options on the in the *RUN OPTIONS* section of the Netlist and Run form.

AMS Options for Global Design Data

The AMS Options form had options for specifying the global design data module and default global signal declarations. You will find information about these options here:

- Global Design Data Module (cds_globals) on page 200
- Global Signals on page 199

AMS Direct Plot Form

Access to the Direct Plot feature is currently missing from the Virtuoso AMS Designer environment.

AMS Designer Simulations

AMS Designer simulations take place in the `netlist` subdirectory of the run directory. For previous versions of AMS Designer, simulations took place directly in the run directory. Because of this difference, paths (from state files) that worked in previous versions of AMS Designer might not continue to work. Verify that paths are correct if you are importing information from a previous version or from an ADE state, particularly on the Model Library Setup form.

Index