

# **Virtuoso<sup>®</sup> AMS Environment User Guide**

**Product Version 5.3**  
**April 2004**

© 2000-2004 Cadence Design Systems, Inc. All rights reserved.  
Printed in the United States of America.

Cadence Design Systems, Inc., 555 River Oaks Parkway, San Jose, CA 95134, USA

**Trademarks:** Trademarks and service marks of Cadence Design Systems, Inc. (Cadence) contained in this document are attributed to Cadence with the appropriate symbol. For queries regarding Cadence's trademarks, contact the corporate legal department at the address shown above or call 800.862.4522.

All other trademarks are the property of their respective holders.

**Restricted Print Permission:** This publication is protected by copyright and any unauthorized use of this publication may violate copyright, trademark, and other laws. Except as specified in this permission statement, this publication may not be copied, reproduced, modified, published, uploaded, posted, transmitted, or distributed in any way, without prior written permission from Cadence. This statement grants you permission to print one (1) hard copy of this publication subject to the following conditions:

1. The publication may be used solely for personal, informational, and noncommercial purposes;
2. The publication may not be modified in any way;
3. Any copy of the publication or portion thereof must include all original copyright, trademark, and other proprietary notices and this permission statement; and
4. Cadence reserves the right to revoke this authorization at any time, and any such use shall be discontinued immediately upon written notice from Cadence.

**Disclaimer:** Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. The information contained herein is the proprietary and confidential information of Cadence or its licensors, and is supplied subject to, and may be used only by Cadence's customer in accordance with, a written agreement between Cadence and its customer. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information.

**Restricted Rights:** Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor.

# Contents

<u>Preface</u> .....	17
<u>Related Documents</u> .....	18
<u>Typographic and Syntax Conventions</u> .....	19
 <u>1</u>	
<u>Overview of the Virtuoso AMS Designer Flow</u> .....	21
<u>The AMS Designer Flow Supports Both Analog and Digital Designers</u> .....	22
<u>Creating HDL Modules for CDBA Cellviews</u> .....	23
<u>Creating HDL Data as You Save CDBA Cellviews</u> .....	23
<u>Creating HDL Data from Pre-existing CDBA Cellviews</u> .....	23
 <u>2</u>	
<u>Quick-Start Tutorial</u> .....	25
<u>The Circuit</u> .....	25
<u>AMS Designer Tools</u> .....	26
<u>Setting Up the Tutorial</u> .....	27
<u>Running from a Script</u> .....	27
<u>Running within the AMS Environment</u> .....	28
<u>Opening the Command Interpreter Window</u> .....	29
<u>Opening the Schematic and Design Configuration</u> .....	29
<u>Netlisting and Compiling</u> .....	33
<u>Elaborating and Simulating the Design</u> .....	43
<u>Summary</u> .....	55
 <u>3</u>	
<u>Setting Up the AMS Environment</u> .....	57
<u>Overview</u> .....	58
<u>The hdl.var File</u> .....	58
<u>The ams.env Files</u> .....	59
<u>AMS Designer Supports Design Management</u> .....	60

## Virtuoso AMS Environment User Guide

---

<u>Specifying the Text Editor to Use</u> .....	60
<u>Specifying Fonts for the Cadence Hierarchy Editor</u> .....	60
<u>Preparing to Use AMS Designer from the Hierarchy Editor</u> .....	62

## 4

<u>Netlisting</u> .....	67
<u>Netlisting Modes Supported by the AMS Netlister</u> .....	68
<u>Automatic Netlisting of a Cellview</u> .....	68
<u>Netlist Updating and Netlisting of Entire Designs</u> .....	70
<u>Netlisting from the UNIX Command Line</u> .....	70
<u>Library Netlisting</u> .....	72
<u>Netlisting of Cells in Response to Changes in CDF</u> .....	74
<u>Preparing Existing Analog Primitive Libraries for Netlisting</u> .....	74
<u>Specifying the Behavior of the Netlister and Compilers</u> .....	74
<u>Opening the AMS Options Windows</u> .....	75
<u>Setting Netlister Options from the Hierarchy Editor</u> .....	77
<u>Opening the CIW AMS Options Window</u> .....	86
<u>Setting Compiler Options</u> .....	98
<u>Viewing the AMS Netlister Log</u> .....	126
<u>Understanding the Output from the AMS Netlister</u> .....	126
<u>How Inherited Connections Are Netlisted</u> .....	127
<u>Inherited Signal Connections</u> .....	128
<u>Inherited Terminal Connections</u> .....	129
<u>Instance Values for Inherited Connections</u> .....	129
<u>Third-Party Tools and Other Cadence Tools</u> .....	130
<u>How Aliased Signals Are Netlisted</u> .....	131
<u>How m-factors (Multiplicity Factors) Are Netlisted</u> .....	132
<u>How Iterated Instances Are Netlisted</u> .....	133
<u>Passing Model Names as Parameters</u> .....	134
<u>Effect of the modelName, model, and modelName Parameters</u> .....	134
<u>Handling of the model* and componentName Parameters</u> .....	136
<u>Precedence of the model* and componentName Parameters</u> .....	136
<u>Specifying Parameters to be Excluded from Netlisting</u> .....	137
<u>Ignoring Parameters for Entire Libraries</u> .....	137
<u>Example: Specifying Parameters to Ignore</u> .....	140

---

<u>Preparing to Netlist User-Defined Functions</u> .....	140
<u>Ensuring that Floating Point Parameters Netlist Correctly</u> .....	143
<u>Defining the Parameter and Specifying a Default Value</u> .....	143
<u>Overriding the Default Parameter for Specific Instances</u> .....	145

## 5

<u>Working with Schematic Designs</u> .....	149
<u>Setting Schematic Rules Checker Options for AMS Designer</u> .....	150
<u>Creating Cellviews Using the AMS Environment</u> .....	152
<u>Preparing a Library</u> .....	152
<u>Creating the Symbol View</u> .....	155
<u>Using Blocks</u> .....	155
<u>Creating a Verilog-AMS or VHDL-AMS Cellview from a Symbol or Block</u> .....	157
<u>Descend Edit</u> .....	160
<u>Creating a Verilog-AMS Cellview</u> .....	161
<u>Creating a VHDL-AMS Cellview</u> .....	163
<u>Creating a Symbol Cellview from a Verilog-AMS Cellview</u> .....	165
<u>Inherited Connections</u> .....	167
<u>Global Signals in the Schematic Editor</u> .....	167
<u>Inherited Connections in a Hierarchy</u> .....	168
<u>Defining Inherited Connections</u> .....	170
<u>How Net Expressions Evaluate</u> .....	171
<u>Net and Pin Properties</u> .....	172
<u>groundSensitivity and supplySensitivity Properties</u> .....	173
<u>Making Connect Modules Sensitive to Inherited Connection Values</u> .....	176

## 6

<u>Using External Text Designs</u> .....	179
<u>Overview of Steps for Using External Text Designs</u> .....	180
<u>Bringing Modules into a Cadence Library</u> .....	180
<u>Specifying the Working Library</u> .....	180
<u>Compiling into Libraries</u> .....	181
<u>Compiling into Temporary Libraries</u> .....	182
<u>Listing Compiled Modules</u> .....	183
<u>Using Text Blocks in Schematics</u> .....	184

## Virtuoso AMS Environment User Guide

---

<u>Using Modules Located in a Cadence Library</u> .....	186
<u>Creating a Configuration</u> .....	186
<u>Preparing for Simulation</u> .....	186

## 7

<u>Using Existing Designs in the AMS Environment</u> .....	187
------------------------------------------------------------	-----

<u>Using Analog Primitives</u> .....	188
<u>Using SPICE and Spectre Netlists and Subcircuits</u> .....	188
<u>Preparing to Use SPICE and Spectre Netlists and Subcircuits</u> .....	188
<u>Placing SPICE and Spectre Netlists and Subcircuits in a Schematic</u> .....	189

## 8

<u>Using Test Fixtures</u> .....	193
----------------------------------	-----

<u>Creating and Using a Textual Test Fixture</u> .....	194
<u>Creating a Textual Test Fixture</u> .....	194
<u>Using a Test Fixture</u> .....	195
<u>Example: Creating and Using a Test Fixture</u> .....	195

## 9

<u>Using Design Configurations</u> .....	199
------------------------------------------	-----

<u>Overview of Configurations</u> .....	200
<u>Creating a Config Cellview</u> .....	200
<u>Using VHDL Modules in a Configuration</u> .....	206
<u>Ensuring HDL Design Unit Information Is Current</u> .....	206
<u>Using a Configuration</u> .....	206

## 10

<u>Preparing a Design for Simulation</u> .....	209
------------------------------------------------	-----

<u>Overview of AMS Design Prep</u> .....	210
<u>What AMS Design Prep Does to Prepare a Design for Simulation</u> .....	210
<u>When to Use AMS Design Prep</u> .....	211
<u>Specifying the Behavior of AMS Design Prep</u> .....	211
<u>Setting Options for Global Design Data</u> .....	211

## Virtuoso AMS Environment User Guide

---

<u>Specifying Global Signals</u> .....	213
<u>Specifying Design Variables</u> .....	216
<u>Specifying Model Files to Use During Elaboration</u> .....	219
<u>Running AMS Design Prep</u> .....	221
<u>How AMS Design Prep Handles Global Signals and Design Variables</u> .....	225
<u>The cds_globals Module</u> .....	226
<u>Global Signals</u> .....	228
<u>Design Variables</u> .....	229

## 11

<u>Elaborating, Simulating, and Plotting Results</u> .....	231
<u>Specifying the Behavior of the Elaborator, Simulator, and Waveform Viewer</u> .....	232
<u>Setting Elaborator Options</u> .....	232
<u>Setting Simulator Options</u> .....	247
<u>Setting Waveform Selection Options</u> .....	274
<u>Creating Probes</u> .....	277
<u>Defining Databases for Waveforms</u> .....	281
<u>Selecting Instances from the Virtuoso Schematic Editing Window</u> .....	282
<u>Selecting Buses</u> .....	283
<u>Selecting Instances from the Scope Navigator</u> .....	284
<u>Copying and Pasting Within Tables</u> .....	285
<u>Elaborating and Simulating</u> .....	286
<u>Viewing Messages</u> .....	289
<u>Plotting Waveforms After Simulation Ends</u> .....	290
<u>Starting the SimVision Waveform Viewer</u> .....	291
<u>Plotting Waveforms Selected on a Schematic (Direct Plot)</u> .....	293

## 12

<u>Using the amsdesigner Command</u> .....	295
<u>Examples</u> .....	297

## 13

<u>Producing Customized Netlists</u> .....	299
<u>Producing Customized Netlists</u> .....	300

## Virtuoso AMS Environment User Guide

---

<u>Identifying the Sections of a Netlist</u>	301
<u>Using ams.env Variables to Customize Netlists</u>	302
<u>Using Netlisting Procedures to Customize Netlists</u>	303
<u>Examples: Problems Addressed by Customized Netlists</u>	314
<u>Example: Adjusting Parameter Values to Account for Number of Fingers</u>	314
<u>Example: Using Symbols that Represent Verilog Test Code</u>	318
<u>Example: Using CDF Instance Parameters to Define Inherited Connections</u>	322
<u>Data Objects Supported for Netlisting</u>	327
<u>Netlister Object</u>	327
<u>Formatter Object</u>	328
<u>Cellview Object</u>	330
<u>Parameter Object</u>	331
<u>Instance Object</u>	332
<u>SKILL Functions Supported for Netlisting</u>	333
<u>Default Netlisting Procedures</u>	334
<u>Netlisting Helper Functions</u>	344

## A

<u>Variables for ams.env Files</u>	359
<u>How AMS Designer Determines the Set of Variables</u>	360
<u>Why AMS Designer Uses ams.env Files, Not .cdsenv Files</u>	360
<u>List of ams.env Variables</u>	362
<u>Detailed Descriptions of ams.env Variables</u>	366
<u>aliasInstFormat</u>	367
<u>allowDeviantBuses</u>	368
<u>allowIllegalIdentifiers</u>	370
<u>allowNameCollisions</u>	372
<u>allowSparseBuses</u>	374
<u>allowUndefParams</u>	376
<u>amsCompMode</u>	377
<u>amsDefinitionViews</u>	378
<u>amsEligibleViewTypes</u>	380
<u>amsExcludeParams</u>	381
<u>amsExpScalingFactor</u>	382
<u>amsLSB_MSB</u>	384



## Virtuoso AMS Environment User Guide

---

<a href="#"><u>amsMaxErrors</u></a>	385
<a href="#"><u>amsScalarInstances</u></a>	386
<a href="#"><u>amsVerbose</u></a>	387
<a href="#"><u>analogControlFile</u></a>	388
<a href="#"><u>bindCdsAliasLib</u></a>	389
<a href="#"><u>bindCdsAliasView</u></a>	390
<a href="#"><u>cdsGlobalsLib</u></a>	391
<a href="#"><u>cdsGlobalsView</u></a>	392
<a href="#"><u>checkAndNetlist</u></a>	393
<a href="#"><u>checkOnly</u></a>	394
<a href="#"><u>checktasks</u></a>	395
<a href="#"><u>compileAsAMS</u></a>	396
<a href="#"><u>compileExcludeLibs</u></a>	397
<a href="#"><u>compileMode</u></a>	398
<a href="#"><u>connectRulesCell</u></a>	400
<a href="#"><u>connectRulesCell2</u></a>	401
<a href="#"><u>connectRulesLib</u></a>	402
<a href="#"><u>connectRulesView</u></a>	403
<a href="#"><u>detailedDisciplineRes</u></a>	404
<a href="#"><u>discipline</u></a>	405
<a href="#"><u>excludeViewNames</u></a>	406
<a href="#"><u>hdlVarFile</u></a>	407
<a href="#"><u>headerText</u></a>	408
<a href="#"><u>ieee1364</u></a>	409
<a href="#"><u>ifdefLanguageExtensions</u></a>	410
<a href="#"><u>incdir</u></a>	411
<a href="#"><u>includeFiles</u></a>	412
<a href="#"><u>includeInstCdfParams</u></a>	413
<a href="#"><u>initFile</u></a>	414
<a href="#"><u>instClashFormat</u></a>	415
<a href="#"><u>iterInstExpFormat</u></a>	416
<a href="#"><u>language</u></a>	417
<a href="#"><u>lexpragma</u></a>	418
<a href="#"><u>logFileAction</u></a>	420
<a href="#"><u>logFileName</u></a>	421
<a href="#"><u>macro</u></a>	423

## Virtuoso AMS Environment User Guide

---

<u>markcelldefines</u>	424
<u>maxErrors</u>	425
<u>messages</u>	426
<u>modifyParamScope</u>	427
<u>ncelabAccess</u>	431
<u>ncelabAfile</u>	432
<u>ncelabAnnoSimtime</u>	433
<u>ncelabArguments</u>	434
<u>ncelabCoverage</u>	435
<u>ncelabDelayMode</u>	436
<u>ncelabDelayType through ncelabMessages</u>	438
<u>ncelabMixEsc</u>	439
<u>ncelabModelFilePaths</u>	440
<u>ncelabNeverwarn through ncelabVipdelay</u>	441
<u>ncsimArguments</u>	444
<u>ncsimEpulseNoMsg through ncsimExtassertmsg</u>	445
<u>ncsimGUI</u>	446
<u>ncsimLoadvpi through ncsimStatus</u>	447
<u>ncsimTcl</u>	448
<u>ncsimUnbuffered through ncsimUseAddArgs</u>	449
<u>ncvhdlArguments</u>	450
<u>ncvlogArguments</u>	451
<u>ncvlogUseAddArgs</u>	452
<u>netClashFormat</u>	453
<u>netlistAfterCdfChange</u>	454
<u>netlistMode</u>	455
<u>netlistUDFAsMacro</u>	456
<u>neverwarn</u>	458
<u>noline</u>	459
<u>nomempack</u>	460
<u>nopragsawarn</u>	461
<u>nostdout</u>	462
<u>nowarn</u>	463
<u>paramDefVals</u>	464
<u>paramGlobalDefVal</u>	465
<u>pragma</u>	466

## Virtuoso AMS Environment User Guide

---

<u>processViewNames</u>	467
<u>prohibitCompile</u>	468
<u>runNcelab</u>	469
<u>runNcsim</u>	470
<u>scaddlgblopts</u>	471
<u>scaddltranopts</u>	472
<u>scale</u>	473
<u>scalem</u>	474
<u>scannotate</u>	475
<u>scapprox</u>	476
<u>scaudit</u>	477
<u>sccheckstmt</u>	478
<u>sccmin</u>	479
<u>sccompatible</u>	480
<u>scdebug</u>	481
<u>scdiagnose</u>	482
<u>scdigits</u>	483
<u>scerror</u>	484
<u>scerrpreset</u>	485
<u>scfastbreak</u>	486
<u>scgmin</u>	487
<u>scgmincheck</u>	488
<u>schomotopy</u>	489
<u>sciabstol</u>	490
<u>scic</u>	491
<u>scicstmt</u>	492
<u>scignshorts</u>	493
<u>scinfo</u>	494
<u>scinventory</u>	495
<u>sclimit</u>	496
<u>sciteratio</u>	497
<u>scmacromod</u>	498
<u>scmaxiters</u>	499
<u>scmaxnotes</u>	500
<u>scmaxrsd</u>	501
<u>scmaxstep</u>	502

## Virtuoso AMS Environment User Guide

---

<u>scmaxwarn</u>	503
<u>scmethod</u>	504
<u>scmodelevaltype</u>	505
<u>scmosvres</u>	506
<u>scnarrate</u>	507
<u>scnotation</u>	508
<u>scnote</u>	509
<u>scopptcheck</u>	510
<u>scpivabs</u>	511
<u>scpivotdc</u>	512
<u>scpivrel</u>	513
<u>scquantities</u>	514
<u>screadic</u>	515
<u>screadns</u>	516
<u>screlref</u>	517
<u>screltol</u>	518
<u>scrforce</u>	519
<u>scscale</u>	520
<u>scscalem</u>	521
<u>scscfincfile</u>	522
<u>scscftimestamp</u>	523
<u>scscfusefileflag</u>	524
<u>scskipcount</u>	525
<u>scskipdc</u>	526
<u>scskipstart</u>	527
<u>scskipstop</u>	528
<u>scspeed</u>	529
<u>scspscflag</u>	531
<u>scstats</u>	532
<u>scstep</u>	533
<u>scstop</u>	534
<u>scstrobedelay</u>	535
<u>scstrobeperiod</u>	536
<u>sctemp</u>	537
<u>sctempeffects</u>	538
<u>sctitle</u>	539

## Virtuoso AMS Environment User Guide

---

<u>sctnom</u>	540
<u>sctopcheck</u>	541
<u>sctransave</u>	542
<u>scusemodeleval</u>	543
<u>scvabstol</u>	544
<u>scwarn</u>	545
<u>scwrite</u>	546
<u>scwritefinal</u>	547
<u>simRunDirLoc</u>	548
<u>simVisScriptFile</u>	549
<u>status</u>	550
<u>templateFile</u>	551
<u>templateScript</u>	552
<u>timescale</u>	553
<u>update</u>	554
<u>use5xForVHDL</u>	555
<u>useDefparam</u>	556
<u>useNcelabNowarn</u>	558
<u>useNcelabSdfCmdFile</u>	559
<u>useNcsimNowarn</u>	560
<u>useNowarn</u>	561
<u>useScaddlgblopts</u>	562
<u>useScaddltranopts</u>	563
<u>useScic</u>	564
<u>useScreadic</u>	565
<u>useScreadns</u>	566
<u>useScscfincfile</u>	567
<u>useScwrite</u>	568
<u>useScwritefinal</u>	569
<u>useSimVisScriptFile</u>	570
<u>useProcessViewNamesOnly</u>	571
<u>verboseUpdate</u>	572
<u>vlogGroundSigs</u>	573
<u>vloglinedebug</u>	574
<u>vlogSupply0Sigs</u>	575
<u>vlogSupply1Sigs</u>	576

## Virtuoso AMS Environment User Guide

---

<u>wfDefaultDatabase</u>	577
<u>wfDefInstCSaveAll</u>	578
<u>wfDefInstCSaveLvl</u>	579
<u>wfDefInstSaveCurrents</u>	580
<u>wfDefInstSaveVoltages</u>	581
<u>wfDefInstVSaveAll</u>	582
<u>wfDefInstVSaveLvl</u>	583
<u>wfDefInstVSaveObjects</u>	584

## B

<u>Updating Legacy SimInfo for Analog Primitives</u>	585
<u>The ams Fields</u>	585
<u>Special Handling of model, modelName, modelname, and componentName</u>	597
<u>Converting an Existing Analog Primitive Library</u>	598

## C

<u>Designing for Virtuoso AMS Compliance</u>	603
<u>Identifiers</u>	604
<u>Follow the Recommended Syntax for Identifiers</u>	604
<u>Ensure that Identifiers Map Cleanly to Netlist Languages</u>	604
<u>Ensure that Identifiers Are Unique within Your Design</u>	605
<u>Terminals</u>	606
<u>Buses</u>	607
<u>Component Description Format</u>	608
<u>Parameters</u>	608
<u>Using Inherited Parameters</u>	608
<u>Using Cell Parameters</u>	609
<u>Using Efficient Formats for Parameter Values</u>	609
<u>Parameterized Cells</u>	609
<u>VHDL-AMS Component Declarations</u>	610
<u>Properties</u>	610
<u>Properties to Avoid Completely</u>	610
<u>Avoid the portOrder Property Unless Required by Special Circumstances</u>	610
<u>Properties to Use Only in AMS Compatibility Mode</u>	611
<u>Properties That Have No Special Meaning in the AMS Environment</u>	611

---

<u>Properties Fully Supported by the AMS Environment</u> .....	612
----------------------------------------------------------------	-----

### D

<u>SKILL Functions and Customization Variables</u> .....	615
----------------------------------------------------------	-----

<u>SKILL Functions</u> .....	616
<u>amsCheckCV</u> .....	617
<u>amsIsPresent</u> .....	618
<u>amsNetlist</u> .....	619
<u>amsProcessCellViews</u> .....	622
<u>amsUIOptionsForm</u> .....	625
<u>amsUIRunNetlisterForm</u> .....	626
<u>ddsCvtAMSTranslateCell</u> .....	627
<u>ddsCvtAMSTranslateLib</u> .....	630
<u>ddsCvtToolBoxAMS</u> .....	632
<u>vmsUpdateCellViews</u> .....	633
<u>Customization Variables</u> .....	635
<u>schHdlNotCreateDB</u> .....	636
<u>schHdlUseVamsForVerilog</u> .....	637
<u>vmsAnalysisType</u> .....	638
<u>vmsCreateMissingMasters</u> .....	639
<u>vmsNcvlogExecutable</u> .....	640
<u>vmsPortProcessing</u> .....	641
<u>vmsRunningInUI</u> .....	642
<u>vmsTemplateScript</u> .....	643
<u>vmsVerboseMsgLevel</u> .....	644

### E

<u>Compiling Cadence-Provided Libraries</u> .....	645
---------------------------------------------------	-----

<u>Purpose of the amsLibCompile Tool</u> .....	646
<u>Running the amsLibCompile Tool Manually</u> .....	646
<u>Example</u> .....	646

## Virtuoso AMS Environment User Guide

---



---

# Preface

---

The Virtuoso<sup>®</sup> AMS environment part of Virtuoso AMS Designer provides a comfortable framework for developing, elaborating, simulating, and debugging blocks. This user guide discusses that environment and details how to make the best use of it.

The preface discusses the following:

- Related Documents on page 18
- Typographic and Syntax Conventions on page 19

## Related Documents

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

- [\*Cadence Analog Circuit Design Environment User Guide\*](#)
- [\*Cadence Mixed-Signal Circuit Design Environment\*](#)
- [\*NC Verilog Simulator Help\*](#)
- [\*NC VHDL Simulator Help\*](#)
- [\*SimVision Analysis Environment User Guide\*](#)
- [\*Virtuoso Spectre Circuit Simulator Reference\*](#)
- [\*Virtuoso Spectre Circuit Simulator User Guide\*](#)
- [\*Cadence Verilog-A Language Reference\*](#)
- [\*Cadence AMS Simulator User Guide\*](#)
- [\*Cadence Application Infrastructure User Guide\*](#)
- [\*Cadence Hierarchy Editor User Guide\*](#)
- [\*Cadence Library Manager User Guide\*](#)
- [\*Cadence Verilog-AMS Language Reference\*](#)
- [\*Component Description Format User Guide\*](#)
- [\*IEEE Standard VHDL Language Reference Manual \(Integrated with VHDL-AMS Changes\), IEEE Std 1076.1. Available from IEEE.\*](#)
- [\*Instance-Based View Switching Application Note\*](#)
- [\*Simvision Waveform Viewer User Guide\*](#)
- [\*Verilog-AMS Language Reference Manual. Available from Open Verilog International.\*](#)
- [\*Verilog-XL Reference\*](#)
- [\*Cadence VHDL-AMS Overview\*](#)
- [\*Virtuoso Schematic Editor User Guide\*](#)

For information about problems, see the [\*Virtuoso AMS Environment Known Problems and Solutions\*](#).

## Typographic and Syntax Conventions

Special typographical conventions are used to distinguish certain kinds of text in this document. The formal syntax used in this reference uses the definition operator, `::=`, to define the 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 can be repeated zero or more times. For example,

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

- Code examples are displayed in constant-width font.

# Virtuoso AMS Environment User Guide

## Preface

---

`/* 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 are set 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 is indented on the next line, like this:

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

---

# Overview of the Virtuoso AMS Designer Flow

---

The AMS Designer flow uses the AMS environment and a set of tools tuned to facilitate the development of mixed-signal designs. The flow gives you a mixed-signal, mixed-language, block-based design solution featuring the following characteristics.

- A top-down or bottom-up development methodology for both analog and mixed-signal designs.
- The ability to mix blocks at different levels of abstraction for an optimized balance of simulation accuracy versus speed
- The ability to create Verilog<sup>®</sup>-AMS netlists for an entire library with a single command
- Automatic, non-destructive conversion of existing analog primitive libraries
- A design capture environment that facilitates both text and schematic data entry
- Support for the Verilog-AMS, VHDL-AMS, Spectre<sup>®</sup>, and SPICE languages
- Flexible and accurate representation of analog/digital interface boundaries and accurate, automatic insertion of interface elements
- Flexible and accurate representation of mixed-language boundaries
- Flexible design configuration, which allows easy switching from one design representation to another
- Accelerated, automatic netlist generation from schematics
- A single debugging environment for analog, digital, and mixed-signal parts of your design
- Stand-alone simulation when desired
- A single unified waveform display for analog, digital, and mixed signal nets in your design

The tools in the AMS Designer flow work smoothly together, so you can move from one tool to another without worrying about whether your data is in the necessary format. For example, when you work within the flow, you can develop a schematic, define a configuration, and then

## Virtuoso AMS Environment User Guide

### Overview of the Virtuoso AMS Designer Flow

---

simulate without working through a netlisting step: the flow netlists the design for you as needed or uses a netlist created by someone else for a block you are referencing.

There are many tools included in the AMS Designer flow. They are listed here, along with cross-references to detailed discussions of their capabilities.

---

Tool	For more information, see
AMS Design Prep	<a href="#"><u>Chapter 10, “Preparing a Design for Simulation”</u></a>
AMS netlister	<a href="#"><u>Chapter 4, “Netlisting”</u></a>
AMS simulator	<a href="#"><u><i>Virtuoso AMS Simulator User Guide</i></u></a>
Hierarchy Editor	<a href="#"><u><i>Cadence Hierarchy Editor User Guide</i></u></a>
Library Manager	<a href="#"><u><i>Cadence Library Manager User Guide</i></u></a>
SimVision	<a href="#"><u><i>SimVision Analysis Environment User Guide</i></u></a>
SimVision Waveform Viewer	<a href="#"><u><i>SimVision Waveform Viewer User Guide</i></u></a>
Virtuoso schematic editor capture tool	<a href="#"><u><i>Virtuoso Schematic Editor User Guide</i></u></a>
CDF editor	<a href="#"><u><i>Component Description Format User Guide</i></u></a>
Schematic rule checker (SRC)	<a href="#"><u><i>Virtuoso Schematic Editor User Guide</i></u></a>
ncshell utility	<a href="#"><u>“Using Mixed-Language Designs” chapter in <i>Virtuoso AMS Simulator User Guide</i></u></a>

---

## The AMS Designer Flow Supports Both Analog and Digital Designers

The AMS Designer flow presents a familiar face to both analog and digital designers. Analog designers often work in a graphical environment while digital designers tend to work in a text-based environment, relying on synthesis to transform their hardware description language (HDL) descriptions into actual circuitry. The AMS Designer flow brings these two approaches together, so a designer who prefers to work with text always has up-to-date HDL representations of schematic designs.

This dual text and graphical approach is available throughout the flow for both analog and digital designs. For example, to follow an all text approach you might use a text editor to code an HDL module and use the `ncvlog`, `ncelab`, and `ncsim` commands to prepare and simulate the design. To follow a graphical approach, you might use the Virtuoso® schematic

editor to create components, use the Hierarchy Editor to define a configuration, and use SimVision to simulate and debug the design interactively. In both cases, you use the SimVision Waveform Viewer to display the simulation results.

## **Creating HDL Modules for CDBA Cellviews**

The AMS simulator operates only on modules containing HDL information. Modules containing other kinds of data, such as the CDBA cellviews produced by a schematic capture tool, must be translated into HDL information before the modules can be simulated. The AMS simulator always uses the Verilog-AMS language for netlists, even if your design consists solely of VHDL and VHDL-AMS blocks.

Depending on the design development approach you use, the AMS environment provides different tools to facilitate the creation of HDL modules.

- If you use a schematic capture tool within the AMS environment, the environment automatically creates the HDL modules whenever you save a complete, valid schematic or change CDF information. As a result, when you are ready to simulate, the HDL modules already exist and are ready to use.
- If you develop designs outside of the AMS environment or with HDL creation turned off, the environment provides tools you can use to create the HDL modules needed for a simulation. With these tools, you can create HDL modules for a given library, cell, or view or for all of the modules used in a specified configuration.

The following sections discuss these approaches in more detail.

### **Creating HDL Data as You Save CDBA Cellviews**

In this approach, each time you save a valid schematic or layout block within the AMS environment, the AMS netlister immediately creates a corresponding HDL module. For example, you change a connection in a schematic, then check and save the cellview. If the cellview passes the check, the AMS environment overwrites any existing HDL information for the cellview with new HDL information that corresponds to the changed schematic. Then you change the CDF for a cell. When you save the CDF, the AMS environment creates updated HDL information for all the views of that cell that are specified as eligible.

### **Creating HDL Data from Pre-existing CDBA Cellviews**

Depending on your needs, you can create HDL data from pre-existing CDBA cellviews for

- Specified cellviews located in libraries

- All the cellviews used in a configuration

### **Creating HDL Data for Libraries, Cells, and Views**

To facilitate using libraries that do not have corresponding HDL modules, the AMS environment provides a translator tool, the AMS netlister, which creates HDL data for the CDBA cellviews within a specified library, cell, or view. If you prefer a graphical interface, you can run the AMS netlister through the *Tools* menu in the CIW. If you prefer a non-graphical approach, you can run the AMS netlister from the UNIX command line.

### **Creating HDL Data for Cells Used in a Configuration**

The AMS environment provides a tool, AMS Design Prep, that checks all the cellviews used in a design configuration and creates up-to-date HDL data for them. You run AMS Design Prep from the hierarchy editor.



---

## Quick-Start Tutorial

---

The Virtuoso® AMS environment and simulator work together to enable you to netlist, compile, elaborate, and simulate a circuit that contains analog, digital, and mixed-signal components. The AMS environment consists of the AMS netlister and AMS Design Prep. The former translates CDBA cellviews in your design to Verilog®-AMS netlists, and the latter prepares your design for simulation by letting you manage the global signals and design variables in your design and by ensuring that the netlists are up to date and compiled for elaboration.

In this tutorial, you use the AMS environment and simulator to netlist, compile, elaborate, and simulate the `top` schematic, which contains analog, digital, and mixed-signal components. If you are interested in the operation of the circuit components, read the next section. Otherwise, skip ahead to “[AMS Designer Tools](#)” on page 26.

### The Circuit

The `top` object used in this tutorial is an 8-bit successive approximation A/D converter. (The schematic is shown on page 30.) At the top level, there are six blocks: two analog, two digital (one Verilog, one VHDL), and two mixed-signal (one Verilog-AMS, one VHDL-AMS).

Type	Block name	Hardware design language used
analog	<code>comparator</code>	Verilog-A module or Verilog-AMS netlist
analog	<code>signalSrc</code>	Verilog-A module instantiating Spectre primitives
digital	<code>sareg</code>	Verilog (digital) module
digital	<code>vhdl_clock</code>	VHDL module
mixed-signal	<code>samplehold</code>	Verilog-AMS module
mixed-signal	<code>daconv</code>	VHDL-AMS module

The `signalSrc` block instantiates a Spectre primitive that provides the reference signal. The global supplies are defined using `analogLib` primitives for the positive and negative rails.

The `samplehold` block is a Verilog-AMS module that has both analog and mixed-signal inputs and an analog output. There are no connection modules (interface elements used to connect digital and analog domains) for this block because each port connects to a port of the same domain.

The `vhdl_clock` block is a VHDL clock that provides the clock signal for the successive approximation register (SAR) block, `sareg`.

The `daconv` block is a VHDL-AMS 8-bit D/A block. This block reads the digital output of `sareg` and converts it to an analog signal that is input into the comparator for comparison to the reference signal. This is a mixed-signal block, but no connection modules are needed because its ports are all connected to ports of the same domain.

The `comparator` block is a Verilog-A block (or a Verilog-AMS netlist) that compares the reference signal after it is sampled to the converted digital output from `sareg` via `daconv` to see if conversion has occurred. Because the output of the comparator connects to a digital port on `sareg`, a connection module is needed for simulation.

Therefore, `sareg` is a Verilog block that connects with

- A VHDL block, `vhdl_clock`
- A VHDL-AMS block, `daconv`, via the digital ports
- A Verilog-A block (or Verilog-AMS netlist), `comparator`, where a connection module is needed

## AMS Designer Tools

This tutorial describes how to use the following tools:

- Command Interpreter Window (CIW)
- Cadence hierarchy editor
- AMS netlister
- AMS compiler: `ncvlog`, `ncvhdl`
- AMS Design Prep
- AMS elaborator: `ncelab`
- AMS simulator: `ncsim` version LDV5.0 or later (using SimVision windows)

## Setting Up the Tutorial

Before you can run this tutorial, you need to set up the files and libraries. To do so,

1. Follow the instructions in the README file to set up the tutorial directories.

`your_install_dir/tools/dfII/samples/tutorials/AMS/README`

where `your_install_dir` is the Custom Integrated Circuits (CIC) software installation.

2. Go to the directory where your copy of the tutorial is located.

## Running from a Script

In this section of the tutorial, you run a script that compiles, elaborates, and simulates the design using the textual descriptions of the components. This exercise illustrates how you can use the AMS simulator without using the AMS environment.

1. Use a viewing tool, such as `vi`, to examine the contents of the `runBehavioral` script. It looks like this:

```
#!/bin/csh -f
echo ""
echo " Compiling all analog and digital modules "
echo " for an all behavioral simulation. "
echo ""

cd AMS_lib
rm -f amsLib/*.pak -exec rm {} \;
compile *.vams
compilevhdl daconv.vhd

cd ../digitalLib
rm -f diglib/*.pak -exec rm {} \;
compile *.v
compile *.vams

cd ..

echo ""
echo " Elaborating TEXT ONLY design. "
echo ""
echo " top:module and comparator:module"
echo ""

#####
# Elab of TEXT ONLY design #
#####

ncelab -timescale 1ns/100ps -discipline logic top:module mixedsignal:connect
      -nowarn DLCRDB

echo " Launching the Virtuoso AMS simulator."
ncsim top -amslic -ANALOGCONTROL top.scs -gui -input demo.tcl
```

## Virtuoso AMS Environment User Guide

### Quick-Start Tutorial

---

This script first removes any existing design compilation (.pak) files and then compiles the text files using two `compile` and one `compilevhdl` scripts.

Notice the option `-nowarn DLCRDB` that is appended to the `ncelab` command. This option filters out warning messages about missing temporary libraries. The libraries are not needed for this tutorial.

2. When you are finished looking at the script, close the file.
3. Go to the `digitalLib` directory.
4. Use a viewing tool to examine the contents of the `compile` script, which is called by the `runBehavioral` script that you looked at in [Step 1](#).

This `compile` script calls `ncvhd1` to compile the `clock.vhd` file and then calls `ncvlog` to compile the other text modules.

```
#!/bin/csh -f
ncvhd1 -use5x clock.vhd
foreach i ($argv)
    echo " Compiling $i "
    ncvlog -ams -use5x $i
end
```

5. When you are finished looking at the script, close the file.
6. Go up to the directory where your copy of the tutorial is located.
7. Type `runBehavioral` at the command line to run the `runBehavioral` script.

This script compiles the text modules, elaborates the design, and then opens the SimVision windows so you can simulate the design. You can safely ignore the warning messages about resolving modules during the elaboration step.

8. Exit from SimVision.

(The tutorial demonstrates how to use SimVision later.)

## Running within the AMS Environment

Before you begin this part of the tutorial, be sure that you have followed the steps described in [“Running from a Script”](#) on page 27.

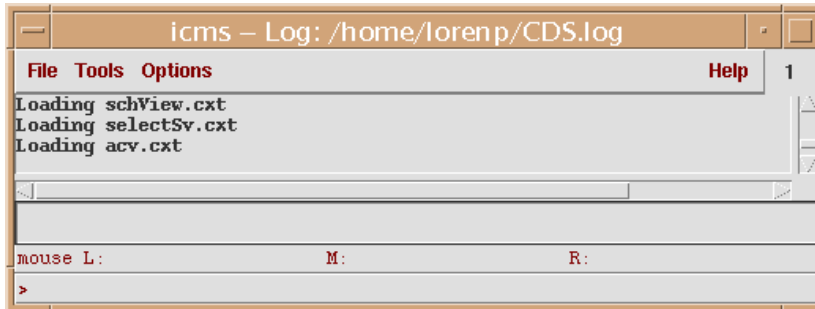
In this part of the tutorial, you use the AMS environment to netlist, compile, elaborate, and simulate the design.

## Opening the Command Interpreter Window

You use the Command Interpreter Window (CIW) to control the design session.

- Type `icms` at the command line.

The CIW appears.



For more information on the CIW, see [“Using the Command Interpreter Window”](#) in *Cadence Design Framework II User Guide*.

## Opening the Schematic and Design Configuration

The tutorial design, at the highest level, consists of a schematic and a corresponding configuration. To open them,

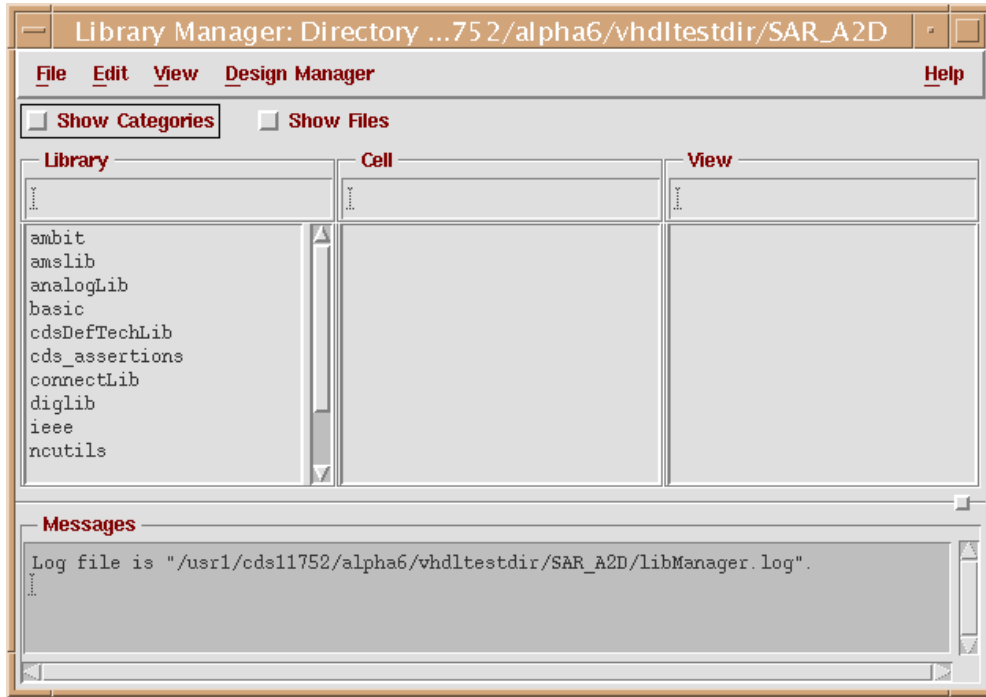
1. In the CIW, choose *Tools – Library Manager*.

## Virtuoso AMS Environment User Guide

### Quick-Start Tutorial

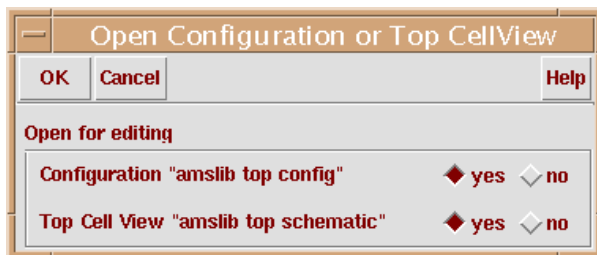
---

The Library Manager form appears.



2. In the *Library* column, click `amslib` to show all the cells in the `amslib` library.
3. In the *Cell* column, click `top` to show all the views in the `top` cell.
4. In the *View* column, double-click on `config`.

The Open Configuration or Top CellView form appears.



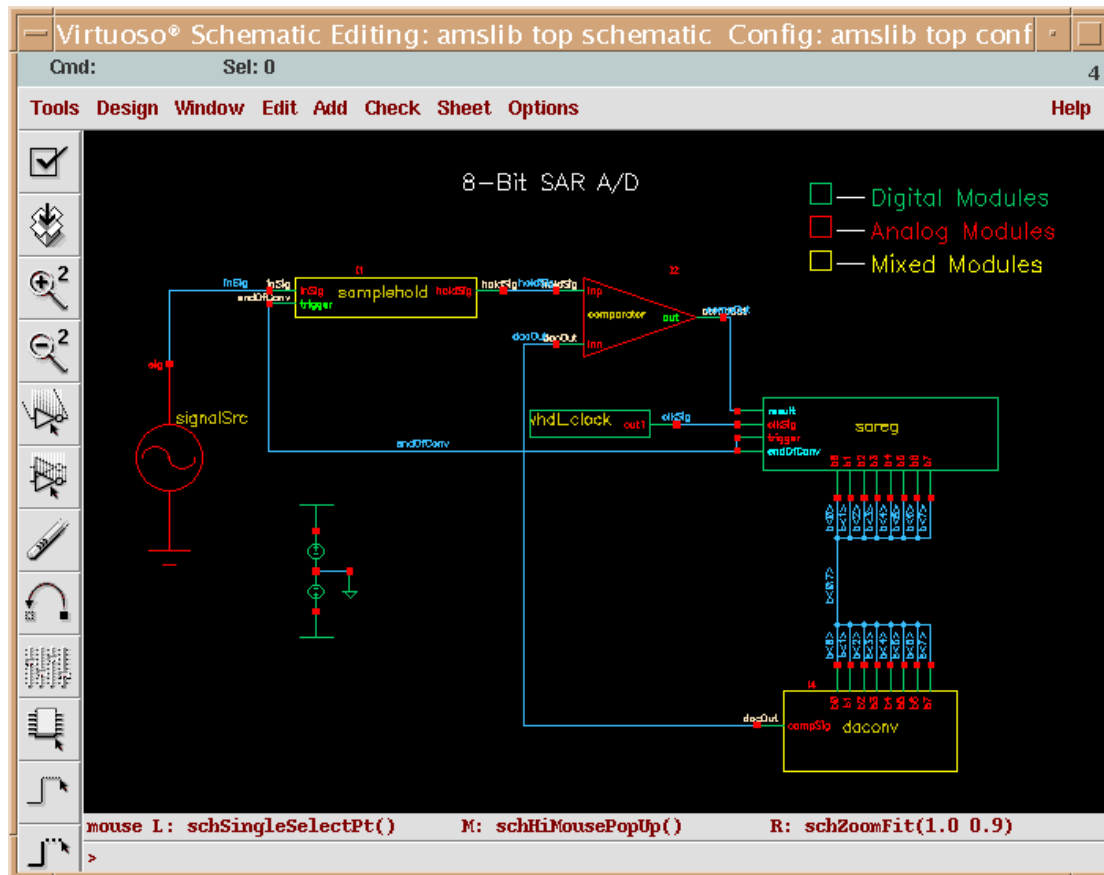
5. Turn on both *Configuration* and *Top Cell View*.
6. Click *OK*.

The hierarchy editor appears, displaying the `top` config view and the Virtuoso® schematic editor appears, displaying the `top` schematic view.

## Virtuoso AMS Environment User Guide

### Quick-Start Tutorial

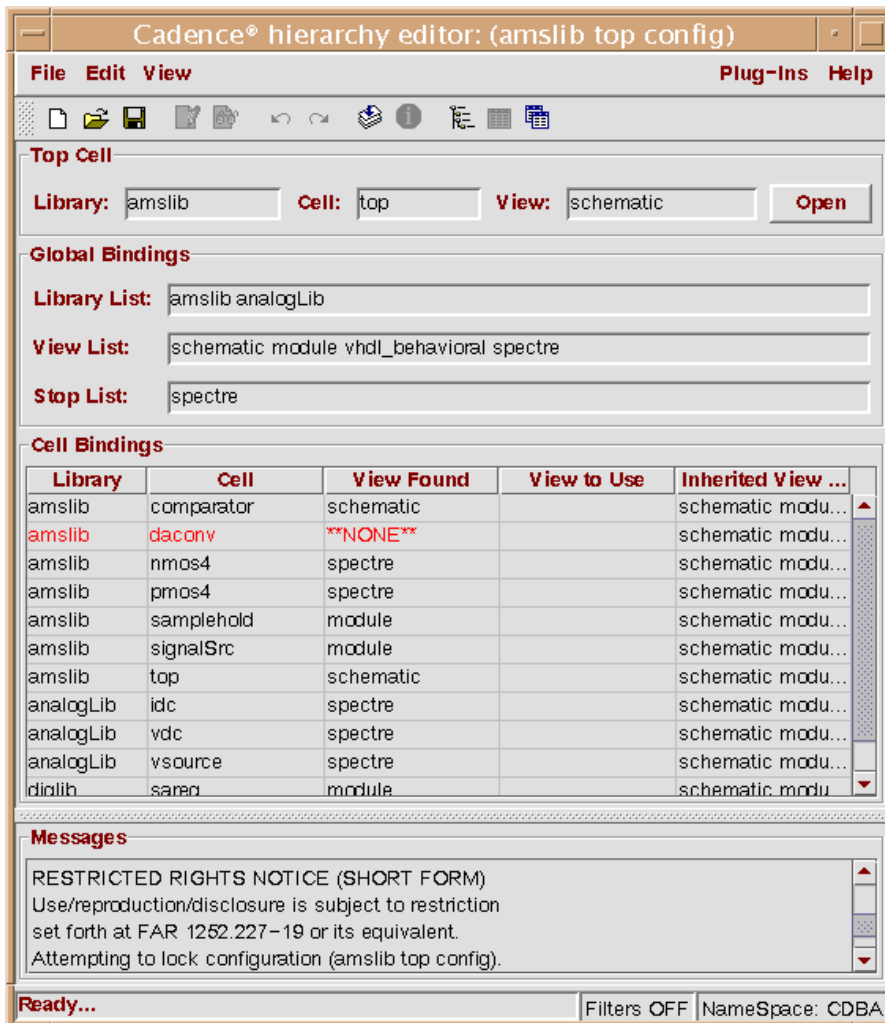
The top schematic is shown below:



## Virtuoso AMS Environment User Guide

### Quick-Start Tutorial

The top config view data is displayed in the Cadence hierarchy editor, as shown below:



Notice that analogLib is included in the *Library List* field. This library must be in the library list because one of the components in the library, vsource, is used in a text cellview.

7. In the *View List* field, add the view daconv\_behav.

The added view contains the entity and architecture for the daconv cell. After this change, the *View List* field displays

```
schematic module vhdl_behavioral spectre daconv_behav
```

8. Choose *View – Update (Needed)*, which updates the configuration.

The daconv cell is bound to the daconv\_behav view.



9. If the Update Sync-up message appears, make sure that the `amslib top config cellview` is selected, then click **OK**.

## Netlisting and Compiling

In AMS Designer, netlisting translates the design into equivalent Verilog-AMS modules. In the compiling step, those Verilog-AMS modules are then converted into internal representations of the design that are optimized for efficient processing.

You can automatically netlist and compile a cellview during a *Check and Save* operation. You can also netlist and compile an entire design with AMS Design Prep. These procedures illustrate different ways of using the AMS environment.

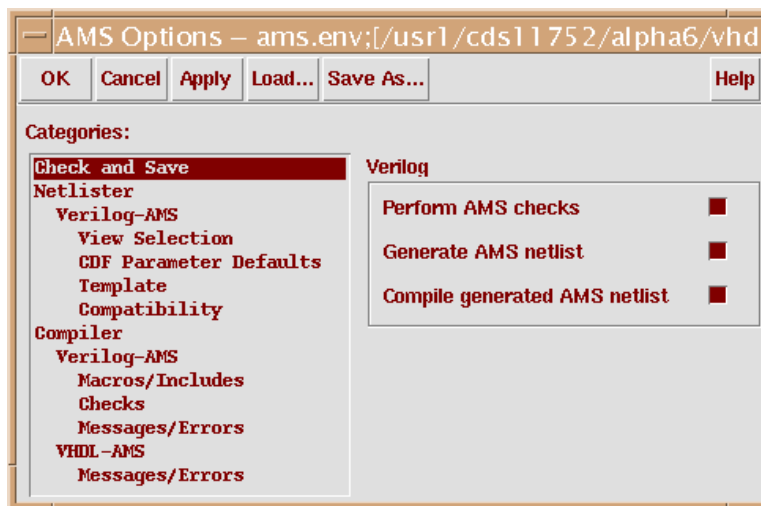
In the tutorial sections that follow, you first netlist the `top` schematic during a *Check and Save* operation, an exercise that illustrates automatic netlisting. Then you use AMS Design Prep to netlist the other blocks of the design.

## Automatic Netlisting and Compiling

When you use automatic netlisting, you can focus on developing your schematics because the manual netlisting step is eliminated. To turn on automatic netlisting,

1. In the CIW, choose *Tools – AMS – Options*.

The AMS Options form appears.



The defaults in the AMS Options form are defined in the `ams.env` file.

2. If necessary, in the *Categories* field, choose *Check and Save*.

## Virtuoso AMS Environment User Guide

### Quick-Start Tutorial

---

3. Turn on *Perform AMS checks*, *Generate AMS netlist*, and *Compile generated AMS netlist*, then click **OK**.

These choices ensure that cellviews are automatically checked, netlisted, and compiled when you click the *Check and Save* button in the Virtuoso Schematic Editing window.

4. Netlist the `top` schematic by clicking the *Check and Save* button in the Virtuoso Schematic Editing window:



AMS netlister messages are sent to the CIW.

The netlist for the `top` schematic is saved in

```
./AMS_lib/amsLib/top/schematic/verilog.vams
```

The netlist looks like this:

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

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

module top ( );
electrical dacOut;
wire [0:7] b;

vsource #(.type("dc"), .dc(5)) (*
integer library_binding = "analogLib"; *) V0 ( cds_globals.\vdd! ,
cds_globals.\gnd! );
vsource #(.type("dc"), .dc(0)) (*
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 ( .result(
compOut ), .clkSig( clkSig ), .b6( b[6] ), .b5( b[5] ), .b0( b[0] ), .trigger(
endOfConv ), .b2( b[2] ), .endOfConv( endOfConv ), .b1( b[1] ), .b3(
b[3] ), .b4( b[4] ), .b7( b[7] ) );

daconv (* integer library_binding = "amslib"; *) I4 ( .b6( b[6] ), .b5(
b[5] ), .b0( b[0] ), .compSig( dacOut ), .b2( b[2] ), .b1( b[1] ), .b3(
b[3] ), .b4( b[4] ), .b7( b[7] ) );

signalSrc (* integer library_binding = "amslib"; *) I0 .sig( inSig );

comparator (* integer library_binding = "amslib"; *) I2 ( .inp(
holdSig ), .inn( dacOut ), .out( compOut ) );

samplehold (* integer library_binding = "amslib"; *) I1 ( .holdSig(
holdSig ), .trigger( endOfConv ), .inSig( inSig ) );

endmodule
```

## Netlisting and Compiling with AMS Design Prep

In the previous section, you netlisted the `top` schematic by doing a check and save of the schematic. In this section, you netlist and compile the rest of your design by using AMS Design Prep. (In this tutorial, there is only one more schematic, the `comparator`, to netlist, but you can use AMS Design Prep to netlist many cellviews at once.)

### Preparing to Run AMS Design Prep

Before you can use AMS Design Prep, you must first install the *AMS* menu entry and then specify a run directory and the location of two files used by AMS Designer.

1. In the hierarchy editor, choose *Plug-Ins – AMS*.

The menu bar changes to include the *AMS* entry, as shown below. The AMS menu contains controls for the AMS environment and simulator.



2. Choose *AMS – Run Directory*.

## Virtuoso AMS Environment User Guide

### Quick-Start Tutorial

---

The AMS Run Directory form appears.

The screenshot shows the 'AMS Run Directory' dialog box. It has a title bar with a minus sign. The main area contains the following elements: a 'Run directory:' label followed by a text field containing '/usr1/cds11752/alpha6/vhdltestdir/SAR\_A2D/top\_run' and a 'Browse' button; an 'Existing run directories:' label followed by a dropdown menu; a checkbox labeled 'Always use this run directory for this configuration'; a section titled 'Copy Run Directory Setup' containing a 'From this run directory:' label with a text field and a 'Browse' button, followed by another 'Existing run directories:' label with a dropdown menu; and a 'Files to copy:' section with two empty list boxes, 'Copy:' and 'Do not copy:', and two arrow buttons between them. At the bottom are four buttons: 'OK', 'Cancel', 'Apply', and 'Help'.

3. In the *Run directory* field, change `top_run` to `tutorial_run`. (Leave the rest of the path intact.)

By default, the run directory for this design is named `top_run`, but, as illustrated here, you can give the directory any name you choose.

4. Click *OK*.

The other entries in the *AMS* menu become active.

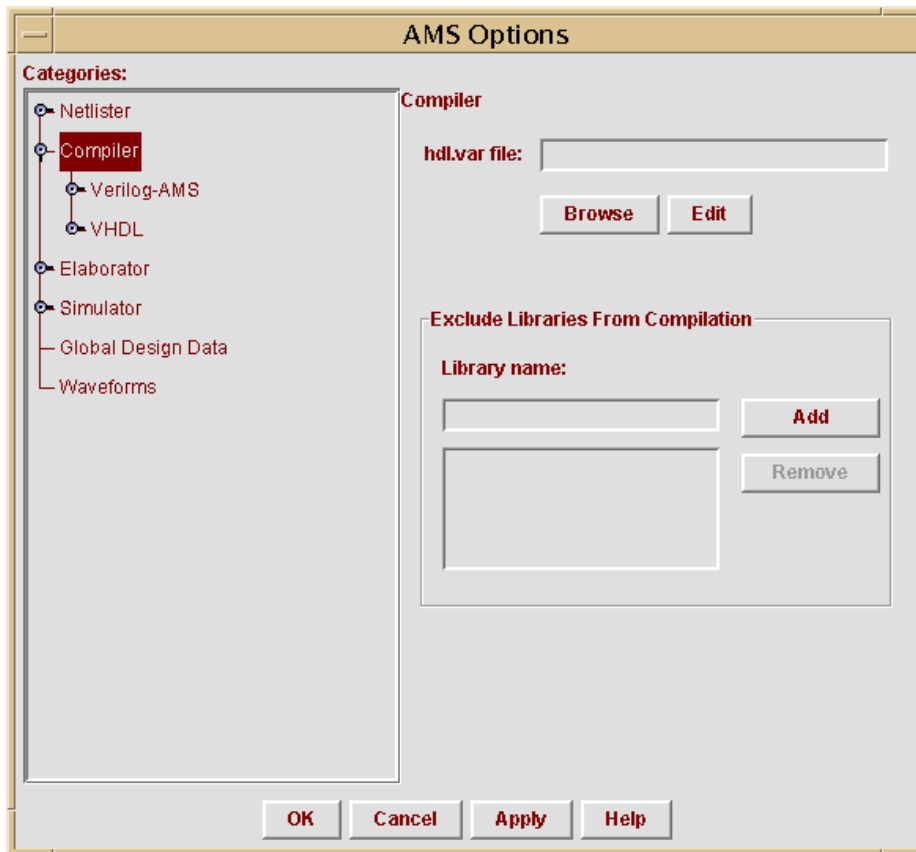
5. In the hierarchy editor, choose *AMS – Options – Compiler*.

## Virtuoso AMS Environment User Guide

### Quick-Start Tutorial

---

The AMS Options form appears, displaying the *Compiler* pane.



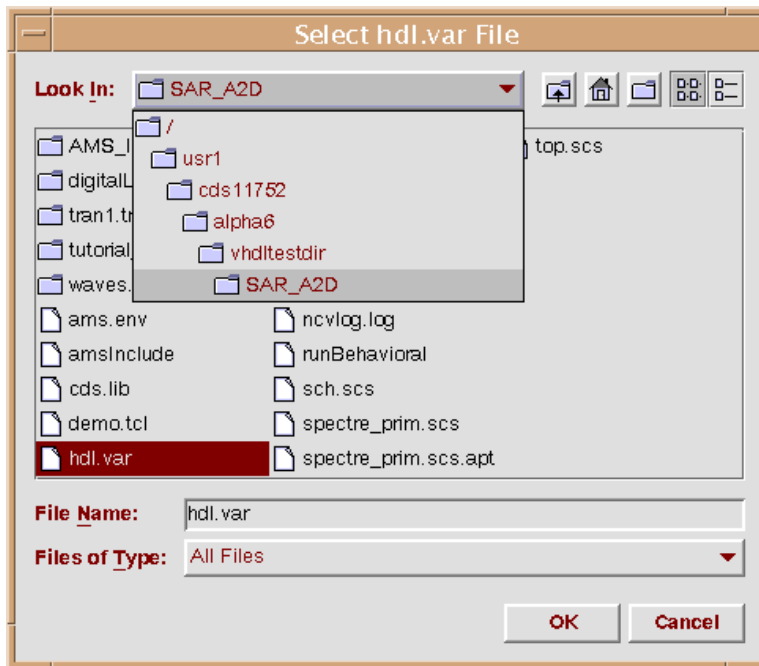
6. Click the *Browse* button.

## Virtuoso AMS Environment User Guide

### Quick-Start Tutorial

---

The Select hdl.var File browser appears.



7. Select the `hdl.var` file in the directory where you typed the `icms` command to open the CIW.
8. Click **OK**.

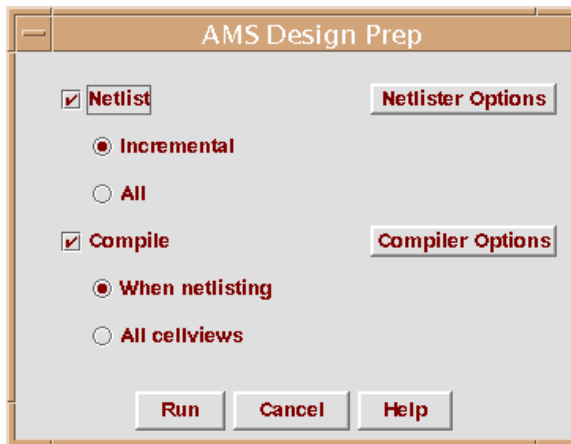
The browser closes. Your `hdl.var` selection appears in the *hdl.var file* field of the *Compiler* pane.

9. Click **OK**.

### Running AMS Design Prep

1. In the hierarchy editor, choose *AMS – Design Prep*.

The AMS Design Prep form appears.



**2. Turn on *Netlist* and *Incremental*.**

These choices tell AMS Design Prep to netlist only cellviews that have changed or that have not been netlisted previously. This is an efficient way to use AMS Design Prep during the cycles of development because only blocks that need netlisting are netlisted.

**3. Turn on *Compile* and *When netlisting*.**

These choices tell AMS Design Prep to compile only cellviews that are being netlisted in the current AMS Design Prep run.

**4. Click *Run*, which runs AMS Design Prep.**

If the following message appears, click **Yes**.

```
The cds_globals module appears to have been
modified outside of AMS Prep.... Continue
processing and overwrite the cds_globals module?
```

If the following message appears, click **Yes**.

```
The configuration has been modified.
```

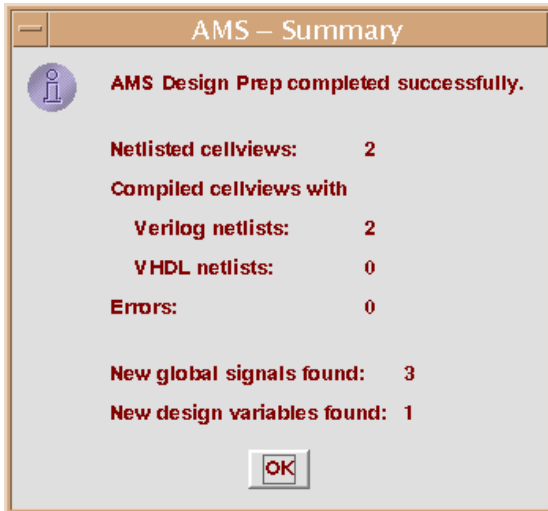
```
Do you want to update and save the configuration
before running Design Prep?
```

## Virtuoso AMS Environment User Guide

### Quick-Start Tutorial

---

After AMS Design Prep finishes running, the Summary window appears.



5. Read the Summary window and click *OK*.

Depending on the version of AMS Designer that you are running, the number of netlisted and compiled cellviews might differ from that shown above. This variation reflects different approaches to handling the files used by AMS Designer.

#### Viewing the Netlist

To view the newly created netlist for the `comparator` schematic,

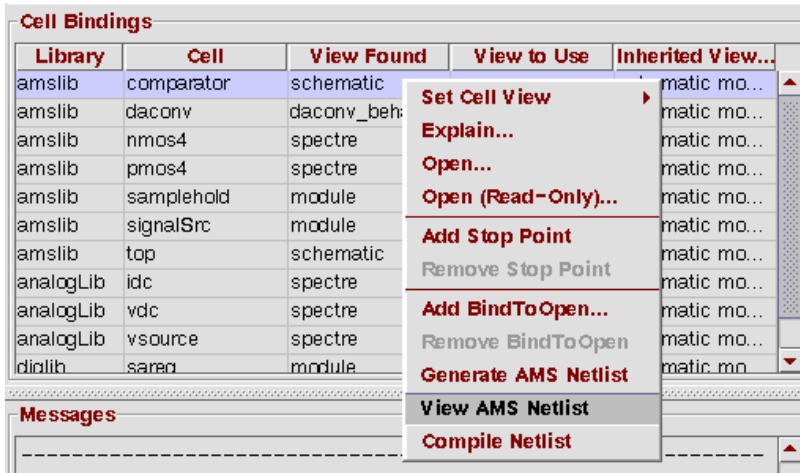
1. In the Cadence hierarchy editor *Cell Bindings* section, find the row for the `comparator` cell.



## Virtuoso AMS Environment User Guide

### Quick-Start Tutorial

2. Right-click on the row, to display the pop-up menu.



3. Choose *View AMS Netlist*.

The netlist appears in a window.

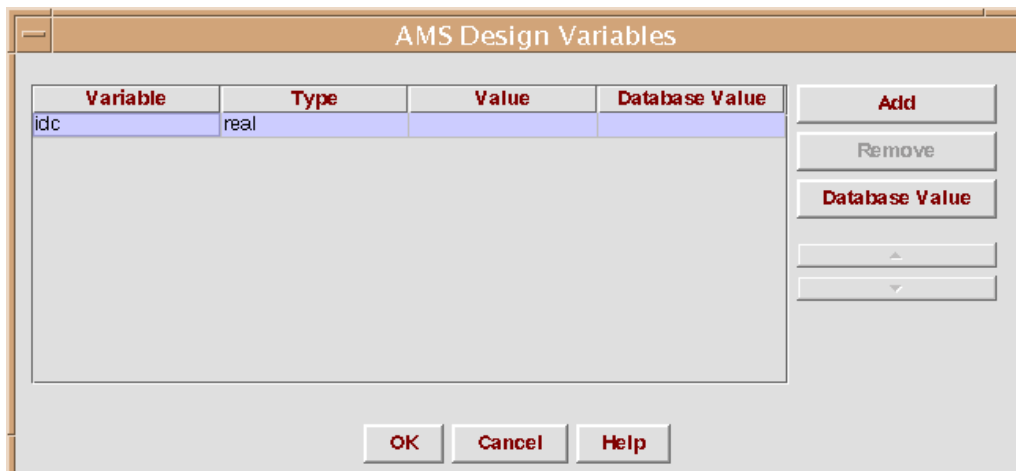
4. When you are done viewing the netlist, quit from the netlist without saving.

#### Setting the Value of a Design Variable

You might have noted in the AMS Design Prep - Summary window the line saying that one new design variable was found in the design. In this section of the tutorial, you assign a value to that variable.

1. In the hierarchy editor, choose *AMS – Design Variables*.

The AMS Design Variables form appears.



## Virtuoso AMS Environment User Guide

### Quick-Start Tutorial

---

2. In the *Value* cell for the `idc` variable, type `5u`.

This value helps characterize a power supply for the design.

3. Click *OK*.

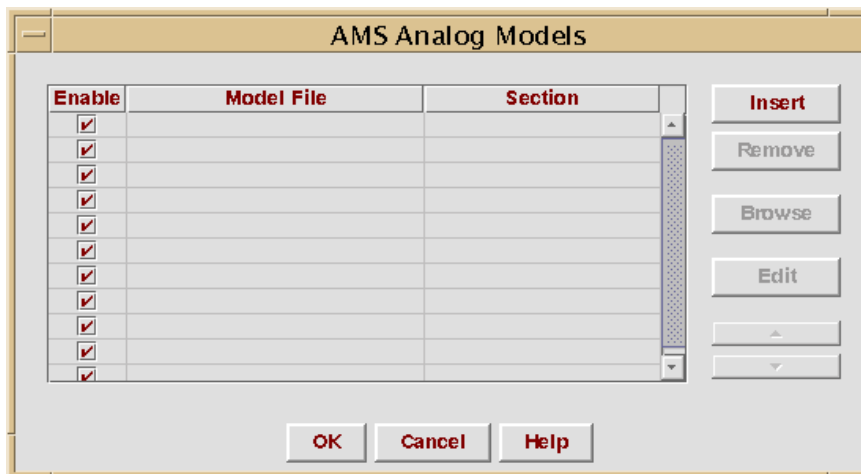
AMS Designer recreates and recompiles the `cds_globals` module so that the `idc` value is available to the design.

#### Setting Up the Analog Model File

You might have noted in the hierarchy editor that the configuration for this design includes cells called `nmos4` and `pmos4`. In this section of the tutorial, you specify the location of the model file that characterizes those components.

1. In the hierarchy editor, choose *AMS – Analog Models*.

The AMS Analog Models form appears.



2. Highlight the top row, then click *Browse*.

The Select Analog Model File browser appears.

3. Select the `spectre_prim.scs` file in the directory where you typed the `icms` command to open the CIW.
4. Click *OK* to close the browser.

Your selection appears in the *Model File* column of the AMS Analog Models form.

5. With the same row highlighted, click *Edit*.

The model file opens so you can review the values that characterize the `nmos4` and `pmos4` components. When you are done looking, close the file without saving.

6. In the AMS Analog Models form, click **OK**.

## Elaborating and Simulating the Design

Now that you have netlisted and compiled the entire design, you are ready to elaborate and simulate it. The AMS elaborator, `ncelab`, resolves disciplines, inserts interface elements, and creates a simulation snapshot of your design. The AMS simulator, `ncsim`, then runs the simulation.

Some of the steps for this tutorial have been done for you. At this point in your own designs, you will probably want to choose *AMS – Global Signals* from the hierarchy editor menu and ensure that the form is filled in as necessary.

## Specifying a Tcl File to Set SimVision Breakpoints

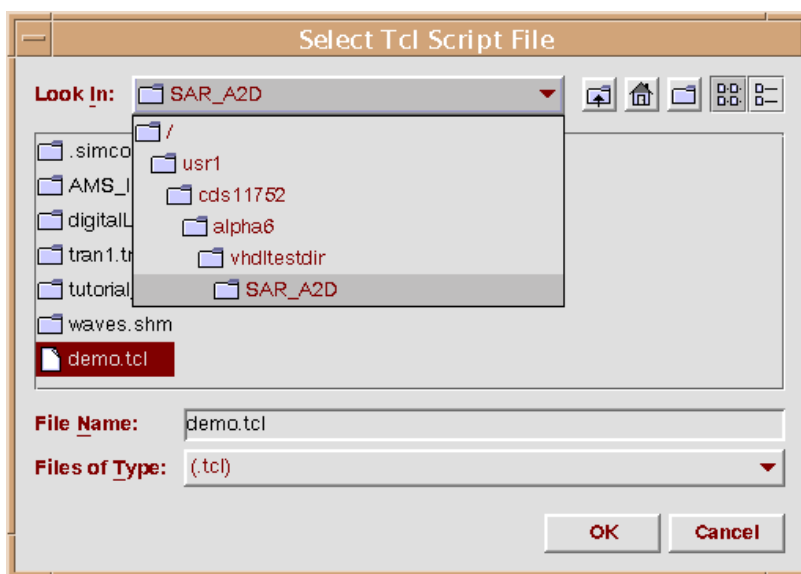
To prepare for the elaboration and simulation,

1. In the hierarchy editor, choose *AMS – Options – Simulator*.

The AMS Options form appears, displaying the *Simulator* pane.

2. Click the *Browse* button.

The Select Tcl Script File browser appears.



## Virtuoso AMS Environment User Guide

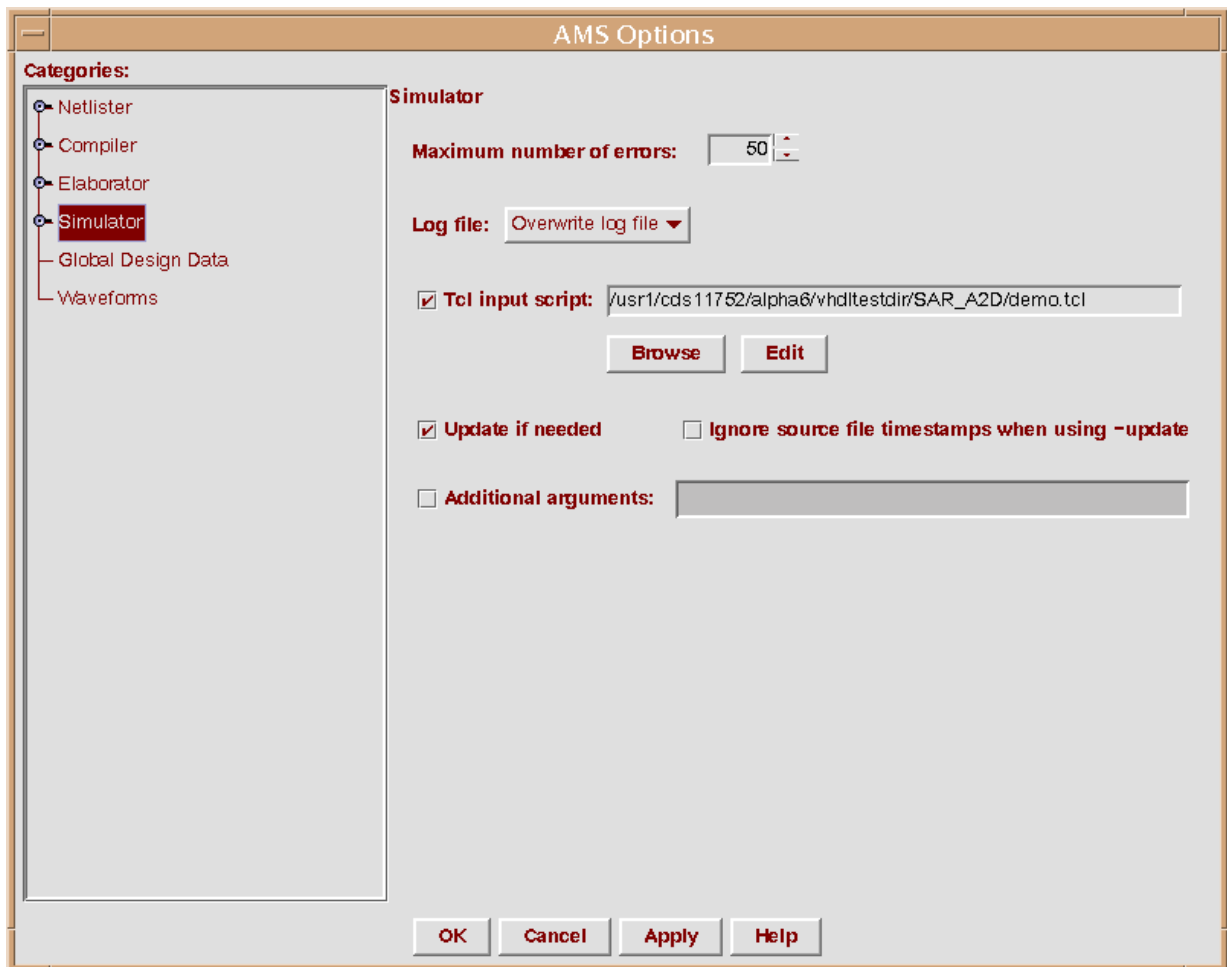
### Quick-Start Tutorial

---

3. Select the `demo.tcl` file in the directory where you typed the `icms` command to open the CIW.
4. Click *OK* to close the browser.

The full path to your `demo.tcl` file appears in the *Tcl input script* field of the *Simulator* pane.

This file sets a breakpoint to be used when SimVision runs. After these steps, the *Simulator* pane looks like this:



5. Click *Apply*.

The AMS Options form, which you use in the next section of the tutorial, remains open.

## Specifying the Simulation Analysis

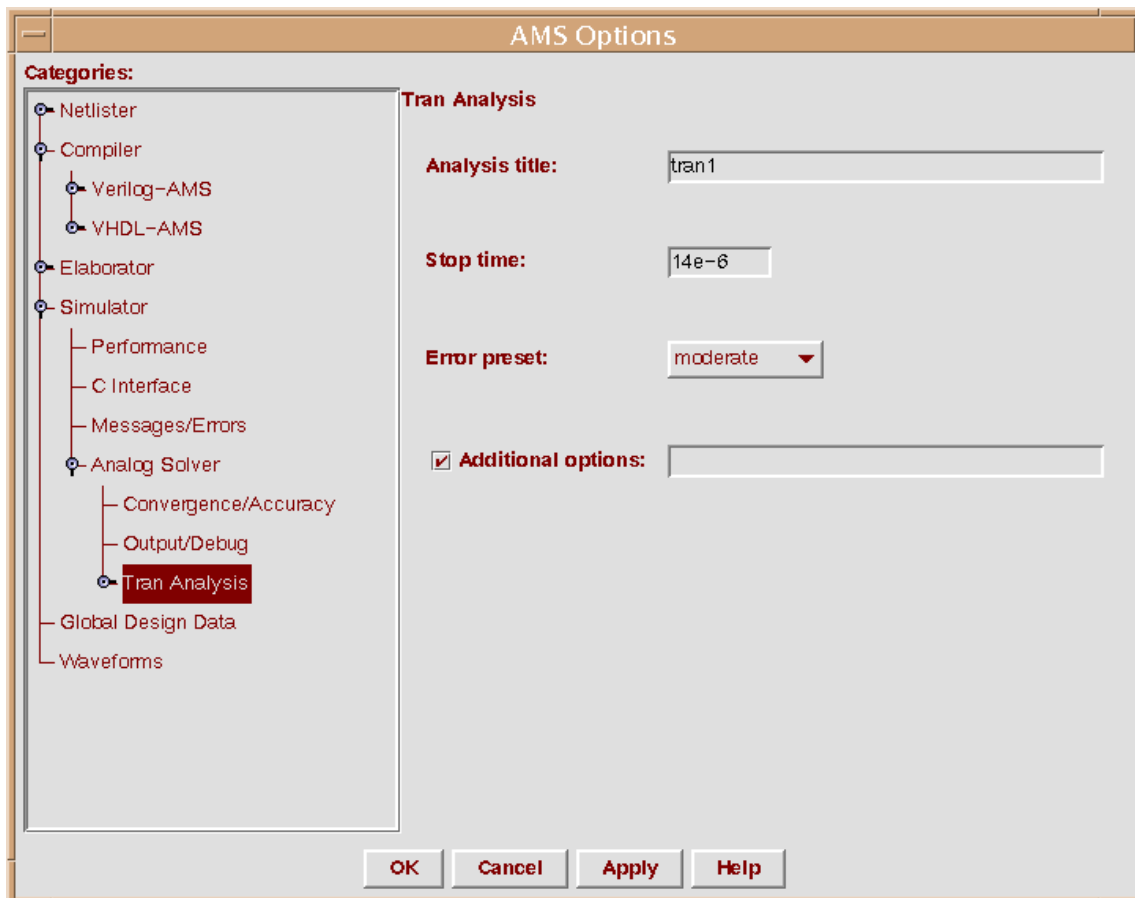
The steps in this section illustrate how you can use the graphical user interface to specify the simulation analysis.

1. In the AMS Options form *Categories* pane, click *Simulator – Analog Solver – Tran Analysis*.

The *Tran Analysis* pane appears.

2. In the *Analysis title* field, type `tran1`.
3. In the *Stop time* field, type `14e-6`.

After these steps, the *Tran Analysis* pane looks like this:



4. In the *Categories* pane, click *Simulator – Analog Solver – Tran Analysis – Convergence/Accuracy*.

The *Tran Convergence/Accuracy* pane appears.

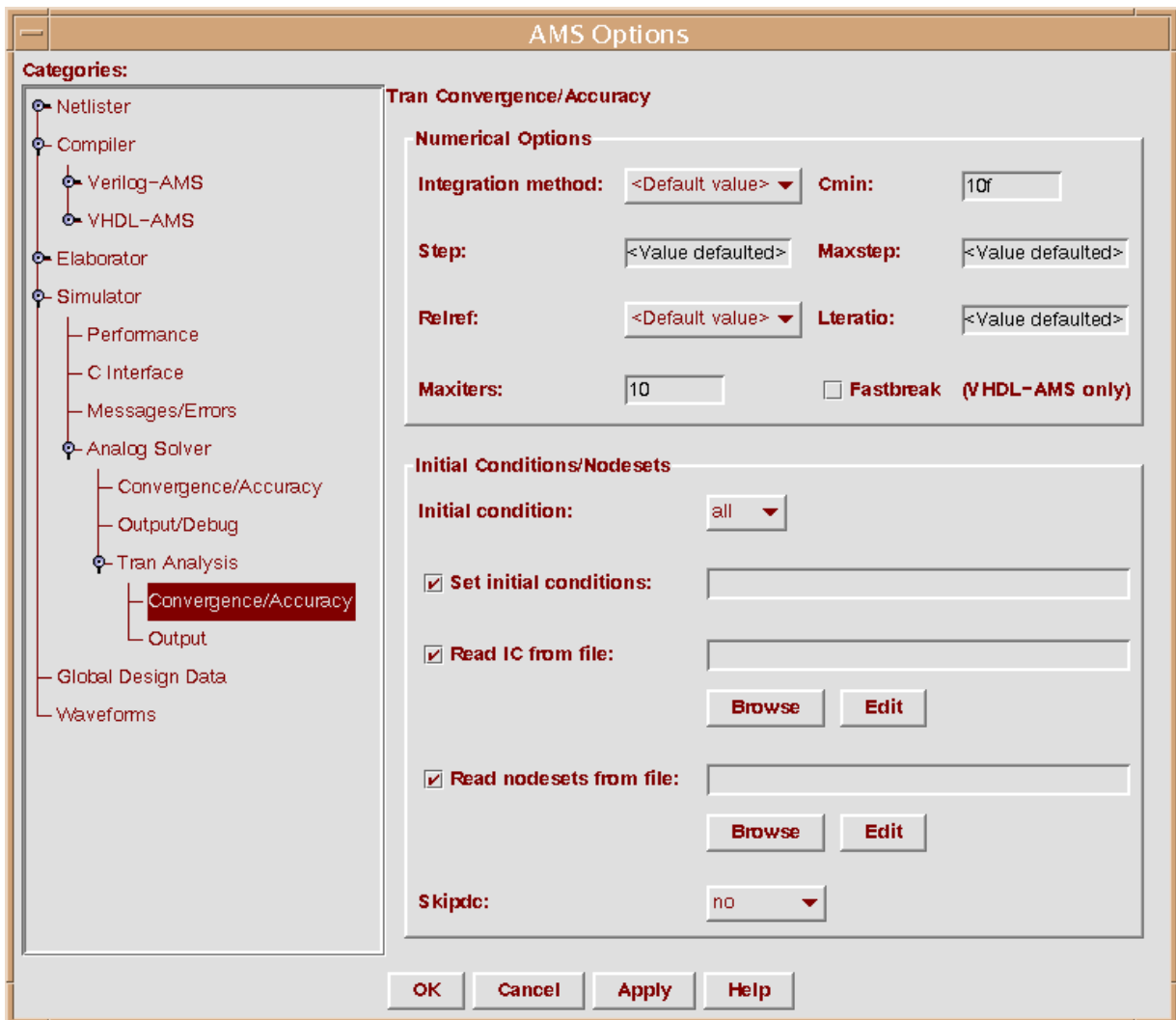
## Virtuoso AMS Environment User Guide

### Quick-Start Tutorial

5. In the *Maxiters* field, type 10.

6. In the *Cmin* field, type 10f.

After these steps, the *Tran Convergence/Accuracy* pane looks like this:



7. Click **OK**.

The AMS Options form closes.

### Specifying Values to Save and Plot

This section of the tutorial illustrates how to designate the information to be saved and plotted during the simulation.

## Virtuoso AMS Environment User Guide

### Quick-Start Tutorial

---

1. In the hierarchy editor, choose *AMS – Save/Plot*.

The AMS Save/Plot form appears.

	Name	Object	Type	Depth	Database	Save	Plot
1	/			all	waves	<input checked="" type="checkbox"/>	<input type="checkbox"/>
2						<input type="checkbox"/>	<input type="checkbox"/>
3						<input type="checkbox"/>	<input type="checkbox"/>
4						<input type="checkbox"/>	<input type="checkbox"/>
5						<input type="checkbox"/>	<input type="checkbox"/>
6						<input type="checkbox"/>	<input type="checkbox"/>
7						<input type="checkbox"/>	<input type="checkbox"/>
8						<input type="checkbox"/>	<input type="checkbox"/>
9						<input type="checkbox"/>	<input type="checkbox"/>

Select from

**Schematic**

**Navigator**

☒ **Save only**

☐ **Save and plot**

**Databases**

**Remove**

**Plot** **Cancel** **Done** **Help**

The first row in the table is a default setting that saves waveforms for the entire design. The tutorial design is small but in a larger design you might want to remove the check mark from the *Save* column for this default row and use other rows to save only the information you need.

2. Turn on *Save and plot*.

With this selection, rows that are added to the table in the future have check marks in both the *Save* and *Plot* columns. Having check marks in these two columns means that the waveforms are plotted as soon as they are calculated during simulation.

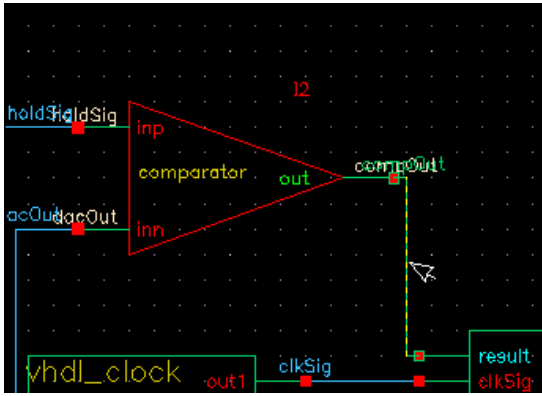
3. Click *Schematic*.

The Virtuoso Schematic Editing window becomes active. It has a message at the bottom indicating that you can select objects to be saved and plotted.

## Virtuoso AMS Environment User Guide

### Quick-Start Tutorial

4. In the Schematic Editing window, select the `compOut` net on the right side of the comparator.



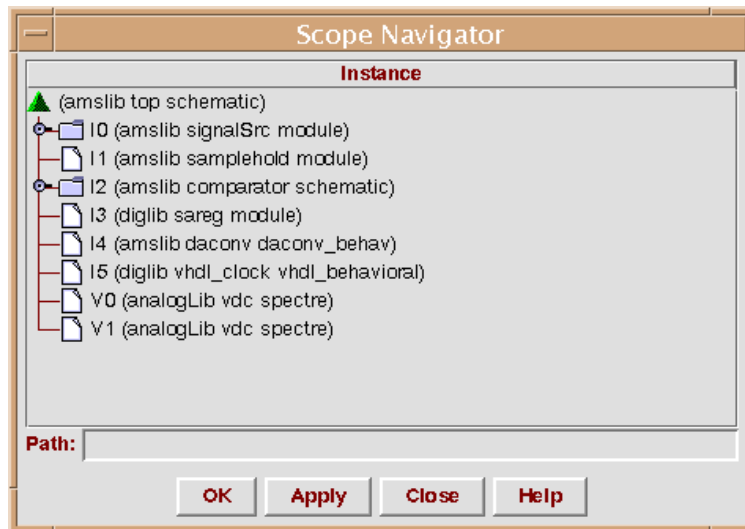
A new row appears for the `/compOut` net in the AMS Save/Plot form. Notice how both the *Save* and *Plot* columns have check marks.

5. Press `Escape` while you are still in the Virtuoso Schematic Editing window to end the selection of objects.

The object selection message disappears from the Schematic Editing window and from the AMS Save/Plot form.

6. In the AMS Save/Plot form, click *Navigator*.

The Scope Navigator form appears, displaying a hierarchical view of the design.



7. Highlight the `I3` instance.

The path to the instance appears in the *Path* field.



## Virtuoso AMS Environment User Guide

### Quick-Start Tutorial

---

#### 8. Click *OK*.

The Scope Navigator form closes. A new row appears in the AMS Save/Plot form with the path to the instance and default information for a new probe.

After these steps, the AMS Save/Plot form looks like this:

	Name	Object	Type	Depth	Database	Save	Plot
1	/			all	waves	<input checked="" type="checkbox"/>	<input type="checkbox"/>
2	/compOut			N/A	waves	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
3	/13			1	waves	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
4						<input type="checkbox"/>	<input type="checkbox"/>
5						<input type="checkbox"/>	<input type="checkbox"/>
6						<input type="checkbox"/>	<input type="checkbox"/>
7						<input type="checkbox"/>	<input type="checkbox"/>
8						<input type="checkbox"/>	<input type="checkbox"/>
9						<input type="checkbox"/>	<input type="checkbox"/>

#### 9. Click *Done*.

The AMS Save/Plot form closes.

## Elaborating and Simulating the Design

1. In the hierarchy editor window, choose *AMS – Run Simulation*.

## Virtuoso AMS Environment User Guide

### Quick-Start Tutorial

---

The AMS Run Simulation form appears.

The screenshot shows the 'AMS Run Simulation' dialog box. It contains several sections for configuration:

- Configuration:** Library: amslib, Cell: top, View: config
- Global Design Data Module:** Library: amslib, Cell: cds\_globals, View: top\_config
- Connect Rules:** Library: (empty), Cell: ConnRules\_5V\_full, View: (empty)
- Simulation Snapshot:** Library: (empty), Cell: top, View: ams1081969769714

Below these sections are buttons for 'Elaborator Options', 'Simulator Options', and 'Analog Models Setup'. To the right of these is a field for 'Tran analysis stop time:' set to '14e-6'. Below these is a 'Run Mode:' dropdown menu set to 'GUI'. At the bottom of the configuration section are two checked checkboxes: 'Run Elaborator' and 'Run Simulator'. Below these is a 'Save' button and a text field containing 'NC commands to runElabSim script'. At the very bottom of the dialog are four buttons: 'Run', 'Cancel', 'Apply', and 'Help'.

Depending on the simulator you are using, the *Cell* field for *Connect Rules* specifies either *mixedsignal* or *ConnRules\_5V\_full*. These names are associated with the different sets of connect modules shipped with various releases of the simulator.

The information specified in this form is passed as command options to the elaborator and simulator.

2. Be sure that both *Run Elaborator* and *Run Simulator* are turned on.
3. Click *Run* to elaborate the design and start the AMS simulator in GUI (interactive) mode, using the SimVision windows.

## Virtuoso AMS Environment User Guide

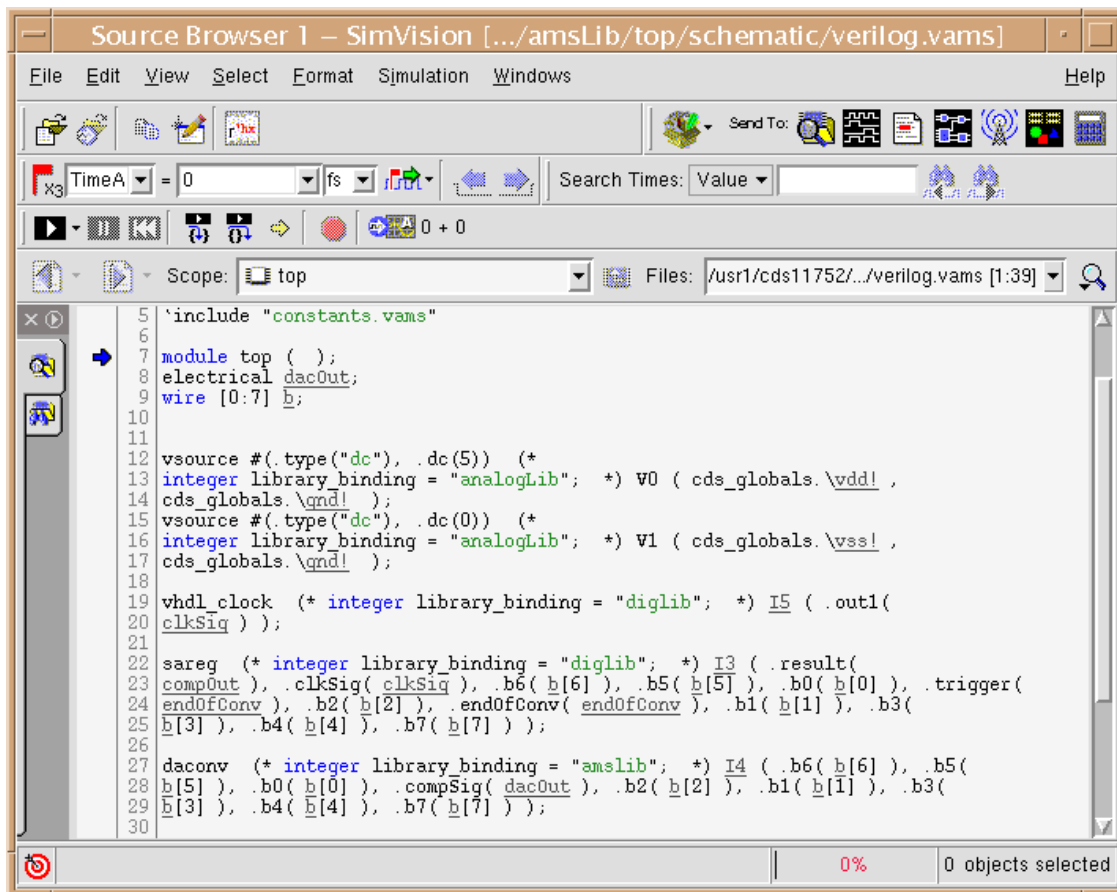
### Quick-Start Tutorial

#### Using the SimVision Windows

As the simulator starts, it opens the Console, Design Browser, and Waveform windows. This tutorial exercises only some of the basic features of SimVision. For detailed information, see *SimVision User Guide*.

1. In the Console window, choose *File – Close Window*.  
(The tutorial does not use this window.)
2. In the Design Browser window, choose *Windows – New – Source Browser*.

The Source Browser appears, showing the Verilog-AMS netlist for the `top` design.



#### Finding the Values of Objects in the Design

You can hover over objects such as signals and see their values. For example,

- In the Source Browser, hold your cursor over `compOut` at about line 23.

Initially this analog value is zero, shown as follows:

```
0 V
waves::top.compOut
Probed to databases: waves from "simulator"
```

## Traversing the Hierarchy in the Source Browser

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

- In the Source Browser, double click on `I5` to descend into the `vhdl_behavioral` view of the VHDL clock.

To return to `top`,

- Click the *Scope Up* button, which is just to the right of the *Scope* field.



## Running the Simulation

The Waveform window already contains the name of the `compOut` net and of all the nets associated with instance `I3` because you used the AMS Save/Plot form earlier to specify them. You are now ready to run the simulation and plot the waveform.

1. In the Waveform window, click the *Run* button



and run the simulation until it ends at about simulation time 14000ns. (You have to click *Run* more than once to get past the breakpoint.)

The waveforms appear as soon as they are calculated and continue marching during the remainder of the simulation.

2. Choose *View – Zoom – Full X* and *View – Zoom – Full Y* to fit the waveforms into the available room in the Waveform window.
3. Position the cursor over the name of the `result` waveform, right-click to open the pop-up menu, and choose *Cut*.

The `result` waveform disappears from the Waveform window.

4. Choose *Edit – Paste*.

## Virtuoso AMS Environment User Guide

### Quick-Start Tutorial

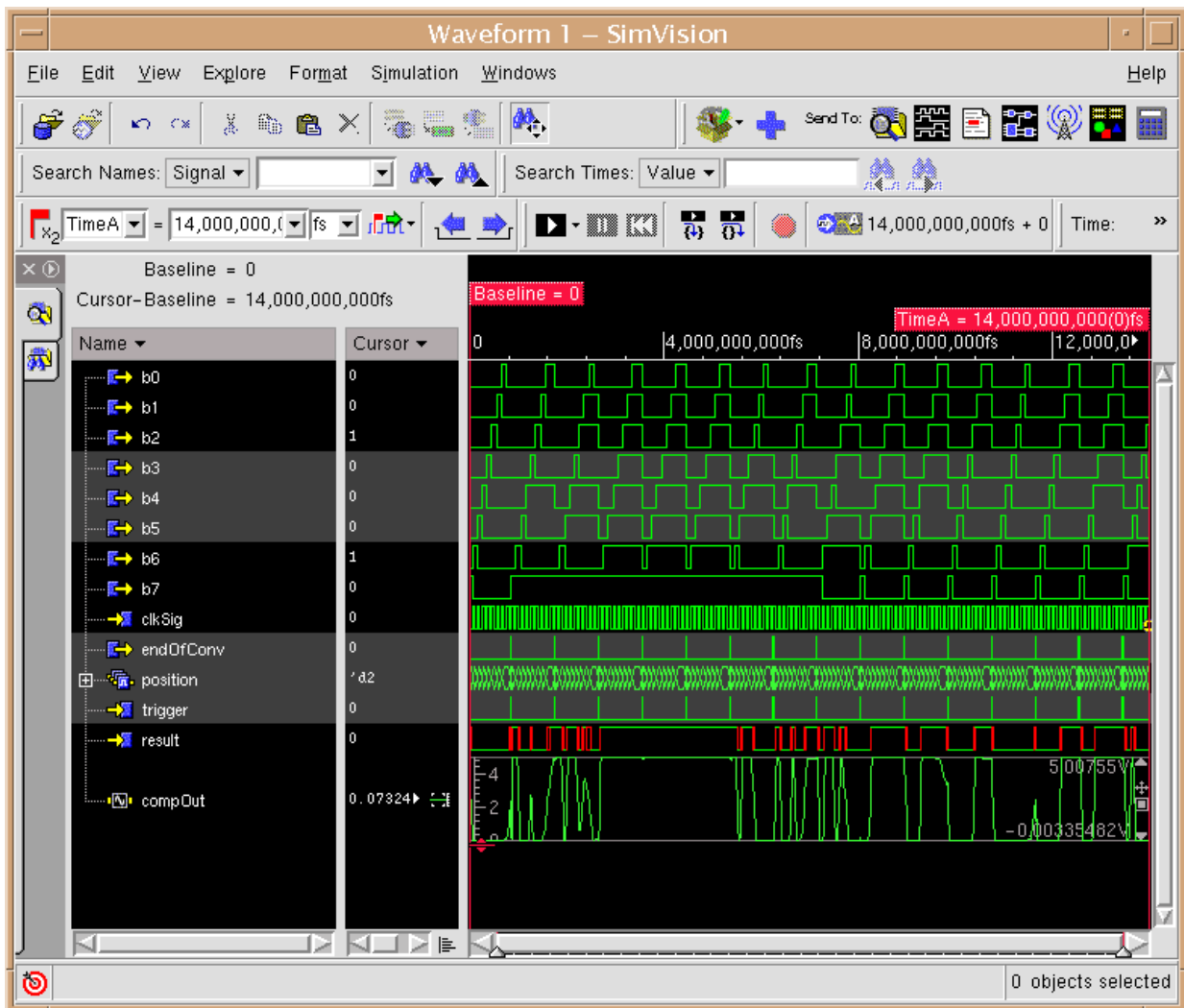
The `result` waveform reappears at the bottom of the window.

5. Position the cursor over the name of the `compOut` waveform, right-click to open the pop-up menu, and choose *Cut*.

The `compOut` waveform disappears from the Waveform window.

6. Choose *Edit – Paste*.

Now the `result` and `compOut` waveforms are together at the bottom of the window.



7. Compare the analog waveform of `compOut` with the corresponding digital waveform, `result`.

These two signals are connected by an automatically inserted connect module.

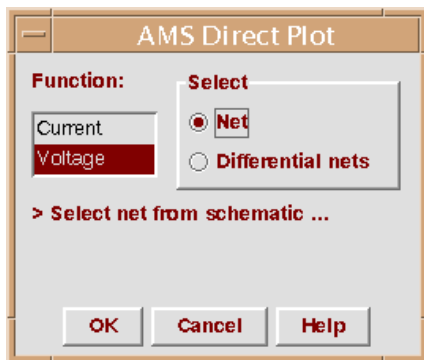
8. When you are done examining the waveforms, choose *File – Exit SimVision*.

## Plotting Waveforms After the Simulation Ends

In this section, you select and plot waveforms that were saved during the simulation. You might do this, for example, if you decide to replot waveforms you have already seen or if you simulated in batch mode and now want to see the results.

1. In the hierarchy editor, choose *AMS – Direct Plot*.

The AMS Direct Plot form appears.



2. Ensure that *Function* is set to *Voltage* and that *Select* is set to *Net*.

These choices mean that the voltage is plotted each time you select a net from the schematic. The message

Select net from schematic ...

appears at the bottom of the Virtuoso Schematic Editing window.

3. In the Virtuoso Schematic Editing window, select the `dacOut` net from the schematic.

A plot of the voltage for the net appears in a Waveform window.

4. In the Waveform window, choose *View – Zoom – Full X*.

5. From the schematic, select the `inSig` net.

The waveform is added to the Waveform window.

6. In the AMS Direct Plot form, change *Select* to *Differential nets*.

The message

Select positive net from schematic ...

appears at the bottom of the Schematic Editing window.

7. From the schematic, select the `dacOut` net.

The message

## Virtuoso AMS Environment User Guide

### Quick-Start Tutorial

Select negative net from schematic ...

appears at the bottom of the Schematic Editing window.

#### 8. Select the inSig net.

A waveform representing the difference of the voltages of the two nets is added to the Waveform window. After these steps, the window looks like this:



## Summary

In this tutorial, you used the AMS environment and simulator to netlist, compile, elaborate, and simulate the `top` design configuration, which contains mixed-signal and mixed-language designs. You used the CIW, hierarchy editor, AMS netlister, AMS compiler, AMS Design Prep, ncelab, and ncsim (with the Cadence SimVision windows) to complete the necessary tasks.

# **Virtuoso AMS Environment User Guide**

## **Quick-Start Tutorial**

---



---

## Setting Up the AMS Environment

---

This chapter contains the following sections:

- [Overview](#) on page 58
- [The hdl.var File](#) on page 58
- [The ams.env Files](#) on page 59
- [Specifying the Text Editor to Use](#) on page 60
- [Specifying Fonts for the Cadence Hierarchy Editor](#) on page 60
- [Preparing to Use AMS Designer from the Hierarchy Editor](#) on page 62

## Overview

Several configuration files help you manage your data and control the operation of the tools used in the AMS environment. Among them are

- `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 “Setting Up Your Environment” chapter of the *Virtuoso AMS Simulator User Guide*.

- `hdl.var`

Defines variables that affect the behavior of tools and utilities. For more information, see [“The hdl.var File”](#) on page 58.

- `setup.loc`

Specifies the search order that tools and utilities use when searching for the `cds.lib`, `hdl.var`, and `ams.env` files. For more information, see [“The setup.loc File”](#) in the “Setting Up Your Environment” chapter of the *Virtuoso AMS Simulator User Guide*.

- `ams.env`

Specifies the basic behavior of AMS Design Prep and the AMS netlister. For more information, see [“The ams.env Files”](#) on page 59.

The AMS environment and the AMS simulator operate on and store data in libraries that are organized according to a Library.Cell:View (L.C:V) approach. For more information, see [“The Library.Cell:View Approach”](#) in the “Setting Up Your Environment” chapter of the *Virtuoso AMS Simulator User Guide*.

## The hdl.var File

The `hdl.var` file is an ASCII text file that contains

- Configuration variables, which determine how your design environment is configured. These include
  - ❑ Variables that you can use to specify the work library where the compiler stores compiled objects and other derived data. For Verilog-AMS, you can use the `LIB_MAP` or `WORK` variables.
  - ❑ For Verilog-AMS, variables (`LIB_MAP`, `VIEW_MAP`, `WORK`) that you can use to specify the libraries and views to search when the elaborator resolves instances.

## Virtuoso AMS Environment User Guide

### Setting Up the AMS Environment

---

- Variables that allow you to specify compiler, elaborator, and simulator command-line options and arguments.
- Variables that specify the locations of support files and invocation scripts.

For example, the `hdl.var` file for the included tutorial contains the following:

```
softinclude $AMSHOME/tools/inca/files/hdl.var
define WORK amslib
define NCVLOGOPTS -linedebug
define VIEW_MAP ($VIEW_MAP, .vs => shell)
```

You can have more than one `hdl.var` file. For example, you can have a project `hdl.var` file that contains variable settings used to support all your projects and local `hdl.var` files, located in specific design directories, that contain variable settings specific to each project, such as the setting for the `WORK` variable.

For more information about the `hdl.var` file, see [“The hdl.var File”](#) in the “Setting Up Your Environment” chapter of the *Virtuoso AMS Simulator User Guide*.

## The ams.env Files

The `ams.env` files are ASCII files that specify the basic behavior of AMS Design Prep and the AMS netlister. Cadence supplies a default `ams.env` file, which you can change to meet your needs. Perhaps the most convenient way to change the settings is to use the graphical interfaces available by choosing either *Tools – AMS – Options* from the CIW menu or *AMS – Options* from the Cadence hierarchy editor menu. For information about using these graphical interfaces, see [“Specifying the Behavior of the Netlister and Compilers”](#) on page 74. For detailed descriptions of the variables used in the `ams.env` file, see [Appendix A, “Variables for ams.env Files.”](#)

The Cadence default `ams.env` file is `your_install_directory/share/cdssetup/amsDirect/ams.env`. You can also have other `ams.env` files tailored to your needs. The AMS tools search for such files using the search mechanism described in the [“Cadence Setup Search File: setup.loc”](#) chapter of the *Cadence Application Infrastructure User Guide*.

Day-to-day changes in the `ams.env` files result from changes you make in the graphical interfaces, but if you need to create or edit an `ams.env` file directly, be aware of the following:

- Lines beginning with a semi-colon are considered comments.
- AMS Designer never writes comments to the `ams.env` files that it generates or overwrites. That means that if you add a comment to an `ams.env` file that AMS Designer reads and then writes, the comment disappears. For example, any comments you add

to the `ams.env` file in the run directory disappear as soon as you apply any changes to the options.

## AMS Designer Supports Design Management

Design management controls access to design files to ensure that information remains synchronized as the design changes. AMS Designer supports design management for all components of a design except for

- The `cds_globals` module
- The contents of temporary (TMP) libraries

As described in the *Cadence Library Manager User Guide*, design management reacts to the settings of certain environment variables, such as `CDS_AUTO_CKIN`. You might need to change the value of these variables to obtain the design management behavior you want.

## Specifying the Text Editor to Use

You can specify the text editor for AMS Designer by defining the UNIX shell `EDITOR` or `CDS_TEXT_EDITOR` variables. The value of the `CDS_TEXT_EDITOR` variable takes precedence over the value of the `EDITOR` variable. For example, the command

```
setenv CDS_TEXT_EDITOR emacs
```

sets up the UNIX environment so that AMS Designer uses the `emacs` text editor.

## Specifying Fonts for the Cadence Hierarchy Editor

By default, the Cadence hierarchy editor uses a variable-width font for messages. However, some messages, such as error messages that include pointers, are best viewed with fixed-width fonts. The easiest way to view such messages in a fixed-width font is to use the NCBrowse window. (For more information, see [“Viewing Messages”](#) on page 289.) If you prefer to see messages in a fixed-width font in the hierarchy editor, you can change the hierarchy editor font.

For example, you notice a case like the one illustrated below where the error pointer bar is misaligned. The pointer should point to the last character in the `initial` statement, but does not. The problem arises because the default font used in the hierarchy editor is a variable-

## Virtuoso AMS Environment User Guide

### Setting Up the AMS Environment

---

width font but the alignment of the pointer is calculated as though the font were a fixed-width font.

Cell Bindings		
Library	Cell	View Found
amslib	iter_master	functional
amslib	iter_master	verilog
amslib	iter_top	schematic
Messages		
Compiling verilog.v in cellview amslib iter_master veri ncvlog: v3.150.(a1): (c) Copyright 1995 – 2000 Cadence initial   ncvlog: *E,EXPSMC (/usr1/cds11752/alpha6/test20/SAR_A2D/AMS_lib/am 6 6): expecting a semicolon (;) [3.10(IEEE)]. Verilog-AMS compile failed for cellview amslib iter_ma Compilation of cellview amslib iter_master verilog was		

To align the pointer properly, you can change the variable-width font to a fixed-width font by substituting a statement like

```
CDS.textFont=courier plain 10
```

for the original statement in the `cdsresource.properties` file. (For the steps involved, see [“Changing the Hierarchy Editor Fonts”](#) in Chapter 1 of the *Cadence Hierarchy Editor User Guide*.)

After this change, the information displays like this.

Cell Bindings		
Library	Cell	View Found
amslib	iter_master	functional
amslib	iter_master	verilog
amslib	iter_top	schematic

Messages		
<pre>Compiling verilog.v in cellview amslib iter_master ver ncvlog: v3.150.(a1): (c) Copyright 1995 - 2000 Cadence initial   ncvlog: *E,EXPSMC (/usr1/cds11752/alpha6/test20/SAR_A2D/AMS_lib/amsLib/i expecting a semicolon (;) [3.10(IEEE)]. Verilog-AMS compile failed for cellview amslib iter_ma Compilation of cellview amslib iter_master verilog was Netlisting amslib iter_top schematic.</pre>		

The error pointer now points to the last character in the `initial` statement, as it should.

## Preparing to Use AMS Designer from the Hierarchy Editor

As noted in several places in this document, you can use AMS Designer in various ways according to your needs. One method, especially useful when you are using a design configuration, is to control AMS Designer from the menu of the hierarchy editor. To prepare AMS Designer for this approach,

1. Start the Cadence hierarchy editor.

There are several ways to do this. For example, you can use either of the following methods.

- ☐ From the CIW, choose *File – Open* and then select a config view.
- ☐ Type `cdsHierEditor` at the UNIX command line.

## Virtuoso AMS Environment User Guide

### Setting Up the AMS Environment

The Cadence hierarchy editor window appears.



2. If necessary, choose *File – Open* to select a design configuration. You must have an open design configuration before proceeding to the next step.
3. Choose *Plug-Ins – AMS* to display the *AMS* menu in the menu bar.



All the choices in the *AMS* menu bar except *Run Directory* are disabled until, in the next step, you select a run directory.

Run directories allow you to tailor the AMS 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:

- ☐ AMS Designer behaviors having to do with connect rules, analog simulation control files, and SimVision.

## Virtuoso AMS Environment User Guide

### Setting Up the AMS Environment

---

- ☐ The analog simulation control file used for simulation.
- ☐ The destination directory for error logs.
- ☐ The directory that holds waveforms.
- ☐ The `cds_globals` module associated with the design.
- ☐ The simulation snapshot that is used.

#### 4. Choose *AMS – Run Directory* in the hierarchy editor.

The AMS Run Directory form appears.

The screenshot shows the 'AMS Run Directory' dialog box. It has a title bar with the text 'AMS Run Directory'. Inside, there are several sections:

- Run directory:** A text field containing the path `/usr1/cds11752/alpha6/vhdltestdir/SAR_A2D/ams_run`. Below it is a 'Browse' button.
- Existing run directories:** A dropdown menu showing `/usr1/cds11752/alpha6/vhdltestdir/SAR_A2D/tutorial_run`.
- ☐ **Always use this run directory for this configuration**
- Copy Run Directory Setup** section:
  - From this run directory:** A text field (empty) with a 'Browse' button below it.
  - Existing run directories:** A dropdown menu showing `/usr1/cds11752/alpha6/vhdltestdir/SAR_A2D/tutorial_run`.
  - Files to copy:** Two large empty rectangular boxes, one labeled 'Copy:' and one labeled 'Do not copy:'. Between them are two small arrow buttons (right and left).

At the bottom of the dialog are four buttons: 'OK', 'Cancel', 'Apply', and 'Help'.

#### 5. Fill in the *Run directory* field.



## Virtuoso AMS Environment User Guide

### Setting Up the AMS Environment

---

- ☐ If you want to use an existing run directory, you can choose the directory from the upper *Existing run directories* cyclic field or click the upper *Browse* button to search the hierarchy for the directory you want to use.
- ☐ If you want to use a new run directory, type the path and name in the *Run directory* field.

**6.** (Optional) Turn on *Always use this run directory for this configuration*.

Turning on this field tells AMS Designer to use the specified run directory every time you open this configuration in the hierarchy editor and install the *AMS* menu. In this case, AMS Designer knows which run directory to use and so all the entries in the *AMS* menu are enabled immediately.

**7.** (Optional) Copy setup information from an existing run directory to a new run directory.

The fields in the *Copy Run Directory Setup* pane provide a way for you to copy setup information such as `ams.env` and `amsSC.scs` files to a newly created run directory.

- a.** Select the run directory to serve as the source of setup files. You can do any one of the following:
  - ☐ Type the path and file name into the *From this run directory* field.
  - ☐ Click the lower *Browse* button and select a path and file name.
  - ☐ Choose an existing run directory from the lower *Existing run directories* cyclic field.
- b.** Use the arrow buttons between the *Copy* and *Do not copy* lists to specify the files to be copied

**8.** Click *OK*.

## Virtuoso AMS Environment User Guide

### Setting Up the AMS Environment

---

AMS Designer enables all the choices in the *AMS* menu.

#### AMS Menu Choices

---

Menu Item	For more information, see
<i>AMS</i>	
<i>Run Directory</i>	<a href="#"><u>"Preparing to Use AMS Designer from the Hierarchy Editor"</u></a> on page 62
<i>Options</i>	<a href="#"><u>"Setting Options for Global Design Data"</u></a> on page 211
<i>Design Prep</i>	<a href="#"><u>"Running AMS Design Prep"</u></a> on page 221
<i>Global Signals</i>	<a href="#"><u>"Specifying Global Signals"</u></a> on page 213
<i>Design Variables</i>	<a href="#"><u>"Specifying Design Variables"</u></a> on page 216
<i>Analog Models</i>	<a href="#"><u>"Specifying Model Files to Use During Elaboration"</u></a> on page 219
<i>Save/Plot</i>	<a href="#"><u>"Creating Probes"</u></a> on page 277
<i>Run Simulation</i>	<a href="#"><u>"Elaborating and Simulating"</u></a> on page 286
<i>Waveform Viewer</i>	<a href="#"><u>"Starting the SimVision Waveform Viewer"</u></a> on page 291
<i>Direct Plot</i>	<a href="#"><u>"Plotting Waveforms After Simulation Ends"</u></a> on page 290
<i>Logfile Viewer</i>	
<i>Netlister Log File</i>	<a href="#"><u>"Viewing the AMS Netlister Log"</u></a> on page 126
<i>Simulator Log Files</i>	<a href="#"><u>"Viewing Messages"</u></a> on page 289

---

---

# Netlisting

---

The AMS netlister translates CDBA 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 CDBA cellview. The netlister places each output file in the corresponding cellview directory.

As explained in this chapter, you can run the AMS netlister explicitly when necessary. However, one of the strengths of the Virtuoso® AMS Designer flow is that you can set up the AMS netlister to run automatically whenever you check and save a schematic. When you run in this way, your design is always ready for simulation.

This chapter contains the following sections:

- [Netlisting Modes Supported by the AMS Netlister](#) on page 68
- [Preparing Existing Analog Primitive Libraries for Netlisting](#) on page 74
- [Specifying the Behavior of the Netlister and Compilers](#) on page 74
- [Viewing the AMS Netlister Log](#) on page 126
- [Understanding the Output from the AMS Netlister](#) on page 126
- [How Inherited Connections Are Netlisted](#) on page 127
- [How Aliased Signals Are Netlisted](#) on page 131
- [How m-factors \(Multiplicity Factors\) Are Netlisted](#) on page 132
- [How Iterated Instances Are Netlisted](#) on page 133
- [Passing Model Names as Parameters](#) on page 134
- [Specifying Parameters to be Excluded from Netlisting](#) on page 137
- [Preparing to Netlist User-Defined Functions](#) on page 140
- [Ensuring that Floating Point Parameters Netlist Correctly](#) on page 143

## Netlisting Modes Supported by the AMS Netlister

The AMS netlister provides several ways to netlist cellviews:

- Automatic netlisting, where an application-specific operation (such as *Check and Save* in the Virtuoso Schematic Editor) triggers netlisting of the cellview being saved
- Netlist updating and netlisting of entire designs using the Cadence hierarchy editor
- Netlisting from the UNIX command line
- Library netlisting, using a menu command in the CIW to netlist an entire library, all the views of a cell, or a single cellview
- Netlisting of cells in response to changes in CDF

The following sections describe each of these methods in more detail and explain how to use them.

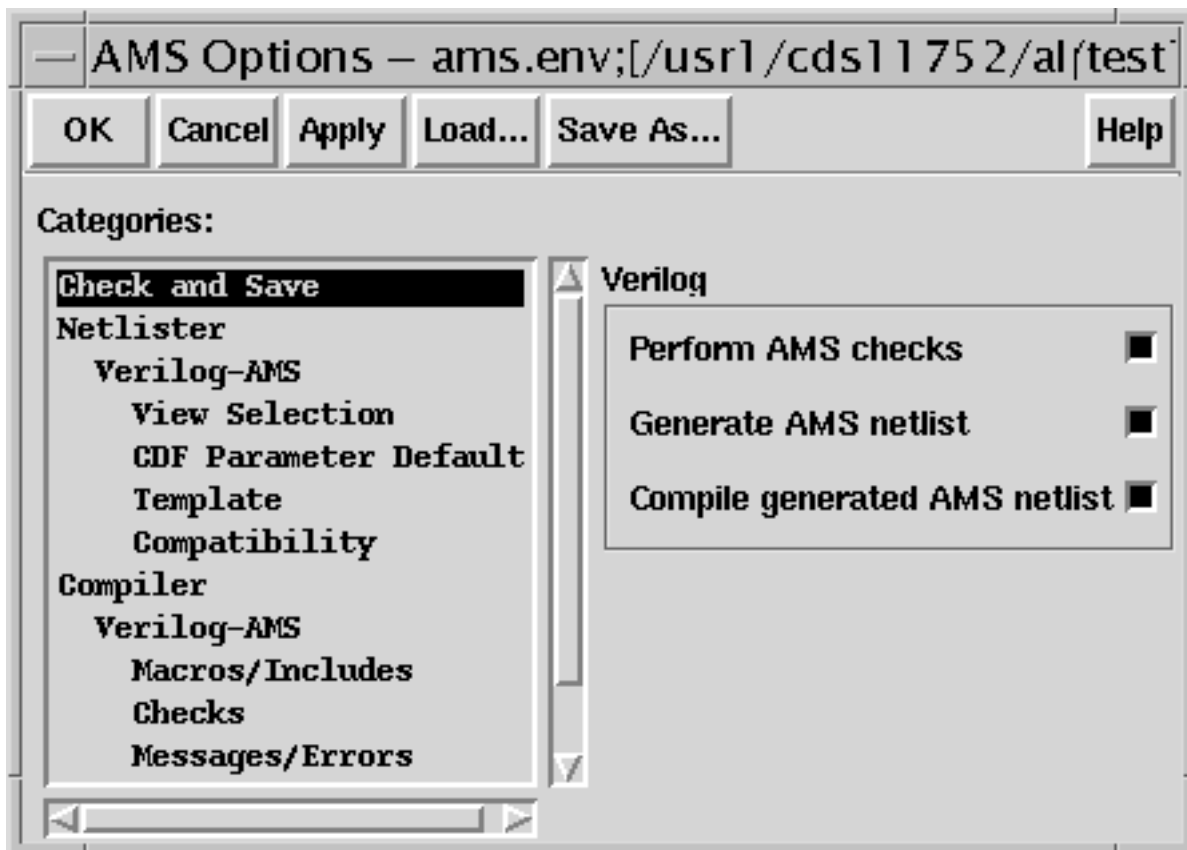
### Automatic Netlisting of a Cellview

This method is perhaps the most likely to be used in day-to-day production work, because netlisting is transparent, occurring automatically whenever you save a CDBA view that has valid connectivity. When you use this method and specify that both netlisting and compiling take place automatically, you always have the information required so the next tool in the flow, AMS Design Prep or the AMS simulator, can run quickly.

To use this method,

1. Choose *Tools – AMS – Options* in the CIW to display the AMS Options window.

2. Select *Check and Save* in the *Categories* field.



3. Select what you want the AMS netlister to do when you check and save a CDBA cellview. You can choose to have the netlister check, netlist, and compile the cellview.

**Note:** If *Generate AMS netlist* is disabled, the AMS netlister removes any netlist previously created by the netlister for the cellview being checked and saved. This behavior continues even when the AMS tools are disabled. This process of removing existing netlists ensures that you do not inadvertently simulate an out-of-date netlist.

4. If necessary, select other choices in the *Categories* field and set the options that control the AMS netlister. For more information, see [“Specifying the Behavior of the Netlister and Compilers”](#) on page 74.
5. Click *OK* or *Apply* to save your settings.

At this point you are ready to work with the CDBA data. For example, you use Virtuoso Schematic Editor to create or edit schematic views, then use the *Check and Save* command. In response, the AMS netlister runs automatically, creating a Verilog-AMS netlist in the cellview directory of your saved schematic view. The newly created netlist is available

to all users of the block: None of the users need to recreate the netlist unless the block changes.

## Netlist Updating and Netlisting of Entire Designs

In this method, the AMS netlister runs under the control of AMS Design Prep. AMS Design Prep is a Cadence hierarchy editor plug-in that operates on the design configuration. This is the easiest method to use when you are primarily interested in ensuring that all the cellviews used in a design have up-to-date netlists.

For guidance on using AMS Design Prep, see [Chapter 10, “Preparing a Design for Simulation.”](#)

## Netlisting from the UNIX Command Line

This method allows you to netlist an entire library, all the views of a cell, or a single cellview without starting the graphical user interface. Instead you use the `amsdirect` command, which has the following syntax.

```
amsdirect_command ::=  
    amsdirect -Lib libName [ -Cell cellName ] [ -View viewName ]  
                [ -VERilog ] [ -Env envFileName ] [ -Log logFileName ]  
                [ -Incremental ] [ -Help ] [ -VERSION ]
```

The following table describes the `amsdirect` command options.

amsdirect Option	Effect
<code>-Lib libName</code>	Specifies the library containing the cellviews that you want to translate from CDBA to Verilog-AMS netlists. If you do not specify a cell with the <code>-Cell</code> option, the AMS netlister translates eligible views for every cell in the library.
<code>-Cell cellName</code>	Specifies the cell containing the cellviews that you want to translate from CDBA to Verilog-AMS netlists. If you do not specify a view with the <code>-View</code> option, the AMS netlister translates each view type listed on the <code>amsEligibleViewTypes</code> variable in the <code>ams.env</code> file.

## Virtuoso AMS Environment User Guide

### Netlisting

amsdirect Option	Effect
<b>-View</b> <i>viewName</i>	Specifies the cellview name that you want to translate from CDBA to a Verilog-AMS netlist. The type of the cellview must be one of: <code>schematic</code> , <code>symbolic</code> , <code>maskLayout</code> , or <code>netlist</code> but the name of the cellview can be any legal name. If you do not specify a view name, the AMS netlister netlists each view type listed on the <code>amsEligibleViewTypes</code> variable in the <code>ams.env</code> file.
<b>-VERIlog</b>	Tells the AMS netlister to produce Verilog-AMS netlists for the processed CDBA cellviews. This option takes precedence over the <code>checkOnly</code> and <code>checkAndNetlist</code> variables, whose settings in the <code>ams.env</code> file otherwise determine the behavior. If both variables are set to <code>nil</code> and you do not use the <code>-VERIlog</code> option, the AMS netlister does nothing.
<b>-Env</b> <i>envFileName</i>	<p>Tells the AMS netlister to read the <code>ams.env</code> variables in the <i>envFileName</i> file and to use them to overlay the base set of <code>ams.env</code> variables. For information about how the base set of variables is determined, see <a href="#">“How AMS Designer Determines the Set of Variables”</a> on page 360.</p> <p><b>Note:</b> If <i>envFileName</i> is read while the base set is being determined, the <code>-Env</code> option has no effect.</p>
<b>-Log</b> <i>logFileName</i>	<p>Tells the AMS netlister to write messages to <i>logFileName</i>. The <code>-Log</code> option takes precedence over the <code>logFileName</code> variable used in <code>ams.env</code> files.</p> <ul style="list-style-type: none"> <li>■ If <i>logFileName</i> is an absolute path, the log file is written to <i>logFileName</i>.</li> <li>■ If <i>logFileName</i> is a relative path and <ul style="list-style-type: none"> <li>□ <code>CDS_LOG_PATH</code> is null, <i>logFileName</i> is placed in the current directory.</li> <li>□ <code>CDS_LOG_PATH</code> is non-null, the value of <code>CDS_LOG_PATH</code> is prepended to <i>logFileName</i>.</li> </ul> </li> </ul> <p>For more information about specifying the log file, see <a href="#">“logFileName”</a> on page 421.</p>
<b>-Incremental</b>	Tells the AMS netlister to netlist only new or revised cellviews.
<b>-Help</b>	Returns a brief description of the <code>amsdirect</code> command and its options.

## Virtuoso AMS Environment User Guide

### Netlisting

---

---

amsdirect Option	Effect
------------------	--------

---

<b>-VERSION</b>	Returns the version number of the AMS netlister.
-----------------	--------------------------------------------------

---

For example, the following command tells the AMS netlister, under the control of the options in the `myams.env` file, to generate Verilog-AMS netlists for all of the eligible views in the `mycell` cell.

```
amsdirect -li mylib -cell mycell -veri -env myams.env
```

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
```

## Library Netlisting

This method allows you to use a form that opens from the CIW to check, netlist, and compile an entire library, all the views of a cell, or a single cellview. You can also use this form to netlist and compile only new or revised cellviews. To use this method,



## Virtuoso AMS Environment User Guide

### Netlisting

---

1. Choose *Tools – AMS – Netlist* in the CIW to open the AMS Netlister form.

The screenshot shows the 'AMS Netlister' dialog box. At the top, there is a title bar with the text 'AMS Netlister'. Below the title bar are four buttons: 'OK', 'Cancel', 'Apply', and 'Help'. The main area of the dialog is divided into two sections. The first section is titled 'Select cellviews' and contains three text input fields labeled 'Library:', 'Cell:', and 'View:'. To the right of the 'View:' field is a 'Browse...' button. The second section is titled 'Actions' and contains three main options, each with a checkbox and a list of sub-options. The first option is 'Perform AMS checks' with a checkbox. The second option is 'Netlist' with a checkbox, and it has two sub-options: 'Incremental' (selected with a diamond) and 'All' (deselected with a diamond). The third option is 'Compile' with a checkbox, and it has two sub-options: 'When netlisting' (selected with a diamond) and 'All cellviews' (deselected with a diamond).

2. Specify the library containing the cellviews that you want to operate on.
3. (Optional) Specify a cell.

If you do not specify a cell, the AMS netlister operates on eligible views for every cell in the library.

4. (Optional) Specify a view.

If you do not specify a view, the AMS netlister operates on each view type specified by the `amsEligibleViewTypes` variable in the `ams.env` file. (The views operated on can be further tuned by specifying view names to process or exclude. For more information, see [“Setting View Selection Options”](#) on page 90.)

5. Select what you want the AMS netlister to do with the specified cellviews in the current run. Choose incremental netlisting if you want to netlist only new or changed cellviews. Note that changes you make here are for this run only and do not affect the settings in the *Options* menu.
6. Click *OK* to begin the run.

## Netlisting of Cells in Response to Changes in CDF

In this method, the AMS netlister runs when you use the CDF editor to update the CDF information for a cell (provided that the `netlistAfterCdfChange` variable in the `ams.env` file is set to `t`). The `amsEligibleViewTypes` variable in the `ams.env` file specifies which cellviews of the cell are eligible for netlisting.

For more information, see [“netlistAfterCdfChange”](#) on page 454 and [“amsEligibleViewTypes”](#) on page 380.

## Preparing Existing Analog Primitive Libraries for Netlisting

Cadence netlisters format instances of analog devices according to the instructions specified in the Simulation Information (`simInfo`) section of the device's CDF. The `simInfo` 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 simulator, the `simInfo` for analog primitives now contains a new section called *ams*.

Cadence provides a conversion tool that helps you generate the new *ams* `simInfo` from the corresponding *spectre* information. For more information, see [Appendix B, “Updating Legacy SimInfo for Analog Primitives.”](#)

## Specifying the Behavior of the Netlister and Compilers

By default, the behavior of the AMS netlister and compilers is controlled by variables stored in files named `ams.env`. These variables tell the netlister and compilers what they should do in general (for example, check the design or check and netlist the design) and what to do in certain situations (if the netlister encounters sparse buses or conflicting bus ranges).

See [Appendix A, “Variables for `ams.env` Files,”](#) for detailed descriptions of the variables.

## Opening the AMS Options Windows

There are two main graphical interfaces that you can use to set the `ams.env` values:

- Through *AMS – Options* in the Cadence hierarchy editor

For guidance on using this interface, see [“Opening the Hierarchy Editor AMS Options Window”](#) on page 75.

- Through *Tools – AMS – Options* in the CIW

For guidance on using this interface, see [“Opening the CIW AMS Options Window”](#) on page 86.

You can set the values of most variables using either of these interfaces. They both operate on the same `ams.env` file so changes that you make from one interface are reflected in the other. However, variables that control what happens when you check and save from a schematic, and variables that specify the views that are eligible for netlisting are available only in the CIW interface.

## Opening the Hierarchy Editor AMS Options Window

To review or change the current options from the Cadence hierarchy editor,

1. Start the hierarchy editor, install the AMS menu item, and select a run directory.

For guidance, see [“Preparing to Use AMS Designer from the Hierarchy Editor”](#) on page 62.

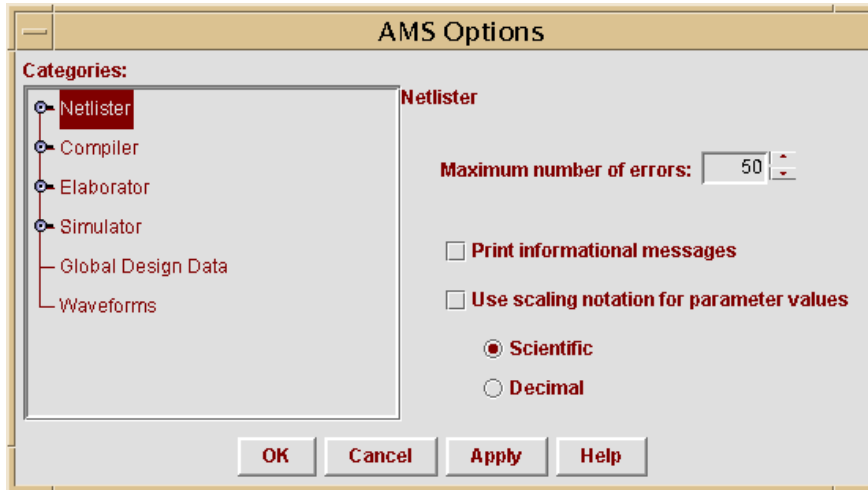
2. Choose *AMS – Options* and then one of the subchoices.

## Virtuoso AMS Environment User Guide

### Netlisting

---

The AMS Options window appears.



The *Categories* pane of the window shows the options categories: *Netlister*, *Compiler*, *Elaborator*, *Simulator*, *Global Design Data*, and *Waveforms*.

#### For guidance on setting these options

#### See

*Netlister*

[“Setting Netlister Options from the Hierarchy Editor”](#) on page 77

*Compiler*

[“Setting Compiler Options”](#) on page 98

*Elaborator*

[“Setting Elaborator Options”](#) on page 232

*Simulator*

[“Setting Simulator Options”](#) on page 247

*Global Design Data*

[“Setting Options for Global Design Data”](#) on page 211

Because the *Global Design Data* options affect AMS Design Prep, rather than the AMS netlister, there is no corresponding graphical interface available from the CIW.

*Waveforms*

[“Setting Waveform Selection Options”](#) on page 274

Note that if you click *OK* or *Apply* after selecting settings in the AMS Options window, the revised options are immediately saved to the AMS environment file, `ams.env`, and become the default values. If you edit the `ams.env` file outside of the Cadence hierarchy editor, the edited settings appear in the AMS Options window the next time AMS reads the `ams.env` file.

## Setting Netlister Options from the Hierarchy Editor

The netlister options available from the hierarchy editor interface are organized in the following categories:

Category	Purpose	For information, see
<i>Netlister</i>	Controls general netlister options	<a href="#"><u>“Setting General Netlister Options” on page 77</u></a>
<i>Verilog-AMS</i>	Controls how language extensions are handled	<a href="#"><u>“Setting Verilog-AMS Options for the Netlister” on page 79</u></a>
<i>CDF Parameter Defaults</i>	Specifies default values for CDF parameters	<a href="#"><u>“Specifying Default Values for CDF Parameters” on page 80</u></a>
<i>Template</i>	Controls what information and files are included in netlist templates	<a href="#"><u>“Setting Template Header and File Include Options” on page 81</u></a>
<i>Compatibility</i>	Controls options that allow you to retain data compatibility with older netlisters, such as the Cadence analog design environment or Verilog Integ	<a href="#"><u>“Setting Compatibility Options” on page 84</u></a>

### Setting General Netlister Options

With the options in this category, you can set the maximum number of error messages displayed in the log file when you netlist, and you can specify the kind of scaling to be used for parameter values.

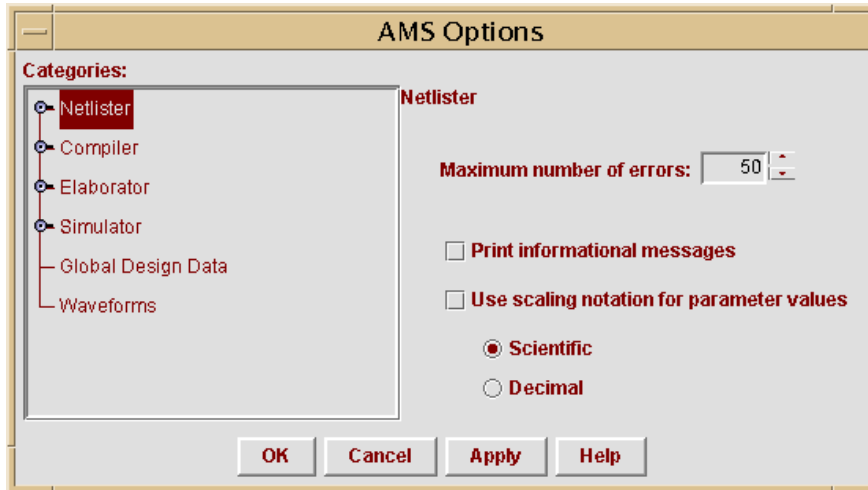
To display the *Netlister* pane,

## Virtuoso AMS Environment User Guide

### Netlisting

---

- Select the *Netlister* category.



#### Setting the Maximum Number of Displayed Error Messages

To set the maximum number of displayed error messages,

- Click and type in the *Maximum number of errors* field or use the up and down arrow keys to set the maximum number of errors the AMS netlister can encounter before it stops processing the design.

For information on the corresponding `ams.env` variable, see [“amsMaxErrors”](#) on page 385.

**Note:** If any errors are found, no netlist is generated and any existing netlist is removed. This behavior ensures that you do not inadvertently simulate an out-of-date netlist.

#### Requesting More Numerous and More Extensive Messages

To increase the amount of information produced as messages, as you might want to do when you are debugging a problem,

- Select the *Print informational messages* option.

For information on the corresponding `ams.env` variable, see [“amsVerbose”](#) on page 387.

#### Setting the Scaling Notation for Parameter Values

You can control the netlisting of scaling factors for parameter values by expanding the scaling factors as desired. To expand scaling factor suffixes,

## Virtuoso AMS Environment User Guide

### Netlisting

---

1. Select the *Use scaling notation for parameter values* option.
2. Set the scaling notation that you want to use for parameter values,
  - ☐ Select *Scientific* to expand scaling factor suffixes in scientific notation.  
For example, this choice expands 5.46M as 5.46e6.
  - ☐ Select *Decimal* to expand scaling factor suffixes in decimal notation.  
For example, this choice expands 5.46M as 5,460,000.

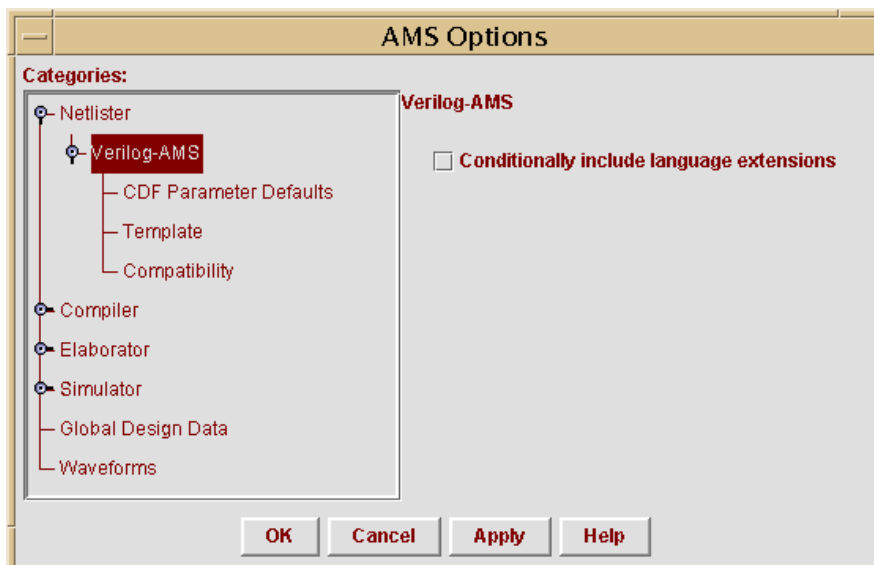
For information on the corresponding `ams.env` variable and a list of scaling factor suffixes supported by various target simulators, see [“amsExpScalingFactor”](#) on page 382.

### Setting Verilog-AMS Options for the Netlister

This pane allows you to set an option that can make AMS Designer netlists compatible with simulators that cannot handle AMS language extensions. The AMS Designer netlister enables this capability by enclosing language extensions in conditional statements. With these conditional statements in place, you can control when language extensions are active.

To set options for language extensions,

1. Select the *Netlister – Verilog-AMS* category to display the *Verilog-AMS* pane.



2. Select the *Conditionally include language extensions* option if you want extended language constructs to be conditionally included.

## Virtuoso AMS Environment User Guide

### Netlisting

If you select this option, the netlister sets off language extensions with ``ifdef INCA` directives, which facilitates using the netlist in simulators that do not support the extensions.

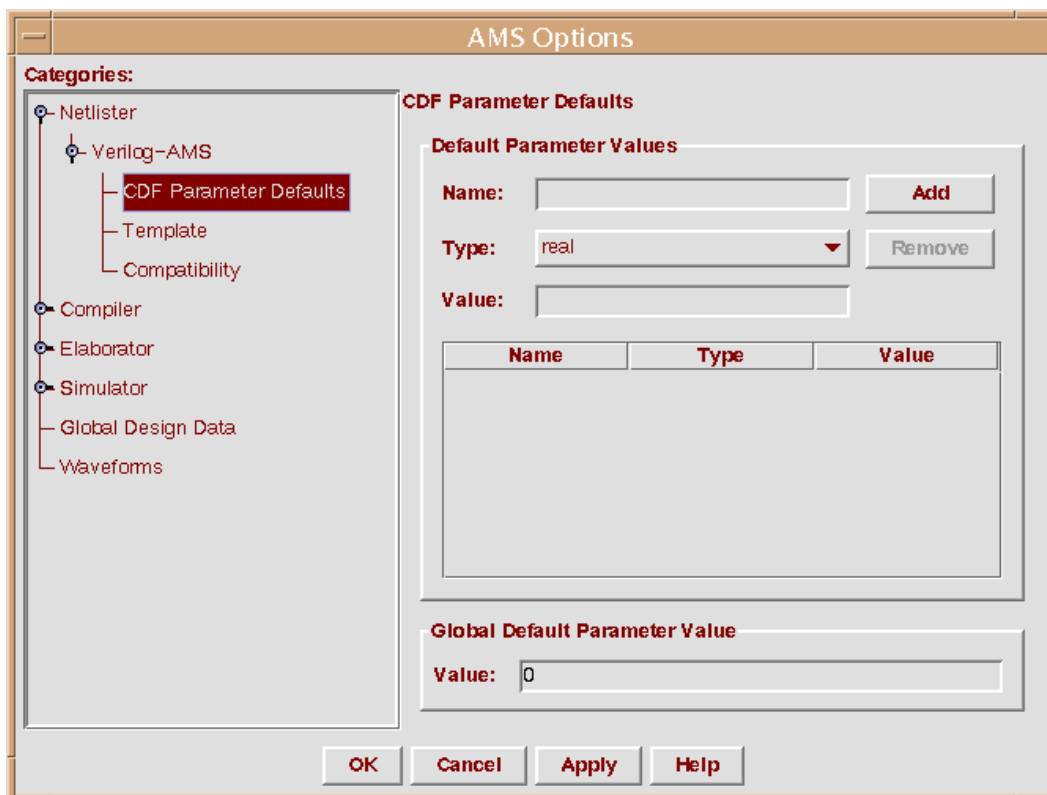
For information on the corresponding `ams.env` variable, see [“ifdefLanguageExtensions”](#) on page 410.

### Specifying Default Values for CDF Parameters

With the options in this category, you can specify default values for CDF parameters.

To display the *CDF Parameter Defaults* pane,

- Select the *Netlister – Verilog-AMS – CDF Parameter Defaults* category to display the *CDF Parameter Defaults* pane.



### Specifying Default Values for Specific CDF Parameters

To specify a default value for a specific CDF parameter,

1. Type the name of the parameter in the *Name* field.



2. In the *Type* field, select *integer* or *real* to indicate the type of the parameter.
3. Type a default value into the *Value* field.
4. Click *Add*.

The newly specified default value appears in the list below.

The AMS netlister uses this default value when the specified parameter does not have a default value in the CDBA data.

For information on the corresponding `ams.env` variable, see “[paramDefVals](#)” on page 464.

To remove the default value for a parameter,

- Select the parameter in the list and then click *Remove*.

#### **Specifying a Global Default Value for CDF Parameters**

To specify a global default parameter value,

- Type a default value into the *Value* field of the *Global Default Parameter Value* section.

The AMS netlister uses this default value for any parameter in the CDBA data that does not otherwise have a default value.

For information on the corresponding `ams.env` variable, see “[paramGlobalDefVal](#)” on page 465.

#### **Setting Template Header and File Include Options**

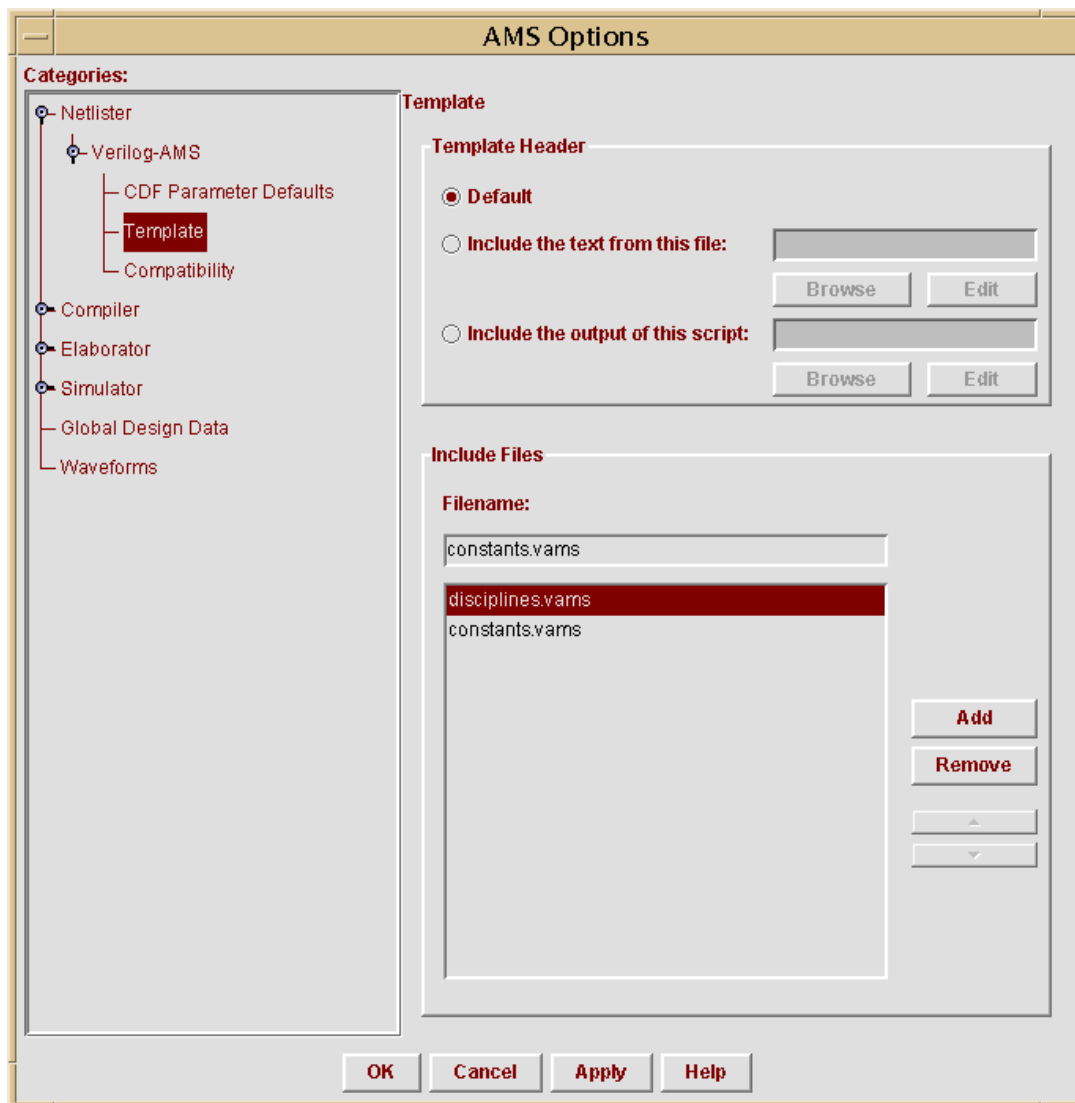
With the options in this category, you can specify headers for AMS netlists and you can specify the names of files to be included by ``include` directives.

To display the *Template* pane,

## Virtuoso AMS Environment User Guide

### Netlisting

- Select the *Netlister – Verilog-AMS – Template* category to display the *Template* pane.



#### Specifying the Header to be Used in AMS Netlists

To specify the header to be used in AMS netlists,

1. In the Template Header area, select the method you want to use to specify template headers.

For information on the corresponding `ams.env` variable, see [“headerText”](#) on page 408

- ❑ *Default*

## Virtuoso AMS Environment User Guide

### Netlisting

---

Inserts the default header at the top of each generated netlist.

For example, choosing *Default* inserts the following text into netlists.

```
// Verilog-AMS netlist generated by the AMS netlister, version 4.4.6.100.43.  
// Cadence Design Systems, Inc.
```

❑ *Include the text from this file*

Inserts the default header and the contents of the specified file at the top of each generated netlist.

For example, choosing *Include the text from this file* and specifying a file containing the following text

```
// Module produced by  
// ASIC Team: Ocelot  
// San Jose Development Center
```

inserts the following at the top of each 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
```

❑ *Include the output of this script*

Inserts the default header and the results of running the specified script at the top of each generated netlist.

For example, Choosing *Include the output of this script* and specifying a file containing the following script

```
echo '// Module produced by:'  
echo '// ASIC Interactive, Ltd.'  
printf '// (c) '  
date '+DATE: %m/%d/%y%n'
```

inserts the following at the top of each 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
```

2. If you select either of the latter two choices in the previous step, specify the path and filename of the text file or script file.

Relative paths are resolved with respect to the directory where you started the AMS software.

## Virtuoso AMS Environment User Guide

### Netlisting

---

For information on the corresponding `ams.env` variables, see “[headerText](#)” on page 408, “[templateFile](#)” on page 551, and “[templateScript](#)” on page 552.

#### Specifying Files to be Included in AMS Netlists

To specify files to include in the netlist by `\include` directives,

1. Type the name of the file to include in the *Filename* field.

The name that you enter appears verbatim in the netlist.

For information on the corresponding `ams.env` variable, see “[includeFiles](#)” on page 412.

2. Click *Add* to add the filename to the list of filenames displayed below the *Filename* field.

The newly added file is included, with a `\include` directive, in every netlist. To control which file of that name is included, you can use the *Include Directory* table accessed by choosing *Compiler – Verilog-AMS – Macros/Includes* in the AMS Options window. For guidance, see “[Specifying Macros and Specifying Directories to be Searched](#)” on page 104.

3. To remove a file from the list, select the filename, and then click *Remove*.
4. If necessary, move a filename up or down in the list by clicking the up and down arrows.

Files are included in the netlist in the order that they appear in the list. The order is important if you have files that use declarations in another file. For example, if `File2` uses a declaration from `File1`, you must ensure that `File1` is above `File2` in the list.

#### Setting Compatibility Options

If you have the following design exceptions in your cellview, the AMS netlister, by default, either issues a warning or does not create a netlist.

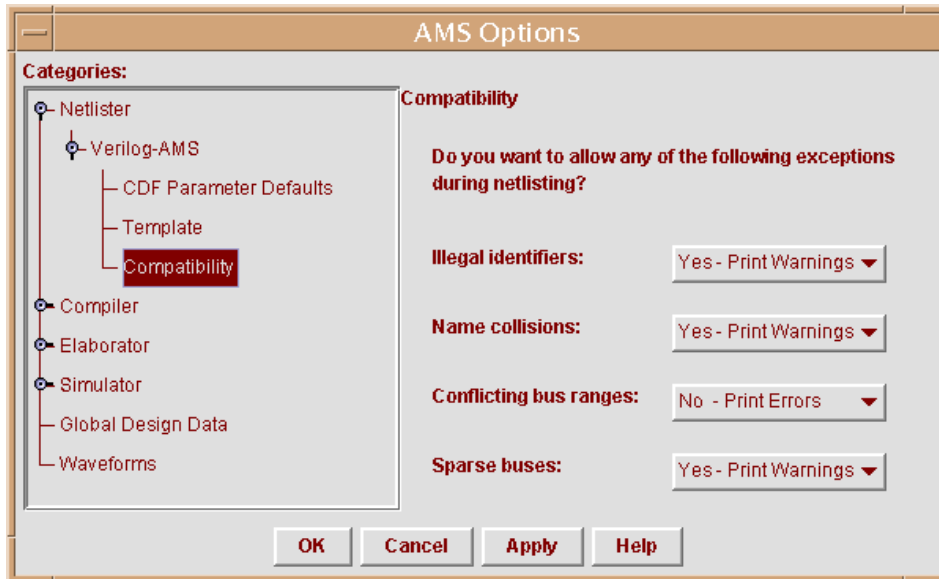
Exception	For an explanation of the exception and a description of the corresponding <code>ams.env</code> variable, see
Illegal identifiers	“ <a href="#">allowIllegalIdentifiers</a> ” on page 370
Name collisions	“ <a href="#">allowNameCollisions</a> ” on page 372
Conflicting bus ranges	“ <a href="#">allowDeviantBuses</a> ” on page 368
Sparse buses	“ <a href="#">allowSparseBuses</a> ” on page 374

To override this default behavior,

## Virtuoso AMS Environment User Guide

### Netlisting

1. Select the *Netlister – Verilog-AMS – Compatibility* category to display the *Compatibility* pane.



2. Select the settings you want for each of the exceptions.

Each exception has the following possible settings:

- ☐ *No – Print Errors*: The AMS netlister issues an error if the netlister encounters the design exception. No netlist is produced.
- ☐ *Yes – Print Warnings*: The AMS netlister issues a warning if the netlister encounters the design exception. Netlisting continues after the netlister makes necessary changes.
- ☐ *Yes – Silently*: The AMS netlister continues netlisting if it encounters the design exception, after making changes that allow netlisting to continue. The AMS netlister does not issue a warning.

If you select *Yes – Print Warnings* or *Yes – Silently*, the AMS netlister makes the following changes when it encounters a design exception.

- ☐ *Illegal identifiers*: Maps non-compliant 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 non-compliant 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 generated netlist is likely to be less readable than one created from compliant

bus data. The associated warning, if issued, tells you how the non-compliant bus data is transformed.

- ❑ Sparse buses: Overdeclares any sparse buses.

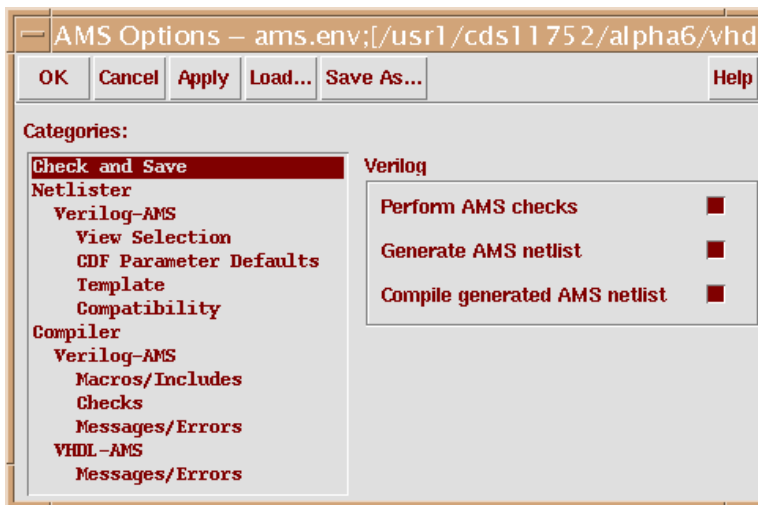
For information on the corresponding `ams.env` variables, see [“allowIllegalIdentifiers”](#) on page 370, [“allowNameCollisions”](#) on page 372, [“allowDeviantBuses”](#) on page 368, and [“allowSparseBuses”](#) on page 374.

## Opening the CIW AMS Options Window

To review or change the current options from the CIW,

- From the CIW, choose *Tools – AMS – Options*.

The AMS Options window appears.



### For guidance on setting these options

### See

*Check and Save*

[“Setting Check and Save Options”](#) on page 86

*Netlister*

[“Setting Netlister Options from the CIW”](#) on page 87

*Compiler*

[“Setting Compiler Options from the CIW”](#) on page 114

## Setting Check and Save Options

With the options in this category, you can specify the overall behavior of the AMS netlister.

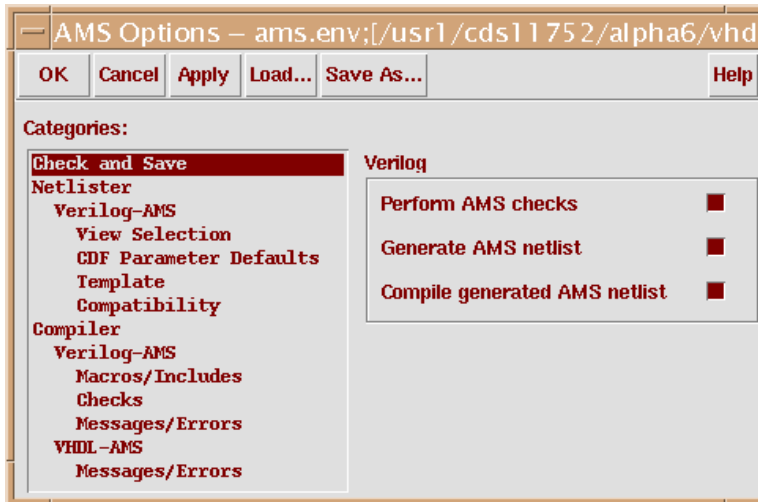
## Virtuoso AMS Environment User Guide

### Netlisting

---

To set the Check and Save options,

1. Select *Check and Save* in the *Categories* field.



2. Select what you want the AMS netlister to do. You can choose to have the netlister check, netlist, and compile the cellview. For information on the corresponding `ams.env` variables, see [“checkOnly”](#) on page 394, [“checkAndNetlist”](#) on page 393, and [“prohibitCompile”](#) on page 468.

For example, assume that the settings are as given in the previous illustration. If you use Virtuoso Schematic Editor to edit a schematic cellview and then use the *Check and Save* command, the AMS netlister automatically checks, netlists, and compiles the cellview.

### Setting Netlister Options from the CIW

The netlister options available from the CIW interface are organized in the following categories:

Category	Purpose	For information, see
<i>Netlister</i>	Controls general netlister options	<a href="#">“Setting General Netlister Options”</a> on page 88

## Virtuoso AMS Environment User Guide

### Netlisting

---

Category	Purpose	For information, see
<i>Verilog-AMS</i>	Controls how language extensions are handled	<a href="#">“Setting Language Extension Options”</a> on page 89
<i>View Selection</i>	Controls how AMS Designer selects views for netlisting	<a href="#">“Setting View Selection Options”</a> on page 90
<i>CDF Parameter Defaults</i>	Specifies default values for CDF parameters	<a href="#">“Setting Default Values for CDF Parameters”</a> on page 92
<i>Template</i>	Controls what information and files are included in netlist templates	<a href="#">“Setting Template Header and File Include Options”</a> on page 93
<i>Compatibility</i>	Controls options that allow you to retain data compatibility with older netlisters, such as the Cadence analog design environment or Verilog Integ	<a href="#">“Setting Compatibility Options”</a> on page 96

#### Setting General Netlister Options

With the option in this category, you can set the maximum number of error messages displayed in the log file when you netlist, and specify the scaling notation to be used for parameter values.

To set the message and scaling notation options,

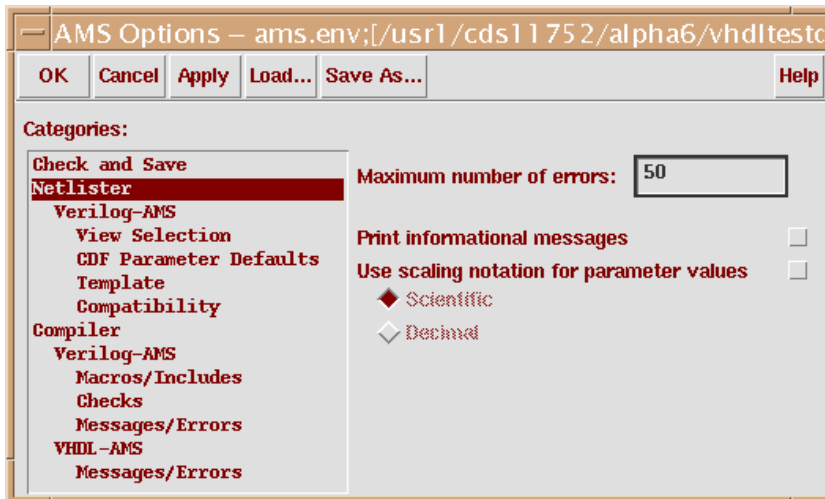
1. Select *Netlister* in the *Categories* field.



## Virtuoso AMS Environment User Guide

### Netlisting

The associated option fields appear.



2. Click and type in the *Maximum number of errors* field to set the maximum number of errors the AMS netlister can encounter before it stops processing the design.

For information on the corresponding `ams.env` variable, see [“amsMaxErrors”](#) on page 385.

**Note:** If any errors are found, no netlist is generated and any existing netlist is removed. This behavior ensures that you do not inadvertently simulate an out-of-date netlist.

3. Turn on *Print information messages* to increase the amount of information produced, as you might want to do when you are debugging a problem.
4. Select a *Use scaling notation for parameter values* button to set the scaling notation that you want to use for parameter values.
  - ☐ Select *Scientific* to expand scaling factor suffixes in scientific notation.
  - ☐ Select *Decimal* to expand scaling factor suffixes in decimal notation.

For information on the corresponding `ams.env` variable, see [“amsExpScalingFactor”](#) on page 382. This cross-reference has a list of scaling factor suffixes supported by various target simulators.

#### Setting Language Extension Options

With the option in this category, you can specify how language extensions are handled.

To set the language extensions option,

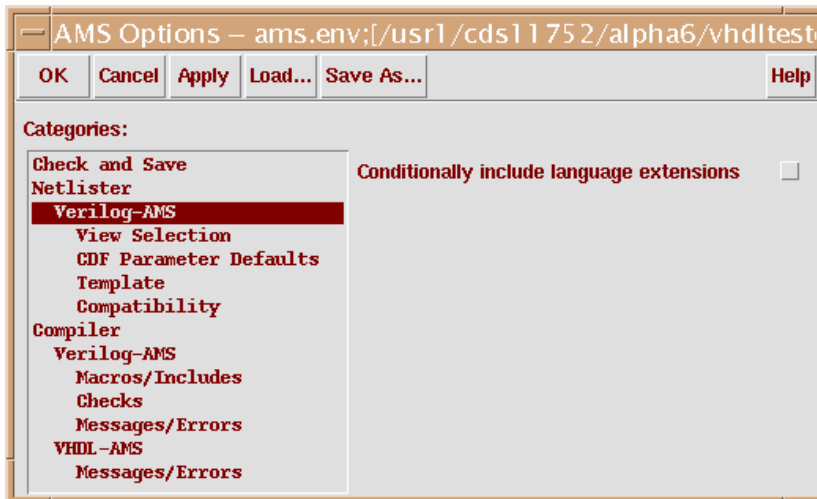
1. Select *Netlister – Verilog-AMS* in the *Categories* field.

## Virtuoso AMS Environment User Guide

### Netlisting

---

The associated option field appears.



2. Select the *Conditionally include language extensions* option if you want extended language constructs to be conditionally included.

If you select this option, the netlister sets off language extensions with ``ifdef INCA` directives, which facilitates using the netlist in simulators that do not support the extensions.

For information on the corresponding `ams.env` variable, see [“ifdefLanguageExtensions”](#) on page 410.

#### Setting View Selection Options

The AMS netlister can translate four types of CDBA cellviews: schematic, symbolic, netlist, and maskLayout.

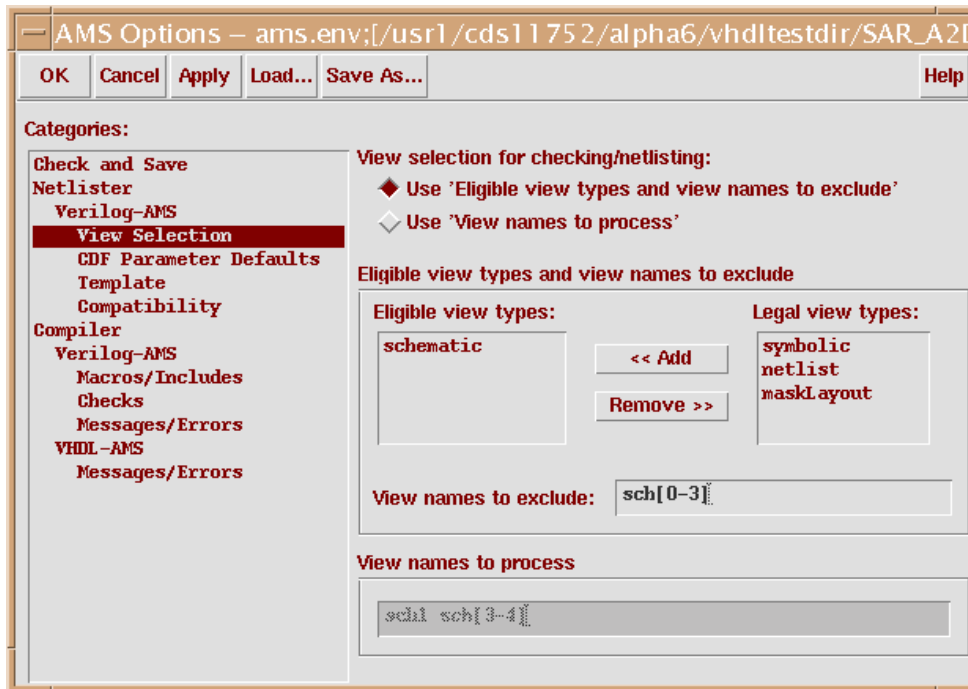
To specify which of the four types are to be netlisted,

1. Select *Netlister – Verilog-AMS – View Selection* in the *Categories* field.

## Virtuoso AMS Environment User Guide

### Netlisting

The associated option fields appear.



2. In the View selection for checking/netlisting area, select the method to be used for choosing views.

- ☐ Selecting *Use 'Eligible view types and view names to exclude'* indicates that the views are specified by the *Eligible view types* field in conjunction with the *View names to exclude* field. If you use this method,
  - a. Select cellview types in the *Legal view types* field and click *Add* to make the cellview types eligible for netlisting. For information on the corresponding `ams.env` variable, see ["amsEligibleViewTypes"](#) on page 380.
  - b. As necessary, select cellview types in the *Eligible view types* field and click *Remove* to make the view types ineligible for netlisting.
  - c. In the *View names to exclude* field, type regular syntax expressions to specify view names that are *not* to be processed. For information on the corresponding `ams.env` variable, see ["excludeViewNames"](#) on page 406.
- ☐ Selecting *Use 'View names to process'* indicates that the views are specified by only the *View names to process* field. If you use this method,

## Virtuoso AMS Environment User Guide

### Netlisting

- a. In the *View names to process* field, type regular syntax expressions to specify view names to be processed. For information on the corresponding `ams.env` variable, see [“useProcessViewNamesOnly”](#) on page 571

For example, assume the options are set to the selections in the previous illustration. Assume also that the AMS netlister runs on a cell that has the six schematic views `sch0`, `sch1`, `sch2`, `sch3`, `sch4`, and `sch5`. In this situation, the netlister processes only the `sch4` and `sch5` views.

On the other hand, if *Use ‘View names to process’* is selected, the netlister processes only the `sch1`, `sch3`, and `sch4` views.

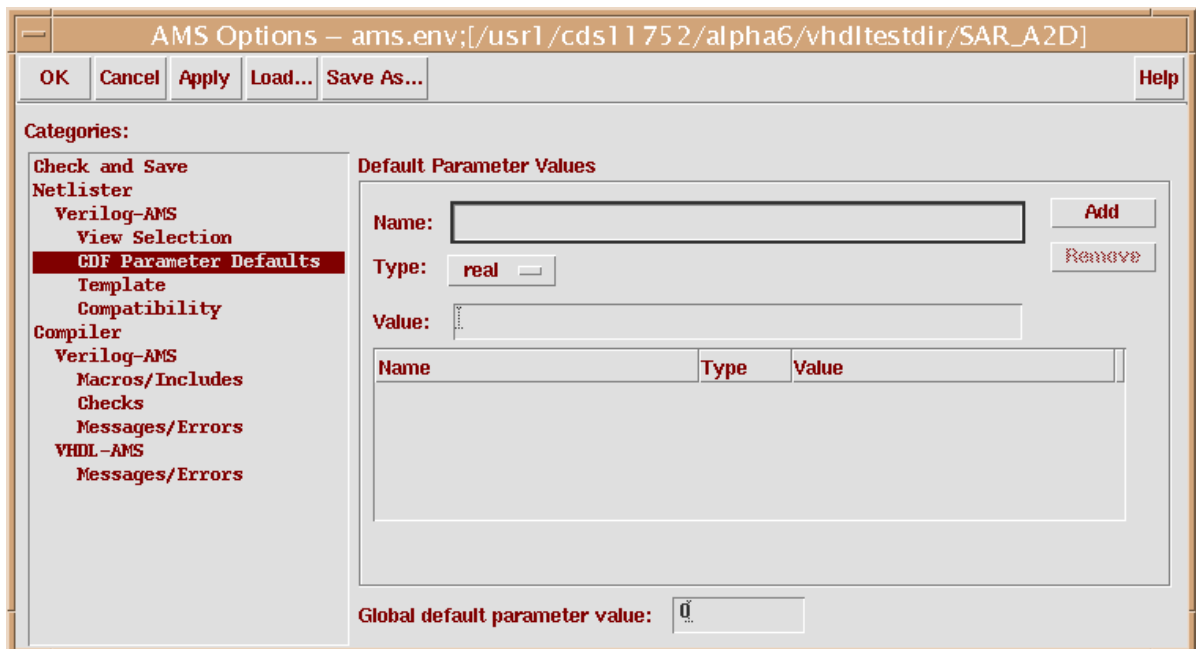
#### Setting Default Values for CDF Parameters

With the options in this category, you can specify parameter and value pairs.

To work with parameters and values,

1. Select *Netlister – Verilog-AMS – CDF Parameter Defaults* in the *Categories* field.

The associated option fields appear.



2. To add a parameter default value,

- a. Type the name of the parameter in the *Name* field.
- b. In the *Type* field, select *integer* or *real* to indicate the type of the parameter.
- c. Type a default value into the *Value* field.

The AMS netlister uses this default value when the parameter does not have a default value in the CDBA data.

- d. Click *Add*.

For information on the corresponding `ams.env` variable, see “[paramDefVals](#)” on page 464.

3. To specify a global default parameter value, type a default value into the *Global default parameter value* field.

For information on the corresponding `ams.env` variable, see “[paramGlobalDefVal](#)” on page 465.

The AMS netlister uses this default value for any parameter in the CDBA data that does not otherwise have a default value.

4. To remove a parameter, select it and then click *Remove*.

#### **Setting Template Header and File Include Options**

With the options in this category, you can specify headers for AMS netlists and the names of files to be included by ``include` directives.

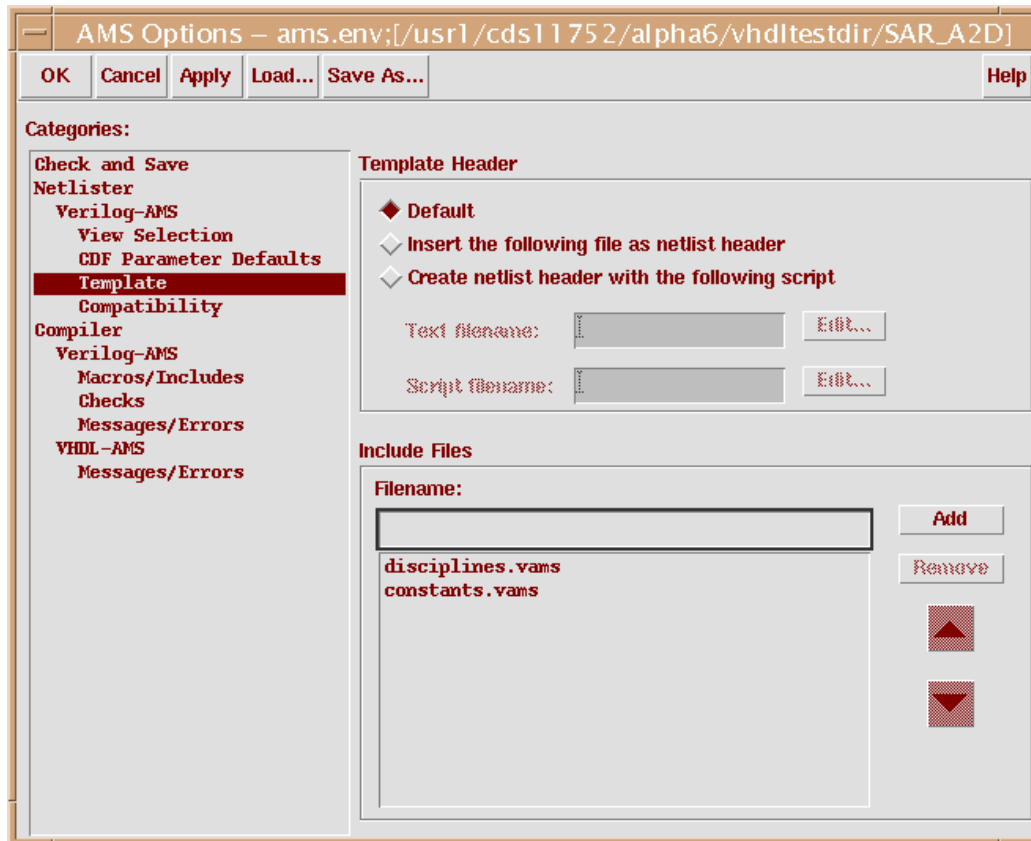
To open the pane,

- Select *Netlister – Verilog-AMS – Template* in the *Categories* field.

## Virtuoso AMS Environment User Guide

### Netlisting

The associated option fields appear.



To specify headers for netlists generated by AMS Designer,

1. In the Template Header area, select the method you want to use to specify template headers.

For information on the corresponding `ams.env` variable, see [“headerText”](#) on page 408.

**Default** Inserts the default header at the top of each generated netlist. For example, choosing *Default* inserts the following text into netlists.

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

## Virtuoso AMS Environment User Guide

### Netlisting

---

*Insert the following file as netlist header*

Inserts the default header and the contents of the specified file at the top of each generated netlist. For example, choosing *Insert the following file as netlist header* and specifying a file containing the following text

```
// Module produced by
// ASIC Team: Ocelot
// San Jose Development Center
```

inserts the following at the top of each generated netlist.

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

// Module produced by
// ASIC Team: Ocelot
// San Jose Development Center
```

*Create netlist header with the following script*

Inserts the default header and the results of running the specified script at the top of each generated netlist. For example, Choosing *Create netlist header with the following script* and specifying a file containing the following script

```
echo '// Module produced by:'
echo '// ASIC Interactive, Ltd.'
printf '// (c) '
date '+DATE: %m/%d/%y%n'
```

inserts the following at the top of each generated netlist.

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

// Module produced by:
// ASIC Interactive, Ltd.
// (c) DATE: 10/10/01
```

2. If the *Text filename* field or the *Script filename* field is active, specify the path and filename of the text file or script file.

Relative paths are resolved with respect to the directory from which you opened the CIW.

For information on the corresponding `ams.env` variables, see [“templateFile”](#) on page 551 and [“templateScript”](#) on page 552.

3. When you are done making changes to the pane, click *OK*.

To specify files to be included in the netlist by ``include` directives,

1. Type the name of the file to include in the *Filename* field.

The name that you enter appears verbatim in the netlist.

For information on the corresponding `ams.env` variable, see [“includeFiles”](#) on page 412.

## Virtuoso AMS Environment User Guide

### Netlisting

---

2. Click *Add* to add the filename to the list of filenames displayed below the *Filename* field.

The newly added file is included, with a ``include` directive, in every netlist. To control which file of that name is included, you can use the *Include Directories* table accessed by choosing *Compiler – Verilog-AMS – Macros/Includes* in the *Categories* pane. For guidance, see “[Specifying Macros and Specifying Directories to be Searched](#)” on page 118.

3. To remove a file, select the filename from the list, and then click *Remove*.
4. If necessary, move a filename up or down in the list by clicking the up and down arrows.

Files are included in the netlist in the order that they appear in the list. The order is important if you have files that use declarations in another file. For example, if `File2` uses a declaration from `File1`, you must ensure that `File1` is above `File2` in the list.

5. When you are done making changes to the pane, click *OK*.

#### Setting Compatibility Options

If you have the following design exceptions in your cellview, the AMS netlister, by default, either issues a warning or does not create a netlist.

Exception	For an explanation of the exception and a description of the corresponding <code>ams.env</code> variable, see
Illegal identifiers	<a href="#">“allowIllegalIdentifiers”</a> on page 370
Name collisions	<a href="#">“allowNameCollisions”</a> on page 372
Conflicting bus ranges	<a href="#">“allowDeviantBuses”</a> on page 368
Sparse buses	<a href="#">“allowSparseBuses”</a> on page 374

You can override this default behavior by changing the settings in the *Compatibility: Verilog-AMS options* category.

To change the settings,

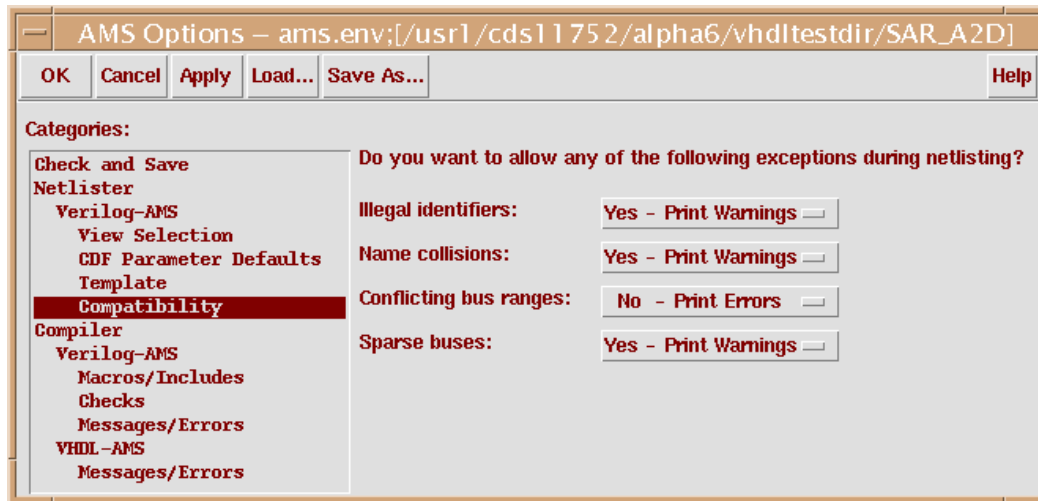
1. Select *Netlister – Verilog-AMS – Compatibility* in the *Categories* field.



## Virtuoso AMS Environment User Guide

### Netlisting

The associated option fields appear.



#### 2. Select the settings you want for each of the exceptions.

Each exception has the following possible settings:

- ☐ *No – Print Errors*: The AMS netlister issues an error if the netlister encounters the design exception. No netlist is produced.
- ☐ *Yes – Print Warnings*: The AMS netlister issues a warning if the netlister encounters the design exception. Netlisting continues after the netlister makes necessary changes.
- ☐ *Yes – Silently*: The AMS netlister continues netlisting if it encounters the design exception, after making changes that allow netlisting to continue. The AMS netlister does not issue a warning.

If you select *Yes – Print Warnings* or *Yes – Silently*, the AMS netlister makes the following changes when it encounters a design exception.

- ☐ *Illegal identifiers*: Maps non-compliant 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 non-compliant 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 generated netlist is likely to be less readable than one created from compliant bus data. The associated warning, if issued, tells you how the non-compliant bus data is transformed.
- ☐ *Sparse buses*: Overdeclares any sparse buses.

For information on the corresponding `ams.env` variables, see [“allowIllegalIdentifiers”](#) on page 370, [“allowNameCollisions”](#) on page 372, [“allowDeviantBuses”](#) on page 368, and [“allowSparseBuses”](#) on page 374.

3. When you are done making changes to the pane, click *OK*.

## Setting Compiler Options

AMS Designer provides many options for the compiler, that allow you to tailor the compiler behavior for your needs. There are two main graphical interfaces that you can use to set the options:

- Through *AMS – Options* in the Cadence hierarchy editor

For guidance on using this interface, see [“Setting Compiler Options from the Hierarchy Editor”](#) on page 98.

- Through *Tools – AMS – Options* in the CIW

For guidance on using this interface, see [“Setting Compiler Options from the CIW”](#) on page 114.

You can set the values of most variables using either of these interfaces. They both operate on the same `ams.env` file so changes that you make from one interface are reflected in the other.

### Setting Compiler Options from the Hierarchy Editor

To review or change compiler options from the Cadence hierarchy editor,

1. From the hierarchy editor, choose *AMS – Options – Compiler*.

If the *AMS* menu entry is not visible, follow the instructions in [“Preparing to Use AMS Designer from the Hierarchy Editor”](#) on page 62.

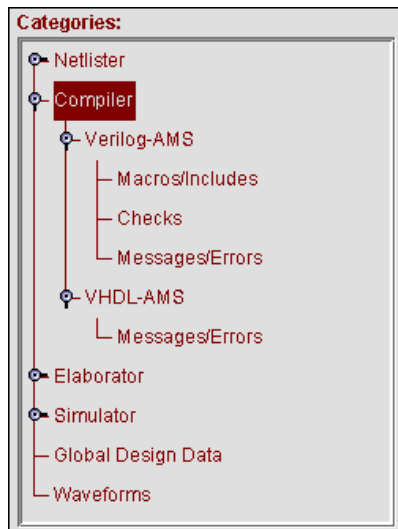
The AMS Options window appears.

## Virtuoso AMS Environment User Guide

### Netlisting

---

- Expand the *Compiler* category as necessary to find the options you want to change. When fully expanded, the Compiler category displays the following subcategories.



- Highlight the category that you want to change.

The corresponding pane appears. For information about the fields in each pane, see the following cross-references:

Category	Purpose	For information, see
<i>Compiler</i>	Specifies an <code>hdl.var</code> file to use and libraries to be excluded from compilation	<a href="#">“Setting General Compiler Options”</a> on page 100
<i>Verilog-AMS</i>	Controls how Verilog-AMS modules are compiled	<a href="#">“Setting Options for the Verilog-AMS Compiler”</a> on page 101
<i>Macros/Includes</i>	Defines macros used during compilation and directories to be searched for included source files	<a href="#">“Specifying Macros and Specifying Directories to be Searched”</a> on page 104
<i>Checks</i>	Specifies the checks to be used on Verilog-AMS modules	<a href="#">“Specifying Checks”</a> on page 106

## Virtuoso AMS Environment User Guide

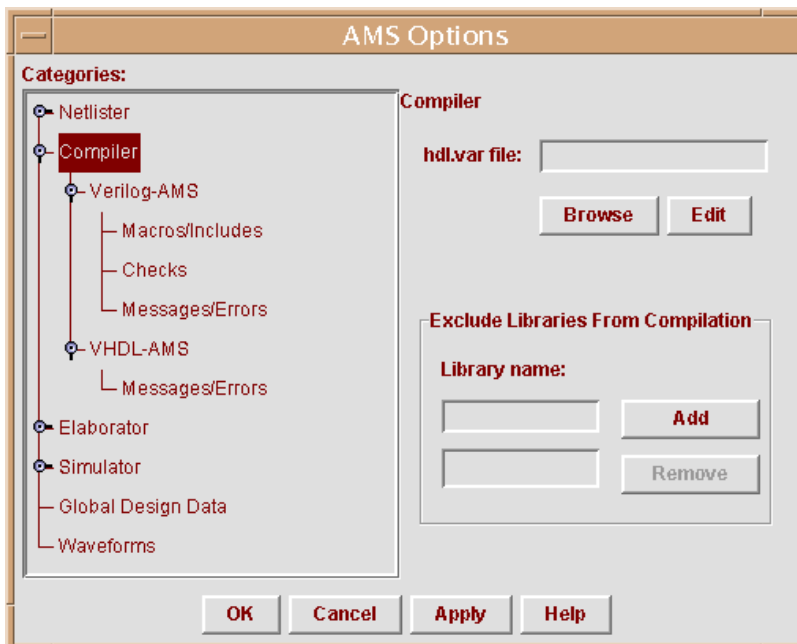
### Netlisting

Category	Purpose	For information, see
<i>Messages/Errors</i>	Controls the production and display of messages generated by compiling Verilog-AMS modules	<a href="#">“Setting Options for Messages Generated by Compiling Verilog-AMS Modules”</a> on page 107
<i>VHDL-AMS</i>	Controls how VHDL-AMS modules are compiled	<a href="#">“Setting Options for the VHDL-AMS Compiler”</a> on page 109
<i>Messages/Errors</i>	Controls the production and display of messages generated by compiling VHDL -AMS modules	<a href="#">“Setting Options for Messages Generated by Compiling VHDL-AMS Modules”</a> on page 112

#### Setting General Compiler Options

To specify an `hdl.var` file to use and to specify libraries to be excluded from compilation,

1. Select the *Compiler* category to display the *Compiler* pane.



2. Type the path and name of the `hdl.var` file into the *hdl.var file* field.

An `hdl.var` file that you specify here is used with the compilers, the elaborator, and the simulator.

Be sure that the path is fully qualified, either as an absolute path or with environment variables. Otherwise, because the compilers run in the directory from which you start the software but the elaborator and simulator run in the run directory, you might unexpectedly use different `hdl.var` files.

### 3. Specify libraries that are not to be compiled.

To enter the library names, type one name at a time into the *Library Name* field and click *Add*. The libraries to be excluded from compilation appear in the list below.

In compile all mode (such as when the `ams.env compileMode` variable is set to `"all"`), AMS Design Prep, by default, compiles every cell referenced in the design hierarchy. However, when you add names to the Exclude Libraries list, cells in the design hierarchy that belong to a library in the list are not compiled.

Be aware that, by working through the Cadence hierarchy editor, you can still compile modules included in libraries of the Exclude Libraries list. Right-clicking on a module in the *Tree*, *Table*, or *Instance Table* view of a design configuration opens a pop-up menu that includes a *Compile Netlist* command. This command overrides the more general prohibition on compilation that you can specify in this Compiler form.

For information on the corresponding `ams.env` variable, see [“compileExcludeLibs”](#) on page 397.

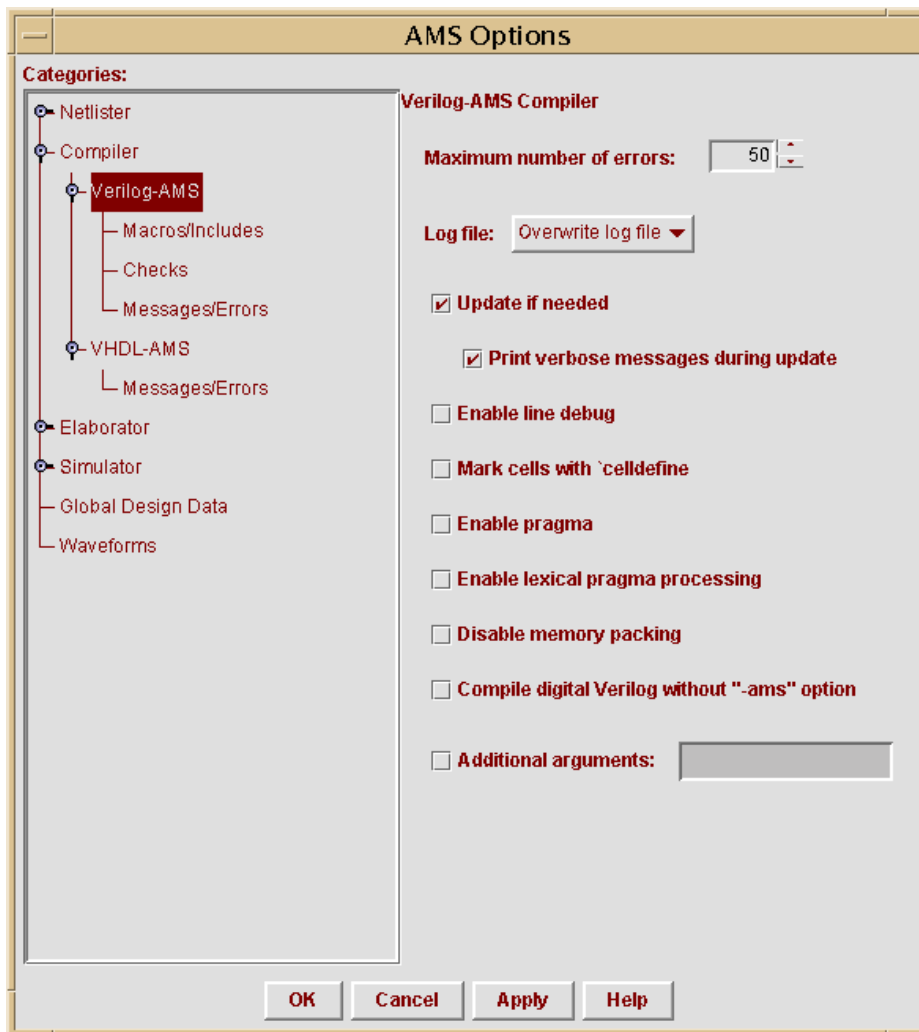
### Setting Options for the Verilog-AMS Compiler

To specify the behavior of the Verilog-AMS compiler,

## Virtuoso AMS Environment User Guide

### Netlisting

1. Select the *Compiler – Verilog-AMS* category to display the *Verilog-AMS Compiler* pane.



2. Fill in and select fields as necessary.

The following table briefly describes the fields. For additional information, see the “ncvlog Command Options” section, in the “Compiling Verilog Source Files with ncvlog” chapter of the *NC-Verilog Simulator Help*.

Field	Corresponding ncvlog Option	Effect
<i>Maximum number of errors</i>	-errormax	Stops compilation if the number of errors reaches the specified maximum limit.

## Virtuoso AMS Environment User Guide

### Netlisting

Field	Corresponding ncvlog Option	Effect
<i>Log file</i>	-append_log, -nolog	Controls the generation of log files.
<i>Update if needed</i>	-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.
<i>Print verbose messages during update</i>	-uptodate_messages	Prints the names of up-to-date modules that otherwise are not printed in the log file.
<i>Enable line debug</i>	-linedebug	Enables support for setting line breakpoints and for single-stepping through code.
<i>Mark cells with 'celldefine</i>	-libcell	Inserts 'celldefine and 'endcelldefine compiler directives to tag module instances as cell instances.
<i>Enable pragma</i>	-pragma	Parses pragmas contained in HDL source files. This field is grayed out when <i>Enable lexical pragma processing</i> is selected.
<i>Enable lexical pragma processing</i>	-lexpragma	Parses pragmas contained in HDL source files and treats <code>translate off</code> and <code>translate on</code> as if they are Verilog <code>`ifdef 0</code> and <code>`endif</code> so that the code between them is not included during compilation.
<i>Disable memory packing</i>	-nomempack	Prepares design units for access by the PLI routine <code>tf_nodeinfo</code> .

## Virtuoso AMS Environment User Guide

### Netlisting

Field	Corresponding ncvlog Option	Effect
<i>Compile digital Verilog without "-ams" option</i>	-ams	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.
<i>Additional arguments</i>	None	See <a href="#">Step 3</a> , following.

3. Type any additional arguments that you want the Verilog-AMS compiler to use into the *Additional arguments* field.

You must not specify a -log argument because the log is automatically written to the ncvlog.log file in the run directory.

#### Specifying Macros and Specifying Directories to be Searched

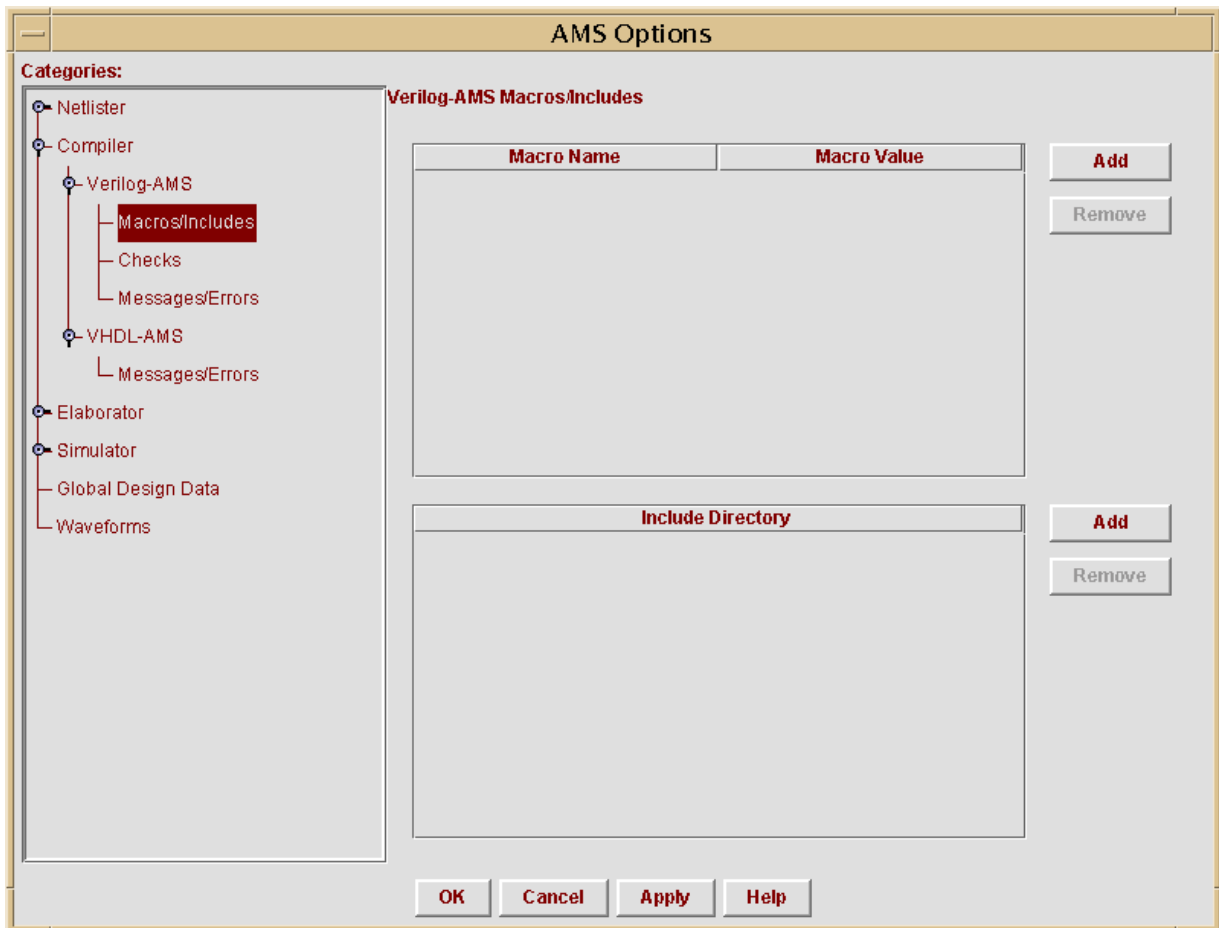
To set options that define macros and specify directories to be searched for include files,



## Virtuoso AMS Environment User Guide

### Netlisting

1. Select the *Compiler – Verilog-AMS – Macros/Includes* category to display the *Verilog-AMS Macros/Includes* pane.



2. To define a macro,
  - a. Click *Add* next to the *Macro Value* column.

A default macro name appears in the *Macro Name* column.
  - b. Click the default macro name and change it as necessary.

For more information, see the information about the `-define` option in the “ncvlog Command Options” section, in the “Compiling Verilog Source Files with ncvlog” chapter of the *NC-Verilog Simulator Help*.
  - c. Click the corresponding *Macro Value* cell and type in the value, if any, that you want to use.

## Virtuoso AMS Environment User Guide

### Netlisting

---

If the value includes a space, enclose the value in quotation marks. For example, you might define a macro as follows:

Macro Name	Macro Value
scanspeed	"very fast"

3. To specify directories to be searched for include files,

a. Click *Add* next to the *Include Directory* table.

A default directory name appears in the *Include Directory* column.

b. Click the default directory name and change it to the actual directory to be searched.

The `-incdir` option is added to the `ncvlog` command. For example, you specify `include_dir` in the *Include Directory* list. When you compile a Verilog-AMS file, the `ncvlog` command generated by AMS Designer has the following:

```
ncvlog
-incdir include_dir
```

For more information, see the information about the `-incdir` option in the “`ncvlog` Command Options” section of the “Compiling Verilog Source Files with `ncvlog`” in the *NC-Verilog Simulator Help*.

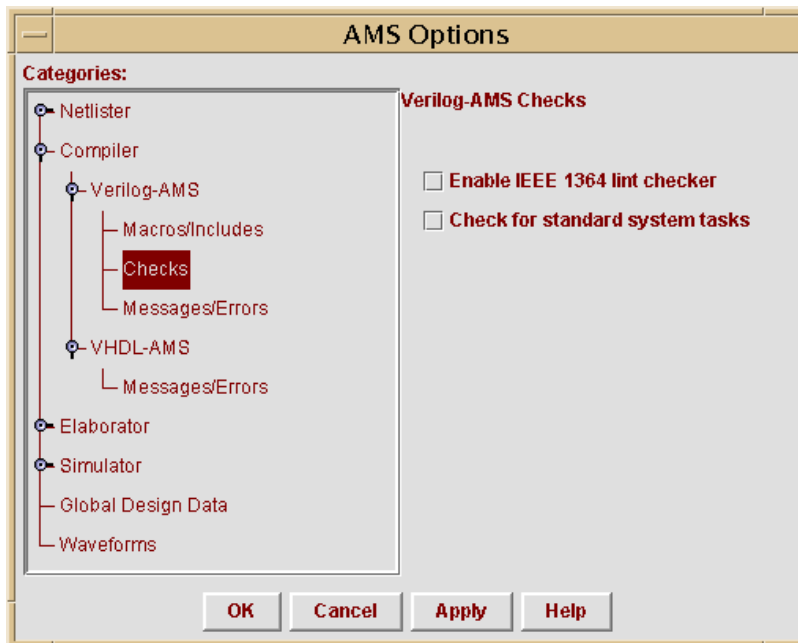
### Specifying Checks

To set options that control checks on Verilog-AMS modules,

## Virtuoso AMS Environment User Guide

### Netlisting

1. Select the *Compiler – Verilog-AMS – Checks* category to display the *Verilog-AMS Checks* pane.



2. Select fields as necessary.

The following table briefly describes the fields. For additional information, see the “ncvlog Command Options” section, in the “Compiling Verilog Source Files with ncvlog” chapter of the *NC-Verilog Simulator Help*.

Field	Corresponding ncvlog Option	Effect
<i>Enable IEEE 1364 lint checker</i>	<code>-ieee1364</code>	Checks the source code for compatibility with the IEEE standard described in <i>IEEE-1364 Verilog Hardware Description Language Reference Manual</i> .
<i>Check for standard system tasks</i>	<code>-checktasks</code>	Checks for the presence of any non-predefined system tasks or functions in the source code.

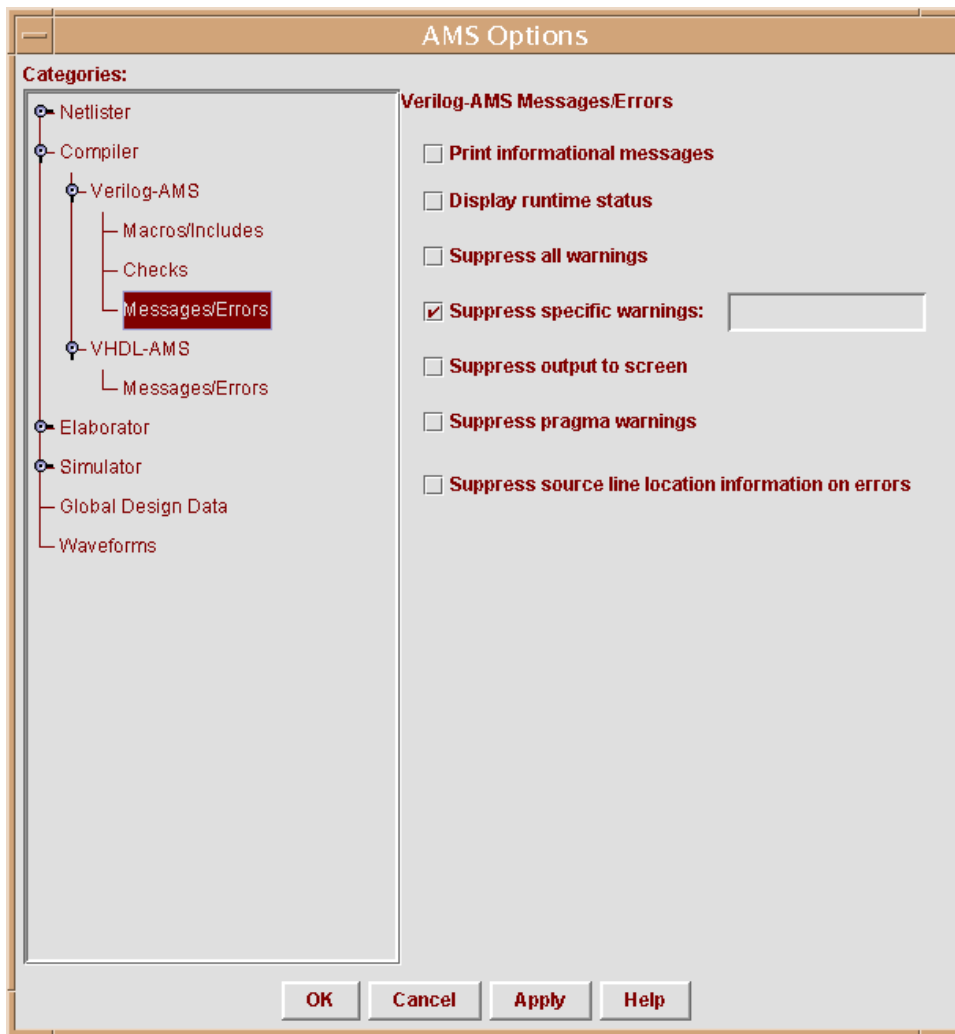
#### Setting Options for Messages Generated by Compiling Verilog-AMS Modules

To set options that control the output of messages and errors,

## Virtuoso AMS Environment User Guide

### Netlisting

1. Select the *Compiler – Verilog-AMS – Messages/Errors* category to display the *Verilog-AMS Messages/Errors* pane.



2. Select and fill in fields as necessary.

The following table briefly describes the fields. For additional information, see the “ncvlog Command Options” section, in the “Compiling Verilog Source Files with ncvlog” chapter of the *NC-Verilog Simulator Help*.

Field	Corresponding ncvlog Option	Effect
<i>Print informational messages</i>	-messages	Prints informational messages as the compiler runs.

## Virtuoso AMS Environment User Guide

### Netlisting

Field	Corresponding ncvlog Option	Effect
<i>Display runtime status</i>	-status	Prints statistics on memory and CPU usage after compilation.
<i>Suppress all warnings</i>	-neverwarn	Suppresses all warning messages.
<i>Suppress specific warnings</i>	-nowarn	Suppresses warning messages that have specified codes. If you enter more than one code, separate them with commas or spaces.
<i>Suppress output to screen</i>	-nostdout	Suppresses output to the screen but does not change what is written to the log file.
<i>Suppress pragma warnings</i>	-nopragswarn	Suppresses warning messages related to pragmas.
<i>Suppress source line location information on errors</i>	-noline	Tells the compiler not to locate the source line of errors, potentially improving performance.

#### Setting Options for the VHDL-AMS Compiler

To specify the behavior of the VHDL-AMS compiler,

1. Select the *Compiler – VHDL-AMS* category.

## Virtuoso AMS Environment User Guide

### Netlisting

AMS Designer displays the *VHDL-AMS Compiler* pane.



#### 2. Select and fill in fields as necessary.

The following table briefly describes the fields. For additional information, see the “ncvhdI Command Options” section, in the “Compiling VHDL Source Files with ncvhdI” chapter of the *NC-VHDL Simulator Help*.

Field	Corresponding ncvhdI Option	Effect
<i>Maximum number of errors</i>	-errormax	Stops compilation if the number of errors reaches the specified maximum limit.

## Virtuoso AMS Environment User Guide

### Netlisting

Field	Corresponding ncvhdl Option	Effect
<i>Log file</i>	-append_log, -nolog	Controls the generation of log files
<i>Update if needed</i>	-update	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.  <b>Note:</b> This field should be selected in almost all circumstances. Not selecting this field can produce timestamp mismatches between entities and their corresponding architectures.
<i>Enable line debug</i>	-linedebug	Enables support for setting line and process breakpoints, and for single-stepping through code.
<i>Enable VITAL checks</i>	-novitalcheck	Turns on VITAL compliance checking.
<i>Enable relaxed VHDL interpretation</i>	-relax	Relaxes the interpretation of some VHDL rules.
<i>Enable pragma</i>	-pragma	Parses pragmas contained in HDL source files. This field is grayed out when <i>Enable lexical pragma processing</i> is selected.
<i>Enable lexical pragma processing</i>	-lexpragma	Parses pragmas contained in HDL source files and treats <code>translate off</code> and <code>translate on</code> as if they are Verilog <code>`ifdef 0</code> and <code>`endif</code> so that the code between them is not included during compilation.

## Virtuoso AMS Environment User Guide

### Netlisting

Field	Corresponding ncvhdl Option	Effect
<i>Compile digital VHDL without "-ams" option</i>	-ams	Omits the -ams command line option when running ncvhdl on files named verillog.vhd. If a file named verillog.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.  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.
<i>Enable VHDL 93 features for digital VHDL</i>	-v93	Enables the VHDL-93 features supported in this release. The -v93 option is used automatically whenever the -ams option is used on a VHDL cellview.
<i>Additional arguments</i>	None	See <a href="#">Step 3</a> , following.

3. Type any additional arguments that you want the VHDL compiler to use into the *Additional arguments* field.

You must not specify a -logfile argument because the log is automatically written to the ncvhdl.log file in the run directory.

#### Setting Options for Messages Generated by Compiling VHDL-AMS Modules

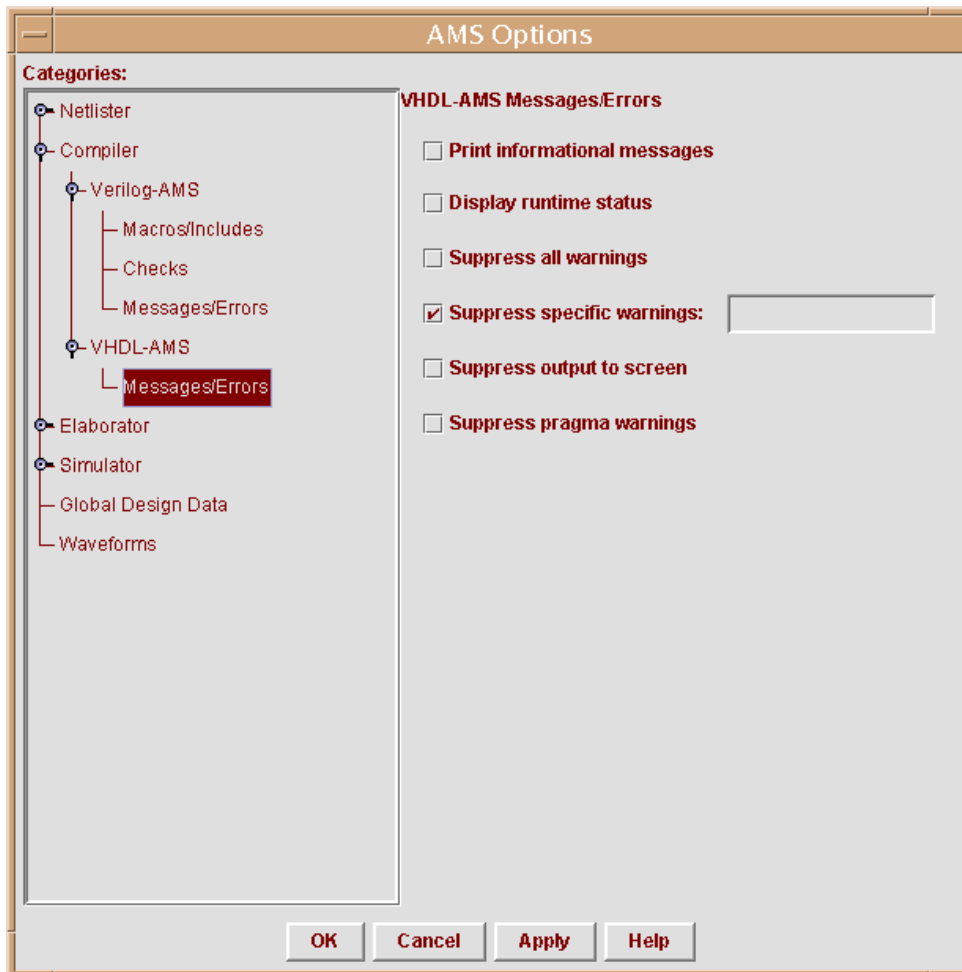
To set options that control the output of messages and errors from the VHDL-AMS compiler,



## Virtuoso AMS Environment User Guide

### Netlisting

1. Select the *Compiler – VHDL-AMS – Messages/Errors* category to display the *VHDL-AMS Messages/Errors* pane.



2. Select and fill in fields as necessary.

The following table briefly describes the fields. For additional information, see the “ncvhdI Command Options” section, in the “Compiling VHDL Source Files with ncvhdI” chapter of the *NC-VHDL Simulator Help*.

Field	Corresponding ncvhdI Option	Effect
<i>Print informational messages</i>	-messages	Prints informational messages as the compiler runs.
<i>Display runtime status</i>	-status	Prints statistics on memory and CPU usage after compilation.

## Virtuoso AMS Environment User Guide

### Netlisting

Field	Corresponding ncvhdl Option	Effect
<i>Suppress all warnings</i>	-neverwarn	Suppresses all warning messages.
<i>Suppress specific warnings</i>	-nowarn	Suppresses warning messages that have specified codes. If you enter more than one code, separate them with commas or spaces.
<i>Suppress output to screen</i>	-nostdout	Suppresses printing of output to the screen but does not change what is written to the log file.
<i>Suppress pragma warnings</i>	-nopragsmawarn	Suppresses warning messages related to pragmas.

### Setting Compiler Options from the CIW

The compiler options available from the CIW interface are organized in the following categories:

Category	Purpose	For information, see
<i>Compiler</i>	Specifies an <code>hdl.var</code> file to use and whether the <code>-ams</code> option is used	<a href="#">“Setting General Compiler Options”</a> on page 115
<i>Verilog-AMS</i>	Controls how Verilog-AMS modules are compiled	<a href="#">“Setting Options for the Verilog-AMS Compiler”</a> on page 116
<i>Macros/Includes</i>	Defines macros used during compilation and directories to be searched for included source files	<a href="#">“Specifying Macros and Specifying Directories to be Searched”</a> on page 118
<i>Checks</i>	Specifies the checks to be used on Verilog-AMS modules	<a href="#">“Specifying Checks”</a> on page 119

## Virtuoso AMS Environment User Guide

### Netlisting

---

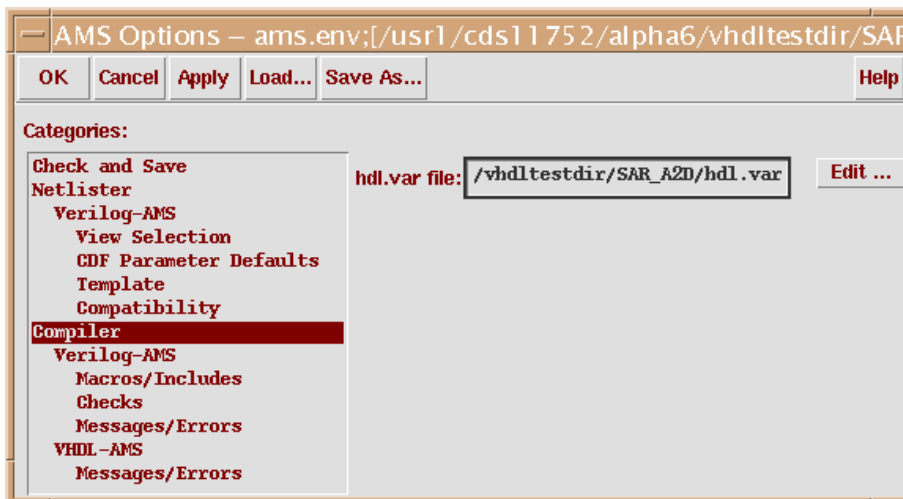
Category	Purpose	For information, see
<i>Messages/Errors</i>	Controls the production and display of messages generated by compiling Verilog-AMS modules	<a href="#">“Setting Options for Messages Generated by Compiling Verilog-AMS Modules”</a> on page 120
<i>VHDL-AMS</i>	Controls how VHDL modules are compiled	<a href="#">“Setting Options for VHDL-AMS Modules”</a> on page 122
<i>Messages/Errors</i>	Controls the production and display of messages generated by compiling VHDL-AMS modules	<a href="#">“Setting Options for Messages Generated by Compiling VHDL-AMS Modules”</a> on page 124

#### Setting General Compiler Options

To specify an `hdl.var` file to use,

1. Select *Compiler* in the *Categories* field.

The associated option field appears.



2. Type the path and name of the `hdl.var` file into the *hdl.var file* field.

An `hdl.var` file that you specify here is used with the compilers, and also with the elaborator and the simulator if you run those tools.

Be sure that the path is fully qualified, either as an absolute path or with environment variables. Otherwise, because the compilers run in the directory from which you start the

## Virtuoso AMS Environment User Guide

### Netlisting

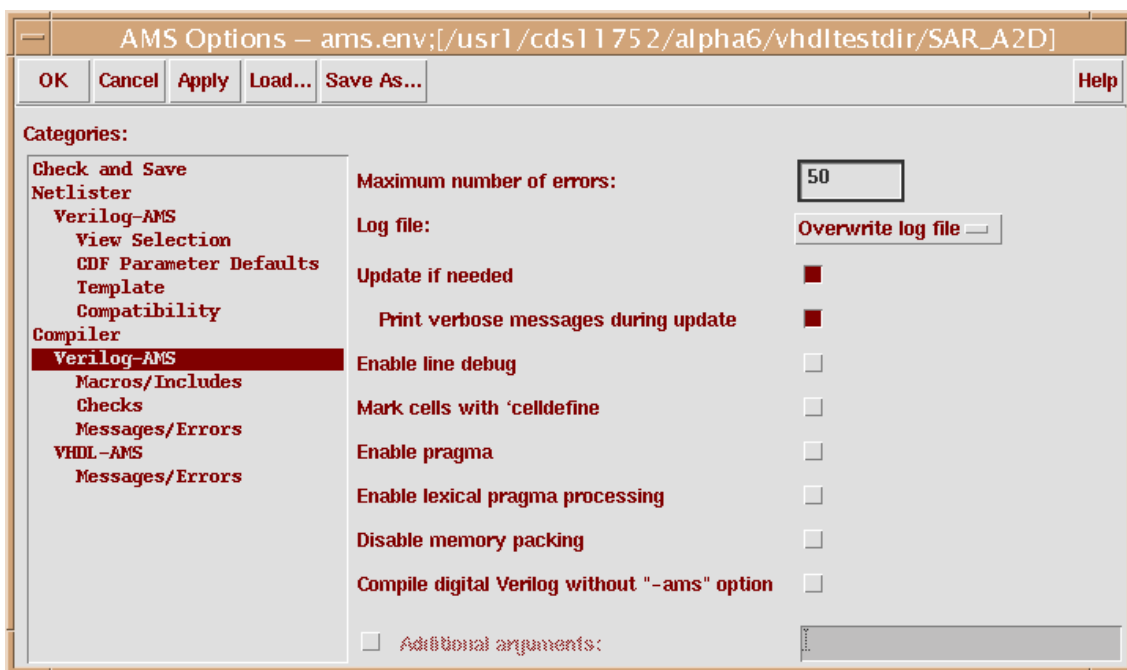
software but the elaborator and simulator run in the run directory, you might unexpectedly use different `hdl.var` files for different AMS tools.

#### Setting Options for the Verilog-AMS Compiler

To specify the behavior of the Verilog-AMS compiler,

1. Select *Compiler – Verilog-AMS* in the *Categories* field.

The associated option fields appear.



2. Fill in and select fields as necessary.

The following table briefly describes the fields. For additional information, see the “ncvlog Command Options” section, in the “Compiling Verilog Source Files with ncvlog” chapter of the *NC-Verilog Simulator Help*.

Field	Corresponding ncvlog Option	Effect
<i>Maximum number of errors</i>	<code>-errormax</code>	Stops compilation if the number of errors reaches the specified maximum limit.
<i>Log file</i>	<code>-append_log</code> , <code>-nolog</code>	Controls the generation of log files.

## Virtuoso AMS Environment User Guide

### Netlisting

Field	Corresponding ncvlog Option	Effect
<i>Update if needed</i>	-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.
<i>Print verbose messages during update</i>	-uptodate_ messages	Prints the names of up-to-date modules that otherwise are not printed in the log file.
<i>Enable line debug</i>	-linedebug	Enables support for setting line breakpoints and for single-stepping through code.
<i>Mark cells with 'celldefine</i>	-libcell	Inserts <code>`celldefine</code> and <code>`endcelldefine</code> compiler directives to tag module instances as cell instances.
<i>Enable pragma</i>	-pragma	Parses pragmas contained in HDL source files. This field is grayed out when <i>Enable lexical pragma processing</i> is selected.
<i>Enable lexical pragma processing</i>	-lexpragma	Parses pragmas contained in HDL source files and treats <code>translate off</code> and <code>translate on</code> as if they are Verilog <code>`ifdef 0</code> and <code>`endif</code> so that the code between them is not included during compilation.
<i>Disable memory packing</i>	-nomempack	Prepares design units for access by the PLI routine <code>tf_nodeinfo</code> .
<i>Compile digital Verilog without "-ams" option</i>	-ams	Omits the <code>-ams</code> command line option when running <code>ncvlog</code> on files named <code>verilog.v</code> . If a file named <code>verilog.v</code> is actually a link, the decision to use or omit the <code>-ams</code> option is based on the extension of the name of the physical file that is the target of the link.
<i>Additional arguments</i>	None	See <a href="#">Step 3</a> , following.

## Virtuoso AMS Environment User Guide

### Netlisting

3. Type any additional arguments that you want the Verilog-AMS compiler to use into the *Additional arguments* field.

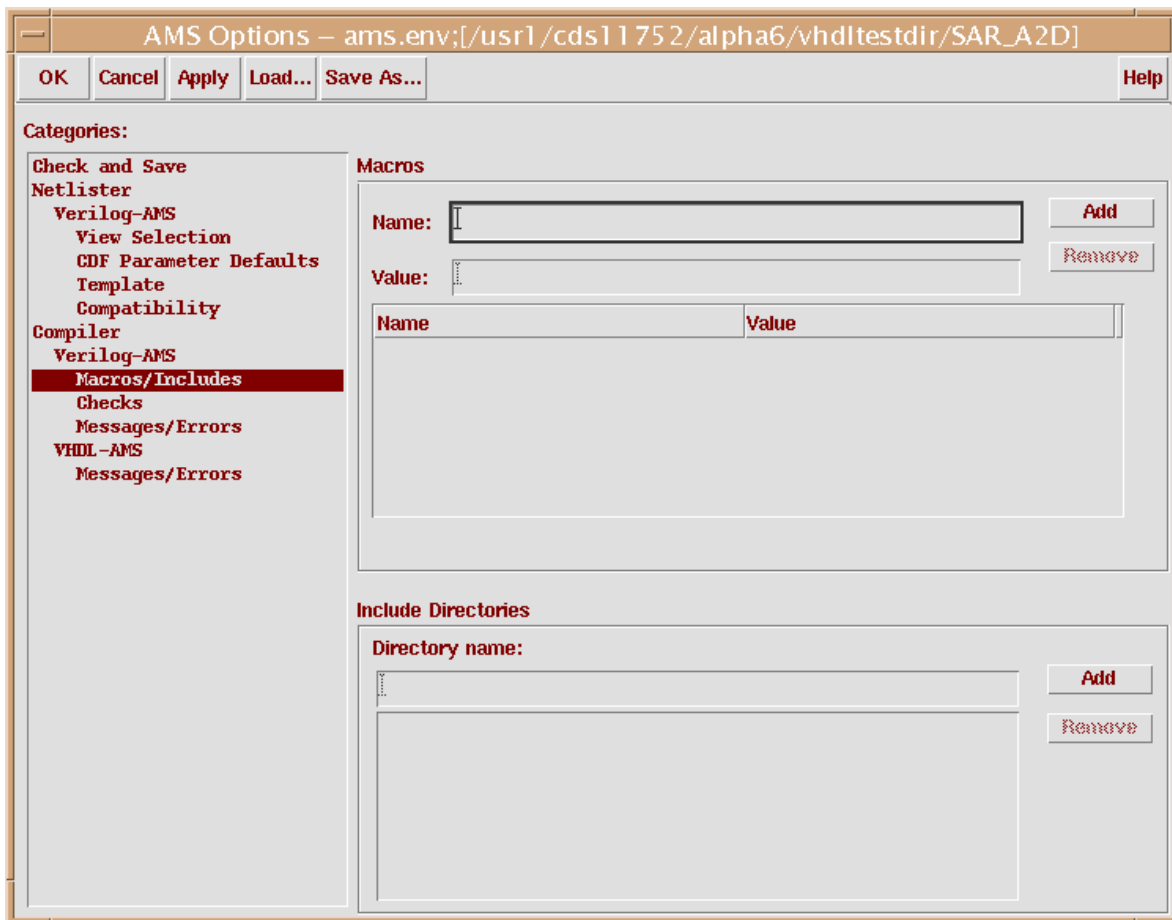
You must not specify a `-log` argument because the log is automatically written to the `ncvlog.log` file in the run directory.

#### Specifying Macros and Specifying Directories to be Searched

To set options that define macros and specify directories to be searched for include files,

1. Select the *Compiler – Verilog-AMS – Macros/Includes* in the *Categories* field.

The associated option fields appear.



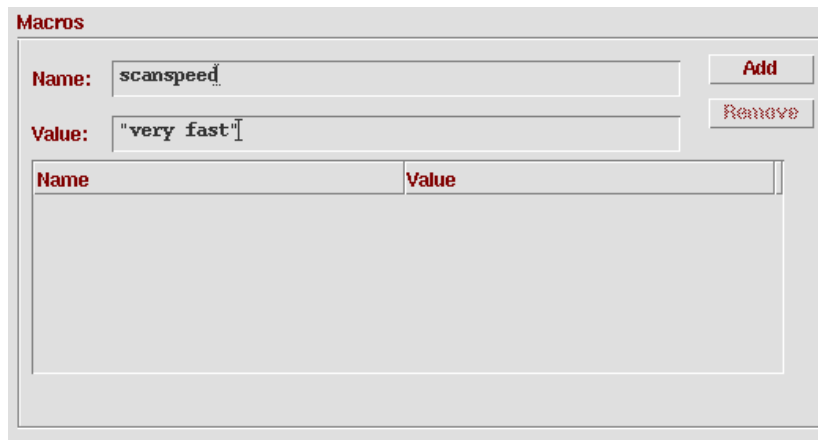
2. To define a macro,
  - a. Type a name into the *Name* field.
  - b. Type the macro value, if any, into the *Value* field.

## Virtuoso AMS Environment User Guide

### Netlisting

---

If the value includes a space, enclose the value in quotation marks. For example, you might define a macro as follows:



Name	Value
------	-------

- c. Click the *Add* button for *Macros*.

The macro name and value appear in the table.

For more information about macros, see the information about the `-define` option in the “ncvlog Command Options” section, in the “Compiling Verilog Source Files with ncvlog” chapter of the *NC-Verilog Simulator Help*.

3. To specify a directory to be searched for include files,
  - a. Type the name of the directory into the *Directory name* field.
  - b. Click the nearby *Add* button.

The `-incdir` option is added to the `ncvlog` command. For example, you specify `include_dir` in the *Directory name* list. When you compile a Verilog-AMS file, the `ncvlog` command generated by AMS Designer has the following:

```
ncvlog
-incdir include_dir
```

For more information, see the information about the `-incdir` option in the “ncvlog Command Options” section of the “Compiling Verilog Source Files with ncvlog” in the *NC-Verilog Simulator Help*.

### Specifying Checks

To set options that control checks on Verilog-AMS modules,

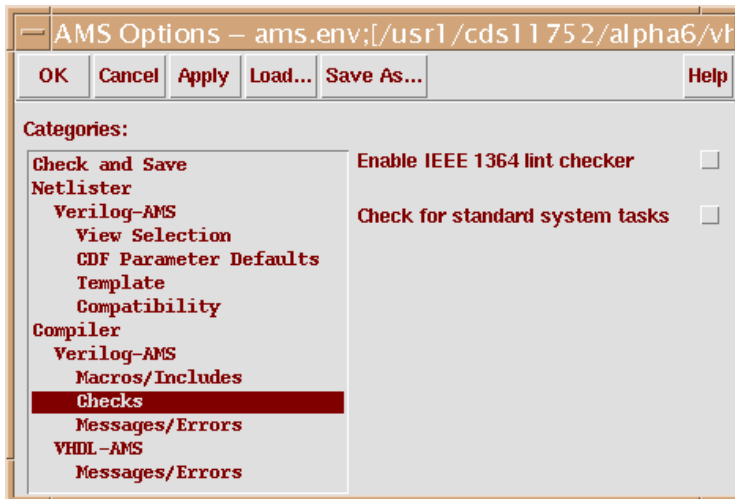
1. Select *Compiler – Verilog-AMS – Checks* in the *Categories* field.

## Virtuoso AMS Environment User Guide

### Netlisting

---

The associated option fields appear.



#### 2. Select fields as necessary.

The following table briefly describes the fields. For additional information, see the “ncvlog Command Options” section, in the “Compiling Verilog Source Files with ncvlog” chapter of the *NC-Verilog Simulator Help*.

Field	Corresponding ncvlog Option	Effect
<i>Enable IEEE 1364 lint checker</i>	<code>-ieee1364</code>	Checks the source code for compatibility with the IEEE standard described in <i>IEEE-1364 Verilog Hardware Description Language Reference Manual</i> .
<i>Check for standard system tasks</i>	<code>-checktasks</code>	Checks for the presence of any non-predefined system tasks or functions in the source code.

#### Setting Options for Messages Generated by Compiling Verilog-AMS Modules

To set options that control the output of messages and errors,

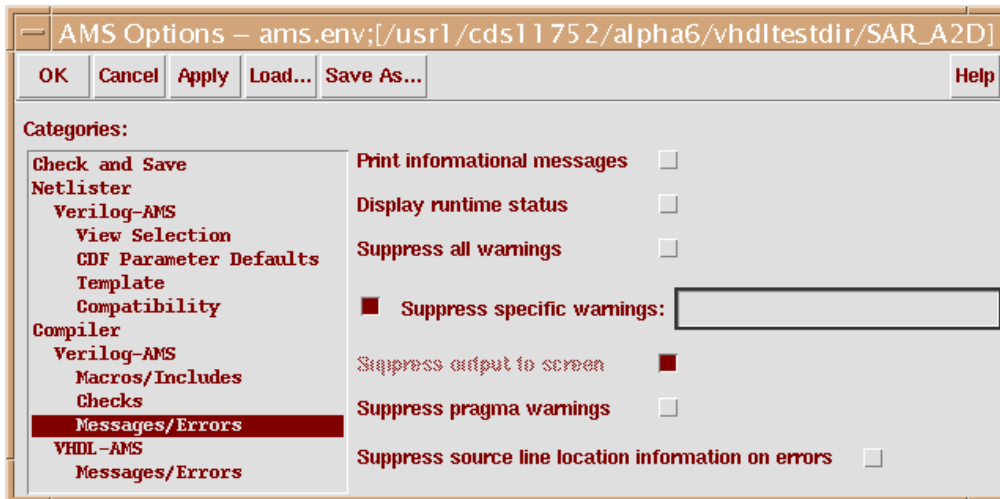
1. Select *Compiler – Verilog-AMS – Messages/Errors* in the *Categories* field.



## Virtuoso AMS Environment User Guide

### Netlisting

The associated option fields appear.



#### 2. Select and fill in fields as necessary.

The following table briefly describes the fields. For additional information, see the “ncvlog Command Options” section, in the “Compiling Verilog Source Files with ncvlog” chapter of the *NC-Verilog Simulator Help*.

Field	Corresponding ncvlog Option	Effect
<i>Print informational messages</i>	-messages	Prints informational messages as the compiler runs.
<i>Display runtime status</i>	-status	Prints statistics on memory and CPU usage after compilation.
<i>Suppress all warnings</i>	-neverwarn	Suppresses all warning messages.
<i>Suppress specific warnings</i>	-nowarn	Suppresses warning messages that have specified codes. If you enter more than one code, separate them with commas or spaces.
<i>Suppress output to screen</i>	-nostdout	Suppresses output to the screen but does not change what is written to the log file.
<i>Suppress pragma warnings</i>	-nopragmawarn	Suppresses warning messages related to pragmas.

## Virtuoso AMS Environment User Guide

### Netlisting

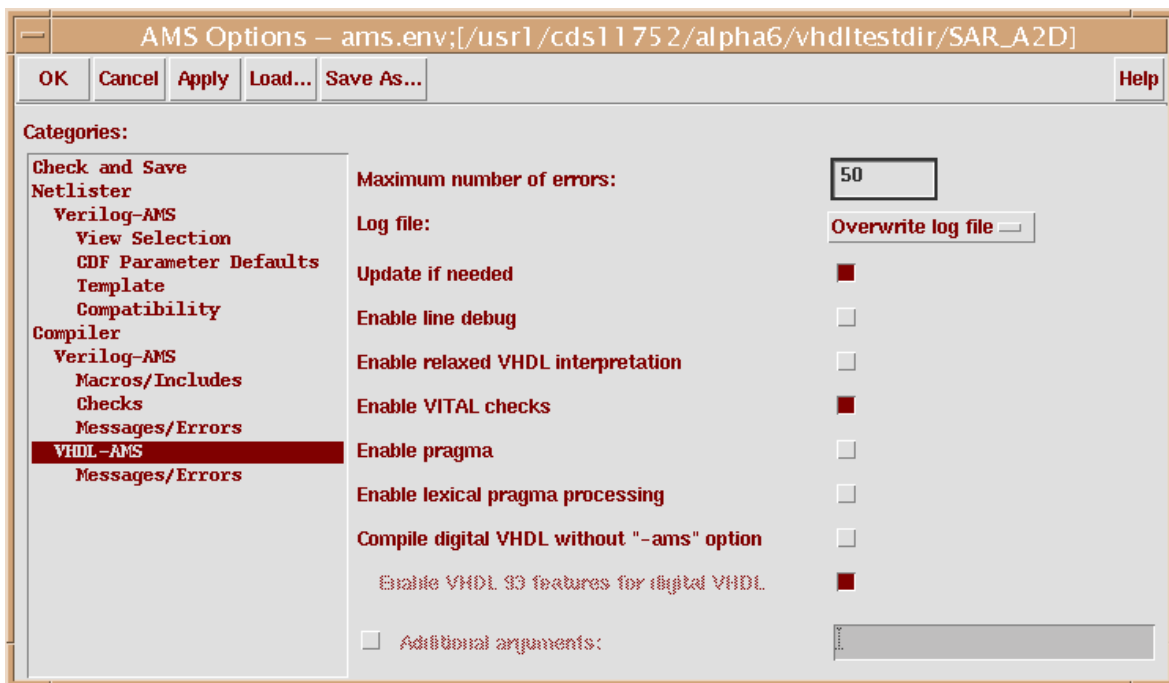
Field	Corresponding ncvlog Option	Effect
<i>Suppress source line location information on errors</i>	<code>-noline</code>	Tells the compiler not to locate the source line of errors, potentially improving performance.

#### Setting Options for VHDL-AMS Modules

To specify the behavior of the VHDL-AMS compiler,

1. Select *Compiler – VHDL-AMS* in the *Categories* field.

The associated option fields appear.



2. Select and fill in fields as necessary.

## Virtuoso AMS Environment User Guide

### Netlisting

---

The following table briefly describes the fields. For additional information, see the “ncvhdl Command Options” section, in the “Compiling VHDL Source Files with ncvhdl” chapter of the *NC-VHDL Simulator Help*.

Field	Corresponding ncvhdl Option	Effect
<i>Maximum number of errors</i>	-errormax	Stops compilation if the number of errors reaches the specified maximum limit.
<i>Log file</i>	-append_log, -nolog	Controls the generation of log files
<i>Update if needed</i>	-update	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.
<i>Enable line debug</i>	-linedebug	Enables support for setting line and process breakpoints, and for single-stepping through code.
<i>Enable relaxed VHDL interpretation</i>	-relax	Relaxes the interpretation of some VHDL rules.
<i>Enable VITAL checks</i>	-novitalcheck	Turns on VITAL compliance checking.
<i>Enable pragma</i>	-pragma	Parses pragmas contained in HDL source files. This field is grayed out when <i>Enable lexical pragma processing</i> is selected.
<i>Enable lexical pragma processing</i>	-lexpragma	Parses pragmas contained in HDL source files and treats <code>translate off</code> and <code>translate on</code> as if they are Verilog <code>`ifdef 0</code> and <code>`endif</code> so that the code between them is not included during compilation.

## Virtuoso AMS Environment User Guide

### Netlisting

Field	Corresponding ncvhdl Option	Effect
<i>Compile digital VHDL without "-ams" option</i>	-ams	Omits the -ams command line option when running ncvhdl on files named verillog.vhd. If a file named verillog.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.  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.
<i>Enable VHDL 93 features for digital VHDL</i>	-v93	Enables the VHDL-93 features supported in this release. The -v93 option is used automatically whenever the -ams option is used on a VHDL cellview.
<i>Additional arguments</i>	None	See <a href="#">Step 3</a> , following.

3. Type any additional arguments that you want the VHDL-AMS compiler to use into the *Additional arguments* field.

You must not specify a -logfile argument because the log is automatically written to the ncvhdl.log file in the run directory.

#### Setting Options for Messages Generated by Compiling VHDL-AMS Modules

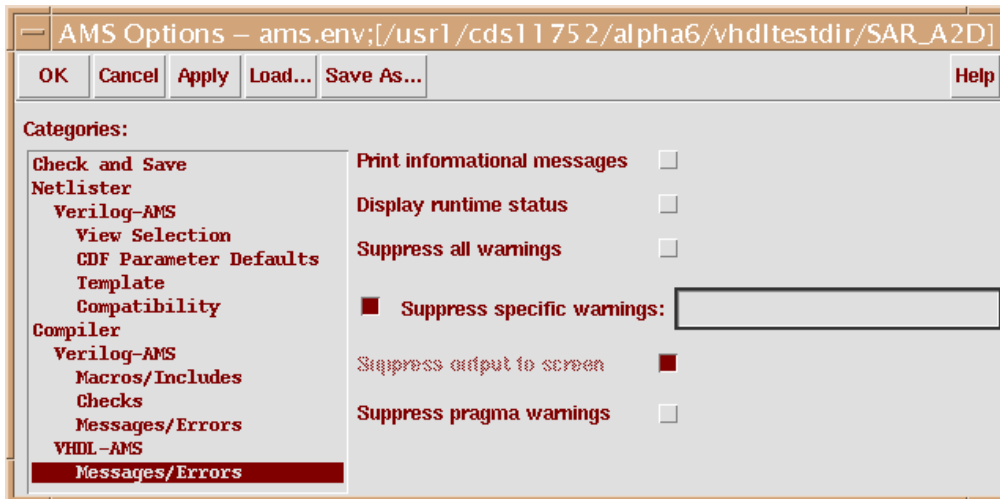
To set options that control the output of messages and errors from the VHDL-AMS compiler,

1. Select *Compiler – VHDL-AMS – Messages/Errors* in the *Categories* field.

## Virtuoso AMS Environment User Guide

### Netlisting

The associated option fields appear.



#### 2. Select and fill in fields as necessary.

The following table briefly describes the fields. For additional information, see the “ncvhdl Command Options” section, in the “Compiling VHDL Source Files with ncvhdl” chapter of the *NC-VHDL Simulator Help*.

Field	Corresponding ncvhdl Option	Effect
<i>Print informational messages</i>	-messages	Prints informational messages as the compiler runs.
<i>Display runtime status</i>	-status	Prints statistics on memory and CPU usage after compilation.
<i>Suppress all warnings</i>	-neverwarn	Suppresses all warning messages.
<i>Suppress specific warnings</i>	-nowarn	Suppresses warning messages that have specified codes. If you enter more than one code, separate them with commas or spaces.
<i>Suppress output to screen</i>	-nostdout	Suppresses printing of output to the screen but does not change what is written to the log file.
<i>Suppress pragma warnings</i>	-nopragswarn	Suppresses warning messages related to pragmas.

## Viewing the AMS Netlister Log

If you want to examine the log produced by the AMS netlister, choose *AMS – Logfile Viewer – Netlister Log File* from the Cadence hierarchy editor menu. The log contains information about the blocks that are translated. For example, the following log describes where two netlists are written.

```
@(#) $CDS: amsdirect version 4.4.6 06/21/2000 19:28 (cds11607) $
Copyright (c) 1999 Cadence Design Systems. All Rights Reserved.

Run date: Thu Jun 22 15:40:42 2000
Run time options used:
    -mpshost cds11752 -mpssession icms8457 -pid 8628

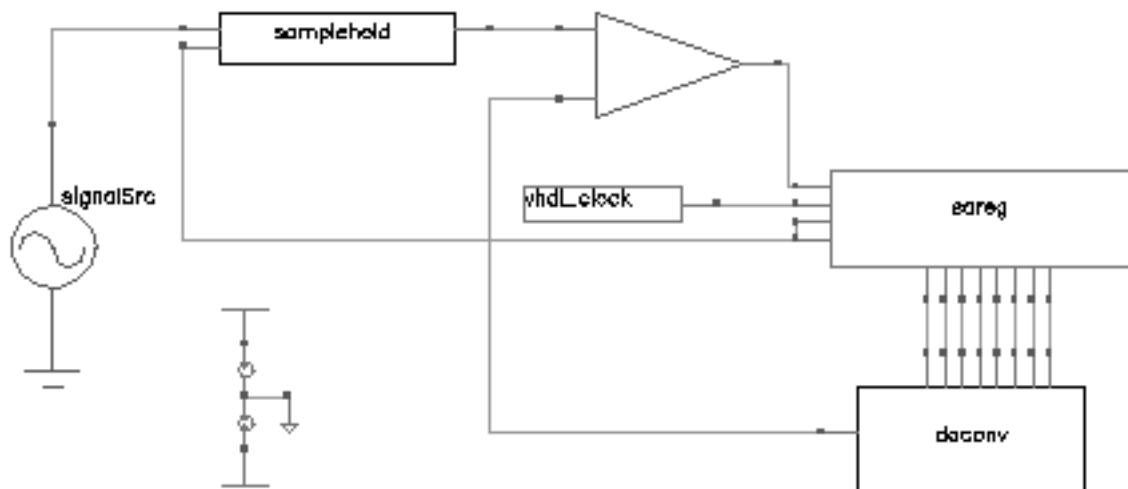
Info: Processing ("amslib" "top" "schematic") ...
Info: Verilog-AMS netlist successfully written to
    /opt/mnt4/lorenp/alpha2/alpha/AMS_lib/amsLib/top/schematic/verilog.vams.
Info: Found 0 errors and 0 warnings.

Info: Processing ("amslib" "comparator" "schematic") ...
Info: Verilog-AMS netlist successfully written to
    /opt/mnt4/lorenp/alpha2/
```

## Understanding the Output from the AMS Netlister

The output of a successful netlisting run is one or more files named `verilog.vams`, each containing a legal Verilog-AMS module that corresponds to a CDBA cellview. The netlister places each output file in the cellview directory.

For example, consider the following schematic.



## Virtuoso AMS Environment User Guide

### Netlisting

---

The AMS netlister translates this top-level schematic into the following Verilog-AMS netlist. In the netlist, notice the attributes (set off between `( * and * )`) used to pass information to the elaborator. Notice too, the out-of-module references to objects in the `cds_globals` module.

```
// 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
```

The previous netlist does not illustrate how the AMS netlister handles inherited connections, which, as described in the following sections, are also translated as attributes.

## How Inherited Connections Are Netlisted

An inherited connection is a net expression associated with either a signal or terminal. You use inherited connections to selectively override global names within your design.

The syntax for the net expression used in the Virtuoso® Schematic Editor is:

```
[@property_name:%:default_net_name]*
```

where *property\_name* is the name of the property whose value can redefine the global signal name and *default\_net\_name* is the global signal name if it is not redefined by the specified property. The *default\_net\_name* must not be a nested netlist property expression; if it is, the net expression is not evaluated.

As described in the next sections, the AMS netlister translates the inherited connections information into attributes in the Verilog-AMS netlist. The AMS elaborator uses these attributes to resolve inherited connections in the same way that other DFII tools do.

See the “[Inherited Connections](#)” section, in the “Understanding Connectivity and Naming Conventions” chapter of the *Virtuoso Schematic Editor User Guide* for detailed information about inherited connections.

## Inherited Signal Connections

In Virtuoso Schematic Editor, an inherited signal connection is created by associating a net expression with the wire that represents the signal. The net expression acts like a wire name; that is, in the same manner that a wire name specifies the name of the net, the default name in the net expression specifies the name of the underlying net.

The AMS netlister translates inherited connection net expressions into net attributes that become part of the Verilog-AMS netlist. The translation of the net expression in the schematic, which has the form

```
[@property_name:%:default_net_name]*
```

occurs in accordance with the following syntax:

```
inh_conn_net_exp_declaration ::=
    wire net_exp_attribute_list net_identifier;
net_exp_attribute_list ::=
    (* integer inh_conn_prop_name = property_name;
       integer inh_conn_def_value = default_net_name; *)
property_name ::=
    string
default_net_name ::=
    string
```

For example, the AMS netlister translates the CDBA 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! ;
```



## Inherited Terminal Connections

In Virtuoso Schematic Editor, an inherited terminal connection is created by associating a net expression with the pin that physically represents the terminal. The pin can exist in a schematic, a 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 port attributes are attached to port declarations rather than net declarations.

## Instance Values for Inherited Connections

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 branch of the hierarchy that is affected. The AMS netlister translates `netSet` properties into `cds_net_set` attributes, which are stored in the Verilog-AMS netlist.

Then, 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.

The CDBA `netSet` properties are represented, in Verilog-AMS netlists, as `cds_net_set` attributes. The syntax is:

```
cds_net_set_declaration ::=
    module_identifier net_set_attribute_list instance_identifier
    (port_list_connection);

net_set_attribute_list ::=
    (* integer cds_net_set[0:n] = { property_name {,property_name} };
    property_list *)

property_list ::=
    property_declaration; {property_declaration;}

property_declaration ::=
    simple_property
    |   inh_conn_expression

simple_property ::=
    property_name = property_value

inh_conn_expression ::=
    integer property_name[0:1] = { property_name, def_net_name }

n ::=
    positive_integer
```

## Virtuoso AMS Environment User Guide

### Netlisting

---

```
property_name ::=
    string
property_value ::=
    string
def_net_name ::=
    string
```

The `cds_net_set` attribute is an array of one or more elements that stores the names of the netSet properties. Each element in the array has a corresponding `property_name` attribute.

In most cases, the `property_value` is a string specifying the override connection. The `property_value` can also be an inherited connection expression. In this case, the `property_name` is represented as a two-element array, using the first element as the new property name and the second element as the new default connection name. For example, consider the following CDBA netSet properties and values.

Property	Value
vdd	3.3v!
xground	[@new_ground:%:gnd5!]

These 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 ) );
```

## Third-Party Tools and Other Cadence Tools

The attributes added to netlists by the AMS netlister are Cadence-specific extensions to the Verilog-AMS language. As a consequence, some third-party tools and some Cadence tools, such as Verilog-XL, interpret the attributes as illegal code so parsing fails. To make sure that the Verilog-AMS netlist that is generated by the AMS netlister can be read by other tools, select the *Netlister – Verilog-AMS* category of the AMS options and turn on *Conditionally include language extensions*. This option encloses the `cds_net_set` attributes in compiler directives so the attributes are used only when the netlist is compiled by the AMS netlister.

For example, with *Conditionally include language extensions* turned on, the previous `cds_net_set` attributes are written to the netlist as follows.

```
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. See [“Setting Verilog-AMS Options for the Netlister”](#) on page 79 for more information.

**Note:** Even though omitting attributes allows compilation to succeed, the simulation results might be incorrect without the connection information found in the attributes.

## How Aliased Signals Are Netlisted

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.

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 `cds_alias` module that the netlister instantiates is implemented as follows:

```
// Verilog HDL for "basic", "cds_alias" "functional"
module cds_alias(a,a);
    parameter width = 1;
    inout [width-1:0] a;
endmodule
```

AMS Designer instantiates this module as necessary in the `cds_globals` module or in cellview netlists created by the netlister. For example, to alias `signal2` and `signal3`, the netlister automatically places the following instantiation in the netlist.

```
cds_alias #(.width(1))
(* integer library_binding = "basic";
integer view_binding = "functional"; *)
ams_alias_inst_0 (signal2, signal3);
```

The `library_binding` and `view_binding` attributes are included by default to specify the binding for the `cds_alias` module. You can use `ams.env` variables to control whether the netlister writes these attributes. For more information, see [“bindCdsAliasLib”](#) on page 389 and [“bindCdsAliasView”](#) on page 390.

## How m-factors (Multiplicity Factors) Are Netlisted

An 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. For more information about the language attributes that AMS Designer provides to support m-factors, see the [“Using an m-factor \(Multiplicity Factor\)”](#) section of chapter 10, in the *Cadence Verilog-AMS Language Reference*.

For example, you have a symbol for the `pmos4` model and the symbol has the following CDF parameters. In this form, *Multiplier* is the prompt for the CDF parameter named `m`, which is the m-factor for this model.

CDF Parameter	Value
Model name	<code>pmos4</code>
Multiplier	<code>2</code>
Width	<code>20u M</code>
Length	<code>3u M</code>

The netlister creates a netlist that includes 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! );
```

Notice how the first line of the netlist fragment passes the value 2 for the parameter `m`. Also, notice in the fifth line, how the `passed_mfactor` attribute identifies the parameter `m` as an m-factor. A similar attribute appears in each instance statement that instantiates the `pmos4` model.

## How Iterated Instances Are Netlisted

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 specifies the number of logical copies to be created. 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. When the `amsScalarInstances` variable is set to `t`, as it is by default, the AMS netlister expands each iterated instance into multiple instances.

For example, you have the following top-level schematic called `iter_top`, which includes an instance of the cell `iter_master`. The name of the instance, `I<3:1>`, indicates that this is an iterated instance representing three logical instances.



As the AMS netlister processes this design, it expands the iterated instance, generating 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! ) ), // The iterated instance
    I_2 ( .a( cds_globals.\gnd! ) ), // is expanded in the netlist
    I_1 ( .a( cds_globals.\gnd! ) ); // when amsScalarInstances = t.
endmodule
```

Notice the `elaboration_binding` attribute in the netlist. When `amsScalarInstances` is set to `t`, the elaborator uses this attribute to determine what the master iterated instance is, and therefore what the instance binding is, for each of the logical copies. When `amsScalarInstances` is set to `nil`, the `elaboration_binding` attribute is unnecessary and does not appear in the netlist.

(For more information about the `amsScalarInstances` variable, see [“amsScalarInstances”](#) on page 386.)

## Passing Model Names as Parameters

This section describes how the AMS netlister handles the `modelName`, `model`, `modelName`, and `componentName` parameters. For additional information, see [“Special Handling of model, modelName, modelName, and componentName”](#) on page 597.

### Effect of the `modelName`, `model`, and `modelName` Parameters

The `modelName`, `model`, and `modelName` parameters (together referred to as `model*` parameters) allow you to pass a model name through the hierarchy. A `model*` parameter must be an AEL expression that has `parseAsCEL = t` and `parseAsNumber = 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, <code>model="foo"</code>	Evaluates it as a literal ( <code>foo</code> )
pPar expression, for example, <code>model="pPar("foo")"</code>	Evaluates it to be the parameter ( <code>foo</code> ), which must be declared as a parameter to the module being netlisted ( <code>parameter foo=defVal;</code> )  The netlister also instantiates an <code>analogmodel</code> construct.
iPar expression that resolves to a literal, for example, <code>model="iPar("foo")",</code> <code>foo="bar"</code>	Replaces the name of the cell on the instantiation to which the <code>modelName</code> parameter applies

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

Assume an instance, `i0`, of a `pmos` device to be bound to a model called `pmos395`.

If `i0` has the parameter `modelname="pmos395"` the resulting Verilog-AMS netlist contains the following instantiation:

```
pmos395 #(.l(3u), .w(9.5u)) (*integer library_binding = "analogLib";*)
      i0 ( .D( vref3 ), .G( vref1 ), .S( cds_globals.\vdd! ) );
```

The syntax used for the Verilog-AMS instance in this case is:

```
model_name #(prop_list) (* attr_list *) inst_name
      (port_connections);
```

### Example 2

Assume an instance, `i0`, of a `pmos` device to be bound to a model called `pmos395`.

If `i0` has the parameter `modelname=pPar("mod1")` and the default value for `mod1` in the `pmos` instance master CDF is "`pmos395`", the resulting Verilog-AMS netlist contains the following parameter declaration 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 syntax used for the Verilog-AMS in this case is:

```
analogmodel #(prop_list) (* attr_list *) inst_name (port_connections);
```

Note that in example 1, the name of the model is used. In example 2, the keyword `analogmodel` is used along with the parameter, `modelname`, which is set to the parameter specifying the name of the model.

### Example 3

Assume an instance, `i0`, of a `pmos` device to be bound to a model called `pmos395`.

A `modelname` parameter value that is an `iPar` expression results in the same behavior as a literal string. The AMS netlister finds the parameter referenced in the `iPar` expression on the instance and replaces the `iPar` expression with the value of the parameter. For example, given:

```
modelname=iPar("foo")
foo="pmos395"
```

## Virtuoso AMS Environment User Guide

### Netlisting

---

The AMS netlister resolves this to:

```
modelName="pmos395"
```

The resulting instantiation is:

```
pmos395 #(.l(3u), .w(9.5u)) (*integer library_binding = "analogLib";*)
      i0 ( .D( vref3 ), .G( vref1 ), .S( cds_globals.\vdd! ) );
```

The syntax used for the Verilog-AMS instance in this case is:

```
model_name #(prop_list) (* attr_list *) inst_name (port_connections);
```

This is the same as the Verilog-AMS produced in example 1.

## Handling of the model\* and componentName Parameters

The values of the `model*` and `componentName` parameters are considered to be literal strings, and are substituted for the cell name of the instances to which they apply. If the AMS netlister detects a `Par` expression in the value of any of the `model*` parameters, it uses the `analogmodel` construct as explained in the previous section. However the value of `componentName` must be a literal string.

When the values of the `model`, `modelName`, and `componentName` parameters are of type string, their values are substituted for the cell name of the instance to which they apply. This is the behavior illustrated in [“Example 1”](#) on page 135.

If, rather than simple string literals, the values of the `model*` parameters are AEL expressions like `pPar( "mod1" )`, the parameters are handled as described in [“Example 2”](#) on page 135. In this case, the AMS netlister generates an `analogmodel` construct using the parameter `modelName` rather than what is found in the CDBA data (`model` or `modelName`).

The `componentName` parameter is always handled as in [“Example 1”](#) on page 135. If the value of a `componentName` parameter is an AEL expression, the AMS netlister produces an error message and does not generate a netlist.

## Precedence of the model\* and componentName Parameters

When an instance has more than one `model*` or `componentName` parameter, the `modelName` parameter has the highest precedence, followed by `model`, `modelName`, and `componentName`.



## Specifying Parameters to be Excluded from Netlisting

This section describes how you can precisely specify parameters to be ignored during netlisting. Depending on your needs, you can specify such parameters for particular cells, for particular designs, for entire libraries, or for any combination of the three.

- To specify ignored parameters at the cell level, you use the `excludeParameters` field in the AMS `simInfo`. For more information, see [“excludeParameters”](#) on page 590.
- To specify ignored parameters for a design, you use the `amsExcludeParams` variable in the `ams.env` file. For more information, see [“amsExcludeParams”](#) on page 381.
- To specify ignored parameters at the library level, you use a CDF parameter called `amsExcludeParams` in the base library CDF. For more information, see the following section, [“Ignoring Parameters for Entire Libraries.”](#)

### Ignoring Parameters for Entire Libraries

To specify ignored parameters at the library level, you use the following steps to add a CDF parameter called `amsExcludeParams` to the base library CDF.

1. From the CIW, choose *Tools – CDF – Edit* to open the Edit Component CDF form.
2. In the Edit Component CDF form, select *Library* as the *CDF Selection* and *Base* as the *CDF Type*.
3. Type in the name of the library for which parameters are to be ignored.

After these steps, the Edit Component CDF form looks like this.

4. Click *Add*.

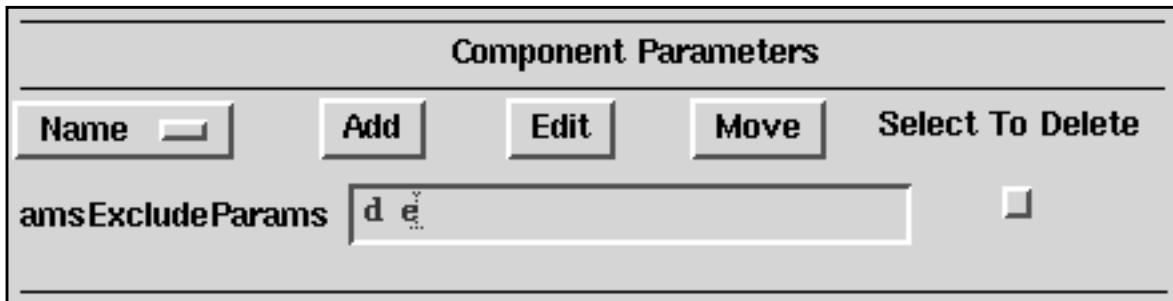
The Add CDF Parameter form appears.

5. In the Add CDF Parameter form, choose *string* for *paramType*, and type in *amsExcludeParams* for *name*, and *yes* for *editable*. In the *defValue* field, type in the parameters to be ignored.

The screenshot shows the 'Add CDF Parameter' dialog box. The 'Add After Parameter' tab is selected. The 'paramType' is set to 'string'. The 'name' field contains 'amsExcludeParams'. The 'defValue' field contains 'd e'. The 'editable' field is set to 'yes'. The 'parseAsNumber', 'parseAsCEL', and 'storeDefault' fields are all set to 'don't use'. The 'prompt', 'use', 'display', and 'dontSave' fields are empty. The 'OK' button is highlighted.

6. Click *OK* to close the Add CDF Parameter form.

7. The *Component Parameters* pane in the Edit Component CDF form changes to look like this, indicating that the parameters are to be ignored whenever they are found associated with a component in this library.



8. Click *OK* to close the Edit Component CDF form.

### Example: Specifying Parameters to Ignore

The effects of the three methods of specifying parameters to exclude are additive. For example, given

- In the `simInfo` field for the cell `cell1`, the *excludeParameters* field set to `f`
- In the `ams.env` file for the design, `amsDirect amsExcludeParams string "a b c"`
- In the base library-level CDF of the library `mylib`, `amsExcludeParams` set to `d e`

the final list of parameters excluded while processing `mylib.cell1:schematic` is `a, b, c, d, e, and f`.

Assuming that no parameters are excluded in the *excludeParameters* `simInfo` field for `mylib.cell2`, the excluded list for that cell is `a, b, c, d, and e`.

Assuming that no parameters are excluded in the *excludeParameters* `simInfo` field for `cell3` and that the `mylib2` library does not have any `amsExcludeParams` parameters set, the excluded list for `mylib2.cell3` is just `a, b, and c`.

## Preparing to Netlist User-Defined Functions

By default, the netlister returns an error when it encounters a call to a Spectre or SPICE user-defined function (UDF) in a schematic because AMS Designer does not support this usage. However, at your request, these UDF calls can be converted into macro references, which are

## Virtuoso AMS Environment User Guide

### Netlisting

---

supported by AMS Designer. If you take this approach, you must ensure that the referenced macros are actually 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 the UDF calls into macro references. For more information, see [“netlistUDFAsMacro”](#) on page 456.

2. Create a file containing the definitions for the macros.

Use formats such as the following, beginning each definition with ``define` and enclosing the actual function in parentheses.

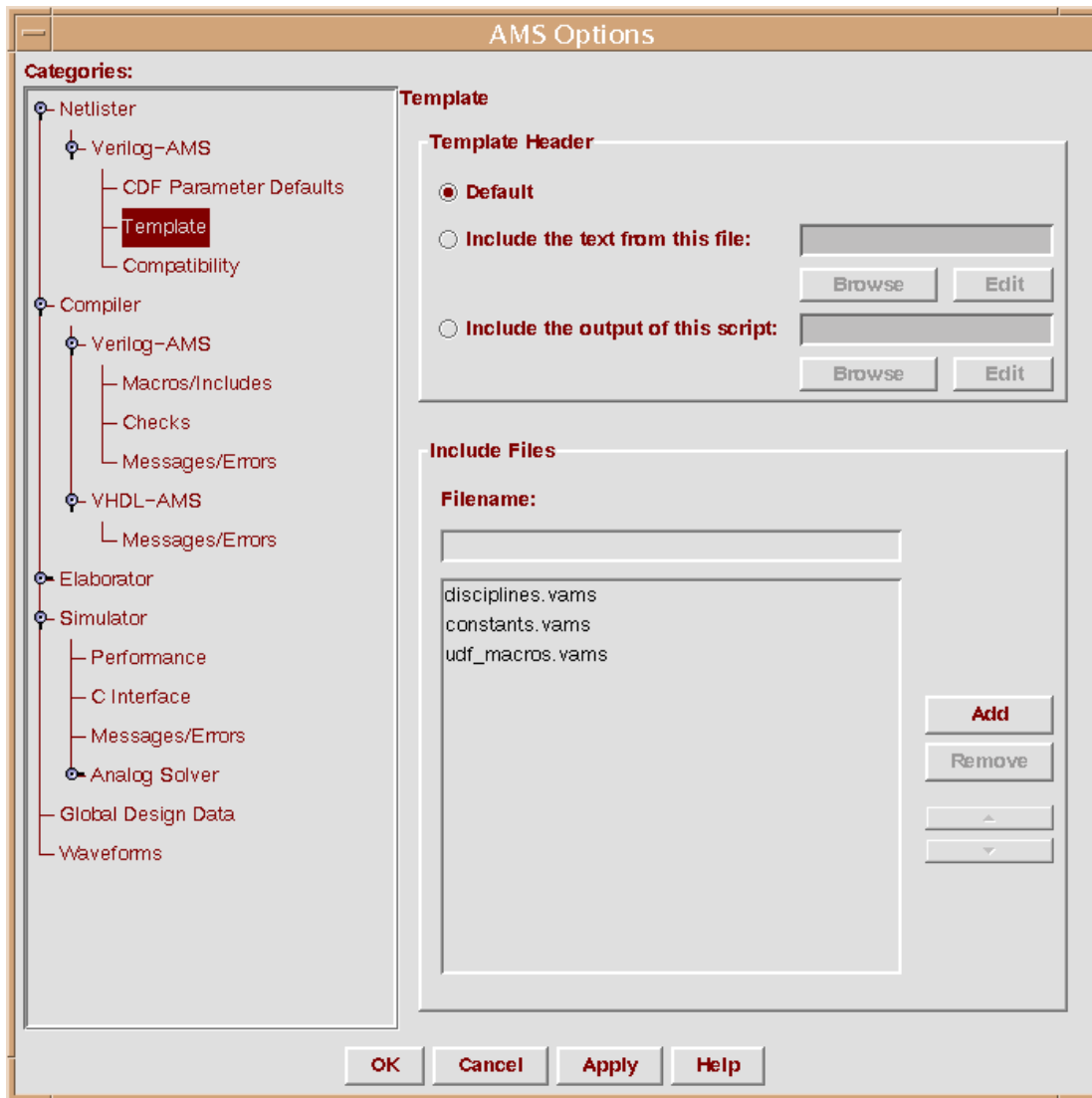
```
`define mod(a,b) ( (a) - (b)*ceil(((a)+0.5)/(b)) )  
`define int(a) ( (abs(a)==(a)) ? floor(a) : ceil(a) )
```

3. Add the new definitions file to the list of files included in netlists.

## Virtuoso AMS Environment User Guide

### Netlisting

For guidance, see [“Setting Template Header and File Include Options”](#) on page 81. For example, if your definitions file is named `udf_macros.vams`, you might set up the *Template* pane like this to include the file in generated netlists.



4. Ensure that the definitions file is found by using the *Include Directory* table accessed by choosing *Compiler – Verilog-AMS – Macros/Includes* in the AMS Options *Categories* pane. For guidance, see [“Specifying Macros and Specifying Directories to be Searched”](#) on page 104.

## Ensuring that Floating Point Parameters Netlist Correctly

This section describes how to specify parameters for a cell so that floating point values appear as intended in netlists. The procedure described here prevents problems that can arise when whole number floating point values are simplified to integer values during netlisting. The solution is to use cell CDF parameters, overriding the values for specific instances by using object properties.

### Defining the Parameter and Specifying a Default Value

You can use the following steps to define such a parameter and specify the default value for it.

1. In the CIW, choose *Tools – CDF – Edit*.

The Edit Component CDF form appears.

2. Type in the *Library Name* and *Cell Name* of the component that specifies the parameter of interest.
3. Ensure that *Base* is selected in the *CDF Type* field.
4. In the Component Parameters section of the form, click *Add*.

The Add CDF Parameter form appears.

5. Select *paramType* string, *parseAsNumber* yes, and *parseAsCEL* yes and type the name of the parameter into the *name* field. For example, the form might look like this for a parameter named `hiVolt`.

The screenshot shows a dialog box titled "Add CDF Parameter". At the top are buttons for "OK", "Cancel", "Apply", and "Help". Below these is a section with "Add After Parameter" and a dropdown menu currently showing "As First Parameter". The main area contains several labeled fields, each with a dropdown menu: "paramType" is set to "string", "parseAsNumber" is set to "yes", "units" is set to "don't use", "parseAsCEL" is set to "yes", and "storeDefault" is set to "don't use". Below these is a text field labeled "name" containing the text "hiVolt". At the bottom is a text field labeled "prompt" which is currently empty.

6. Click **OK**, to close the Add CDF Parameter form.

The new component parameter appears in the Edit Component CDF form.

7. Type the default parameter value into the field labeled with the name of your parameter. Use the number format that you want to appear in the netlist. For example, to be compatible with text views coded in VHDL-AMS, which is a strongly typed language, you might need to enter a whole number as a number with a decimal point followed by zero.

Unless overridden on the instance (as described in [“Overriding the Default Parameter for Specific Instances”](#) on page 145), this is the value of the parameter for this cell of the



design whenever it is instantiated. For example, the form might look like this for the CDF parameters `hiVolt` and `refVolt`.

**Edit Component CDF**

OK Cancel Apply Help

CDF Selection ☒ Cell ☐ Library CDF Type Ba

Library Name

Cell Name  Browse

File Name

Load Save File Name Select Char

**Component Parameters**

Name	Add	Edit	Move	Select
hiVolt				
refVolt				

8. Click **OK** to close the form.

## Overriding the Default Parameter for Specific Instances

Specifying a component parameter in the CDF as described in the previous section provides a default value for that parameter. If you need to specify a different parameter value for a specific instance, you can follow these steps.

## Virtuoso AMS Environment User Guide

### Netlisting

1. Select the instance in the schematic that is to use a non-default parameter value.
2. Open the Edit Object Properties form for the instance.
3. Change the value for the CDF parameter to the value you need for the instance.

**Edit Object Properties**

OK Cancel Apply Defaults Previous Next Help

Apply To

Show ☐ system ☐ user ☒ CDF

Property	Value	Dis
Library Name	<input type="text" value="amslih"/>	<input type="button" value="off"/>
Cell Name	<input type="text" value="daconv"/>	<input type="button" value="valu"/>
View Name	<input type="text" value="symbol"/>	<input type="button" value="off"/>
Instance Name	<input type="text" value="I4"/>	<input type="button" value="off"/>

CDF Parameter	Value	Dis
hiVolt	<input type="text" value="11.0"/>	<input type="button" value="off"/>
refVolt	<input type="text" value="6.0"/>	<input type="button" value="off"/>

4. Click OK.

Now when you netlist the schematic, the parameter appears in the netlist with the non-default value. Parameters that use the default value do not appear in the netlist. For example, with the Edit Component CDF and Edit Object Properties forms set up as in the previous illustrations, `hiVolt` appears in the netlist but `refVolt` does not.

## Virtuoso AMS Environment User Guide

### Netlisting

---

```
daconv #(.hiVolt(11.0)) (* integer library_binding = "amslib"; *) I4  
( .b6( b6 ), .b5( b5 ), .b0( b0 ), .compSig( dacOut ), .b2( b2 ), .b1(  
b1 ), .b3( b3 ), .b4( b4 ), .b7( b7 ) );
```

## **Virtuoso AMS Environment User Guide**

### **Netlisting**

---

---

## Working with Schematic Designs

---

One of the AMS Designer flows allows you to create a Verilog<sup>®</sup>-AMS netlist for each cellview you save using the Virtuoso<sup>®</sup> 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. This chapter briefly reviews some of the features of the schematic editor: for complete information, see the *[Virtuoso Schematic Editor User Guide](#)*.

This chapter contains the following sections:

- [Setting Schematic Rules Checker Options for AMS Designer](#) on page 150
- [Creating Cellviews Using the AMS Environment](#) on page 152
- [Inherited Connections](#) on page 167
- [Net and Pin Properties](#) on page 172

## Setting Schematic Rules Checker Options for AMS Designer

You use the Setup Schematic Rules Checks form to specify the checking that occurs when you check and save a schematic. For AMS Designer, the checks can give you feedback regarding CDBA-to-Verilog-AMS translation without having to generate a netlist. By default, this feature is not enabled.

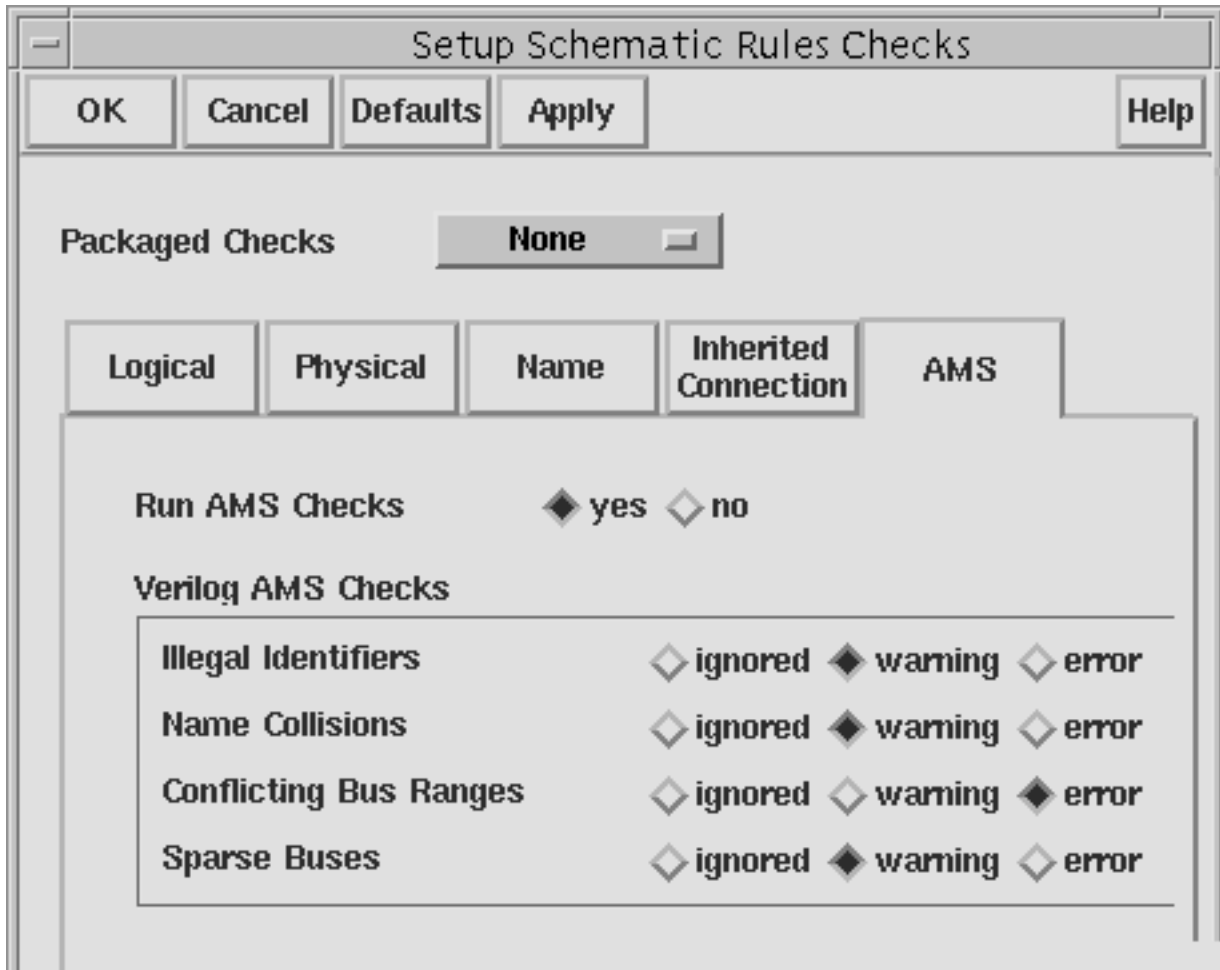
After you set these rules and enable the checks, they are run by any of the following commands: *Design – Check and Save*, *Check – Current Cellview*, *Check – Hierarchy*.

To set up the Schematic Rules Checker (SRC) rules,

1. From the Virtuoso Schematic Editing window, choose *Options – Check Rules Setup*.

The Setup Schematic Rules Checks form appears.

2. Click *AMS* to open the pane for the AMS Designer checks.



3. To enable the AMS checks, select *yes* for the *Run AMS Checks* field.
4. If necessary, change the severity levels for the *Verilog AMS Checks*.

For more information about these checks, see the following references.

Check	Reference
Illegal identifiers	<a href="#">“allowIllegalIdentifiers”</a> on page 370
Name collisions	<a href="#">“allowNameCollisions”</a> on page 372
Conflicting bus ranges	<a href="#">“allowDeviantBuses”</a> on page 368
Sparse buses	<a href="#">“allowSparseBuses”</a> on page 374

## Creating Cellviews Using the AMS Environment

This section describes how to create symbol, block, and Verilog-AMS cellviews in the AMS environment.

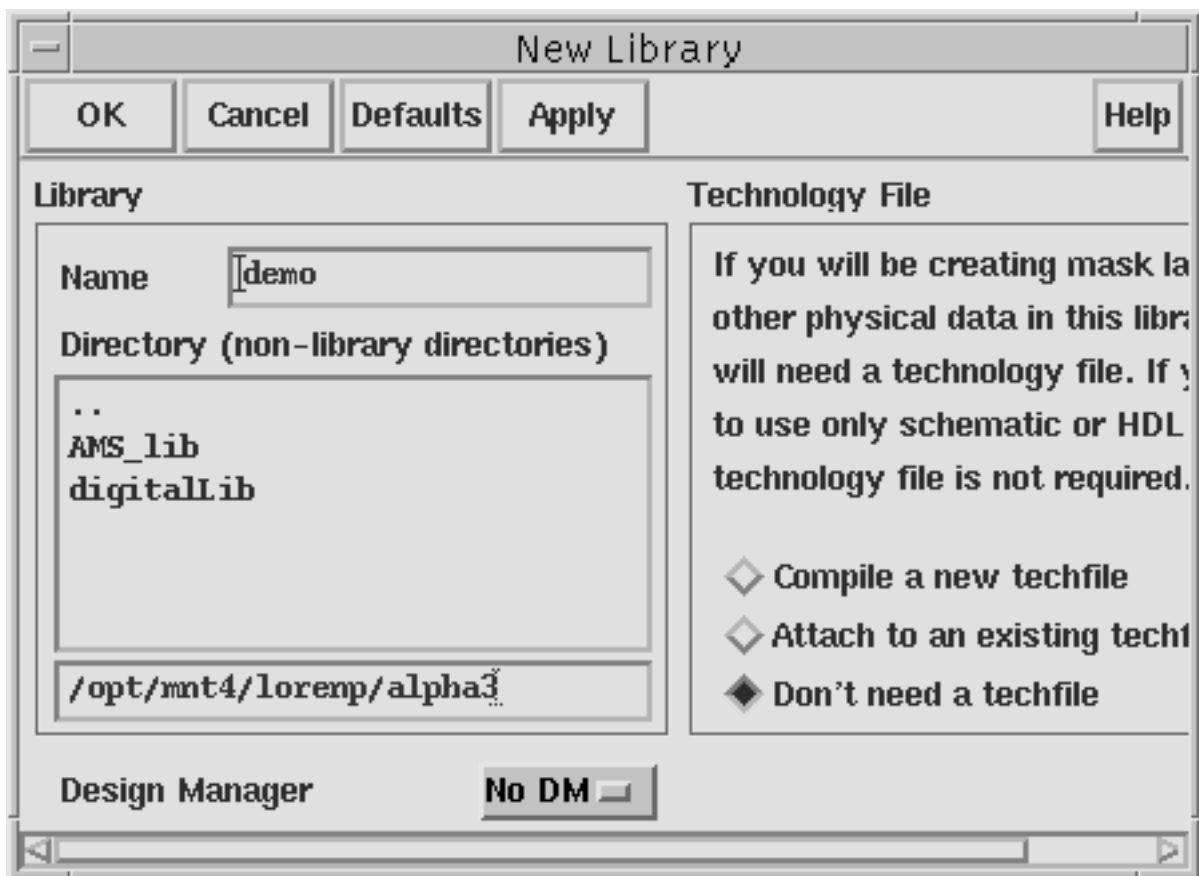
### Preparing a Library

Before you create a cell, you must have a library in which to place it. You can create and store Verilog-AMS components in any Cadence component library. You can create a new library or use one that already exists.

To create a new library,

1. From the Command Interpreter Window (CIW), choose *File – New – Library*.

The New Library form appears.



2. Type the new library name and directory and click on the radio button for *Don't need a techfile*. Click *OK*.



You can also use the Cadence library manager to create a new library.

1. From the CIW, choose *Tools – Library Manager*.

The library manager appears.

2. Choose *File – New – Library*.

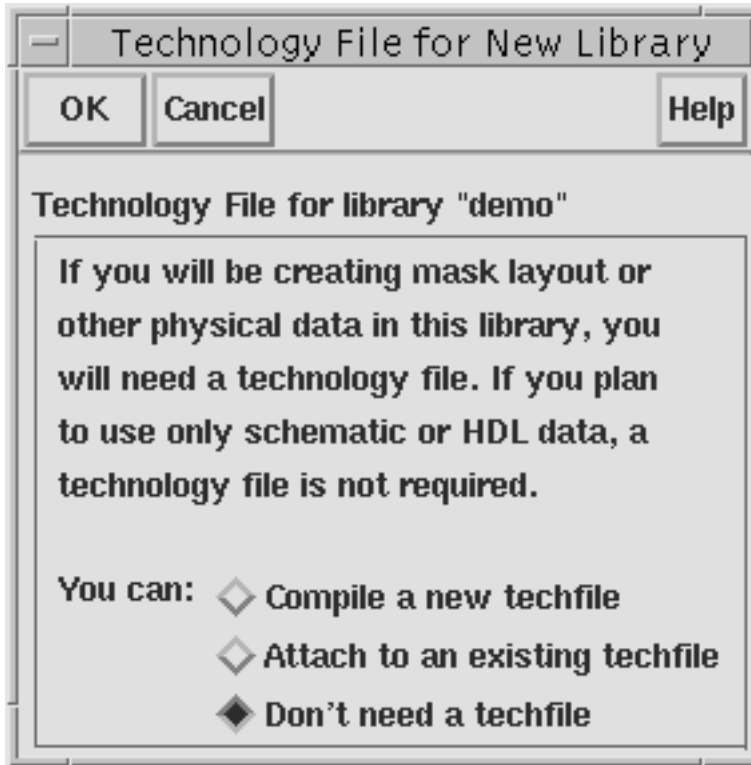
The New Library form appears. This form is different from the New Library form that you can open from the CIW.

The screenshot shows the 'New Library' dialog box. It has a title bar with a minus sign and the text 'New Library'. The dialog is divided into three main sections. The first section, titled 'Library', contains a 'Name' text field with the text 'demo' and a 'Directory' list box. The list box contains the following items: '..', 'AMS\_lib', 'digitalLib', 'sch. tran', 'schematic.shm', 'tran1. tran', and 'waves. shm'. Below the list box is a text field containing the path '/opt/mnt4/lorenp/alpha3'. The second section, titled 'Design Manager', contains two radio buttons: 'Use NONE' and 'Use No DM'. The third section at the bottom contains four buttons: 'OK', 'Apply', 'Cancel', and 'Help'.

3. In the *Name* field, type the new library name.
4. In the *Directory* list box, choose the directory where you want to place the library.

**5.** Click *OK*.

A second form appears, asking if you need a technology file for this library.



**6.** Set *Don't need a techfile* on and click *OK*.

The AMS environment creates a new library with the name you specify in the directory you specify.

## Creating the Symbol View

To include a Verilog-AMS module in a schematic, you must create a symbol to represent the function described by the module. There are four ways to create this symbol:

- Choose *File – New – Cellview* from the CIW and specify the target tool as *Composer-Symbol*.
- Copy an existing symbol using the *Copy* command in the library manager. Look in `analogLib` for good examples to copy.
- Create a new symbol from another view using *Design – Create Cellview – From Pin List* or *Design – Create Cellview – From Cellview* in the Schematic Design Editor. To create a new symbol this way, you must first have an existing view with defined input and output pins.
- Use a block to represent a Verilog-AMS component, as described in [“Using Blocks”](#) on page 155.

However you create the symbol, it must reside in an existing library as described in [“Preparing a Library”](#) on page 152.

## Pin Direction

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.

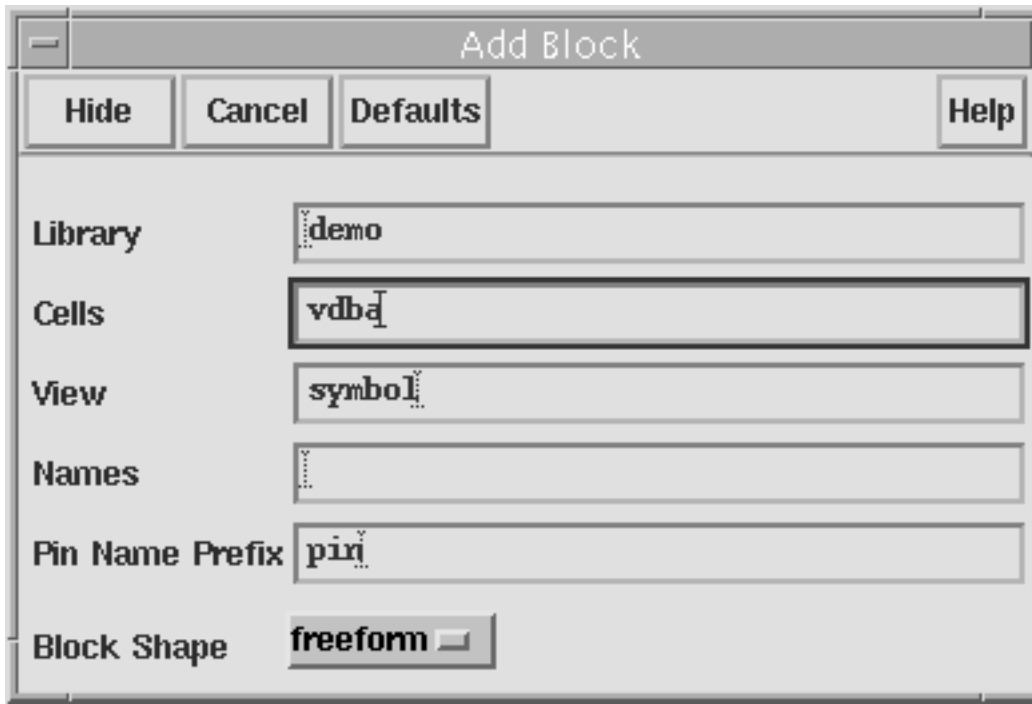
## Using Blocks

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.

In a schematic, to create a block and wire it, follow these steps:

1. Choose *Add – Block* in the Virtuoso Schematic Editing window.

The Add Block form appears.



The screenshot shows the 'Add Block' dialog box. The title bar is 'Add Block'. Below the title bar are four buttons: 'Hide', 'Cancel', 'Defaults', and 'Help'. The main area of the dialog contains several labeled input fields: 'Library' with the text 'demo', 'Cells' with the text 'vdba', 'View' with the text 'symbol', 'Names' (empty), 'Pin Name Prefix' with the text 'pin', and 'Block Shape' with a dropdown menu showing 'freeform'.

2. Type a library name, cell name, and view name.

Specify a cell and view combination that does not exist in that library. You can have schematic, analog HDL, or Verilog-AMS views for that cell, but you cannot already have a symbol view. The default library name is the current library, and the default view name is `symbol`.

3. (Optional) 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.

4. Set the *Block Shape* cyclic field.

5. Place the block as described in the following table.

<b>If Block Shape is set to freeform</b>	<b>If Block Shape is set to anything else</b>
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.	Drag the predefined block to the location where you want to place it and click.  Refer to the <u><i>Virtuoso Schematic Editor User Guide</i></u> for details about modifying the block samples using the <code>schBlockTemplate</code> variable in the <code>schConfig.il</code> file.

As you place each block, the schematic editor labels it with an instance name. If you leave the *Names* field of the Add Block form empty, the editor generates unique new names for the blocks.

The editor automatically creates a symbol view for the block.

6. Choose *Add – Wire (narrow)* or *Add – Wire (wide)* from the Virtuoso Schematic Editing window menu. When you connect the wire, the pin is created automatically. (To delete such a pin, you must use *Design – Hierarchy – Descend Edit* to descend into the block symbol.)

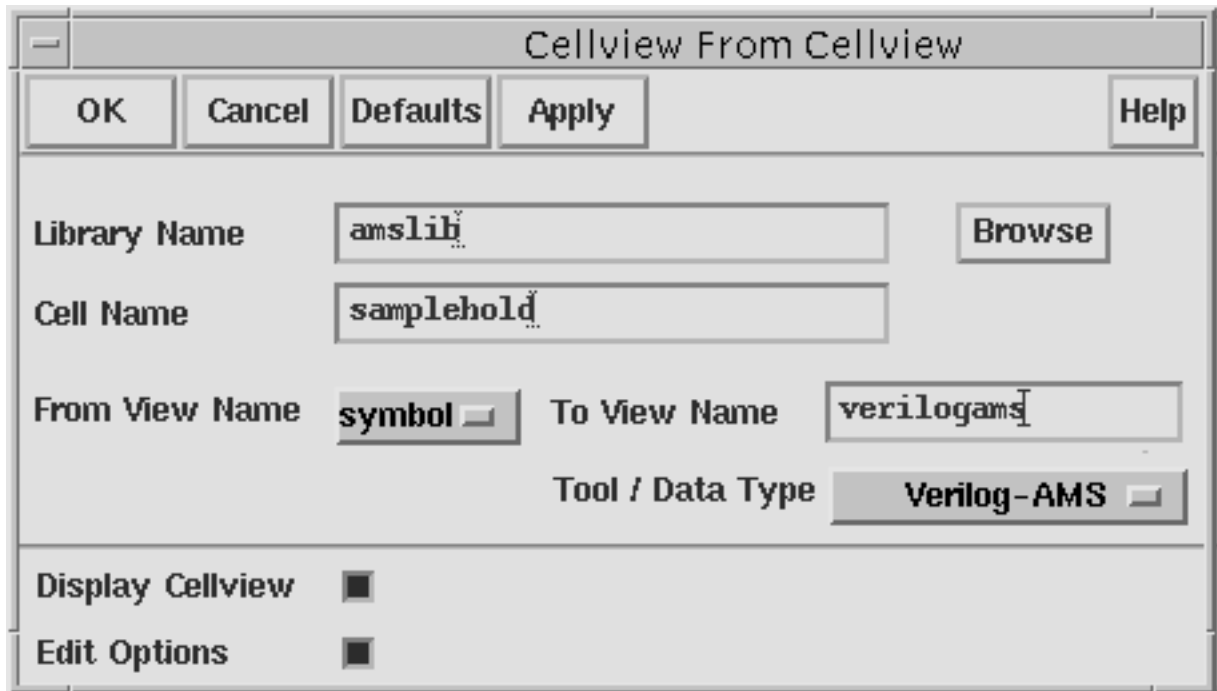
The *Pin Name Prefix* field on the Add Block form specifies the name for the automatically created pin.

## Creating a Verilog-AMS or VHDL-AMS Cellview from a Symbol or Block

Once you have an existing symbol or block, you can create a Verilog-AMS or VHDL-AMS cellview for the function identified by that symbol or block. To create the cellview, follow these steps:

1. Open the Symbol Editor in one of two ways:
  - ☐ From the CIW, choose *File – Open* and specify the component or block symbol.
  - ☐ From the library manager, choose *File – Open* or double-click on the symbol view.
2. From the Symbol Editor window, choose *Design – Create Cellview – From Cellview*.

The Cellview From Cellview form appears.



3. In the *From View Name* cyclic field, choose *symbol*.
4. in the *Tool / Data Type* cyclic list, choose either *Verilog-AMS* or *VHDLAMS*, according to the kind of view you want to create.
  - ❑ If you choose *Verilog-AMS*,
    - a. In the *To View Name* field, type a name such as `verilogams`. To comply with AMS Designer guidelines, the name should be all lower-case.
    - b. Click *OK*.

An active text editor window opens, showing the template for a Verilog-AMS module.

```
//Verilog-AMS HDL for "amslib", "samplehold" "verilogams"
`include "constants.vams"
`include "disciplines.vams"
module samplehold ( holdSig, inSig, trigger );
    input inSig;
    input trigger;
    output holdSig;
endmodule
```

The AMS environment creates the first few lines of the module based on the symbol information. Pin and parameter information is included automatically.

- ❑ If you choose *VHDLAMS*, and want to create an architecture,
  - a. In the *To View Name* field, type a name such as `samplehold_behav`. To comply with AMS Designer guidelines, the name should be all lower-case.
  - b. Click *OK*.

An active text editor window opens, showing the template for a VHDL-AMS architecture.

```
-- Create Architecture:
-- Library=amslib,Cell=samplehold,View=samplehold_behav
-- Time:Apr 17 09:32:36 2003
-- By:lorenp

ARCHITECTURE samplehold_behav of samplehold IS
BEGIN

END samplehold_behav;
```

The AMS environment creates the first few lines of the module based on the symbol information.

- ❑ If you choose *VHDLAMS*, and want to create an entity,
  - a. Use the default name of `vhdlams` in the *To View Name* field.
  - b. Click *OK*.

An active text editor window opens, showing the template for a VHDL-AMS entity.

```
library ieee, std;
use ieee.std_logic_1164.all;
use ieee.electrical_systems.all;
entity samplehold is
port ( terminal trigger : electrical;
      terminal \inSig : electrical;
      terminal \holdSig : electrical );
end samplehold;
```

The AMS environment creates the first few lines of the module based on the symbol information.

**5. Finish coding the module, then save the file and quit the text editor window.**

The AMS environment does not create the cellview until you exit from the editor.

Here is an example of a completed Verilog-AMS module:

```
//Verilog-AMS HDL for "amslib", "samplehold" "verilogams"
`include "constants.vams"
`include "disciplines.vams"

module samplehold ( holdSig, inSig, trigger );
```

## Virtuoso AMS Environment User Guide

### Working with Schematic Designs

---

```
input inSig;
input trigger;
output holdSig;

electrical gnd, inSig, holdSig;
real value;

analog begin
  // A digital event to which analog is made sensitive
  @(posedge(trigger))
    value = V(inSig);

  V(holdSig) <+ transition( value, 1n, 2.5n);
end
endmodule
```

When you save the module and quit the text editor window, the AMS environment checks the syntax in the text file. If the syntax checker finds any errors or problems, a dialog box appears with a message like the following:

```
Parsing of verilog-ams file failed:
Do you want to view the error file and re-edit HDL?
```

Click **Yes** to display the HDL Parser Error/Warnings window and to reopen the module file for editing.

### Editing Verilog-AMS and VHDL-AMS Cellviews Outside of the AMS Environment

You can create Verilog (digital), Verilog-AMS, VHDL (digital) and VHDL-AMS source files either inside or outside of the AMS environment. However, if you use source files created outside of the AMS environment you give up the automatic cross-checking among views that the environment performs. For example, if, outside the environment, you edit the port list of a text view, you must remember to update the corresponding symbol view. If you make a similar change within the environment, AMS Designer automatically prompts you to update the symbol.

If you do create HDL views outside of the environment, ensure that they are compiled by choosing the AMS Design Prep compile all option. For example, from the Command Interpreter Window (CIW) choose *AMS – Design Prep* to open the AMS Design Prep form. In that form, select *Compile* and *All*, and then click *Run*.

### Descend Edit

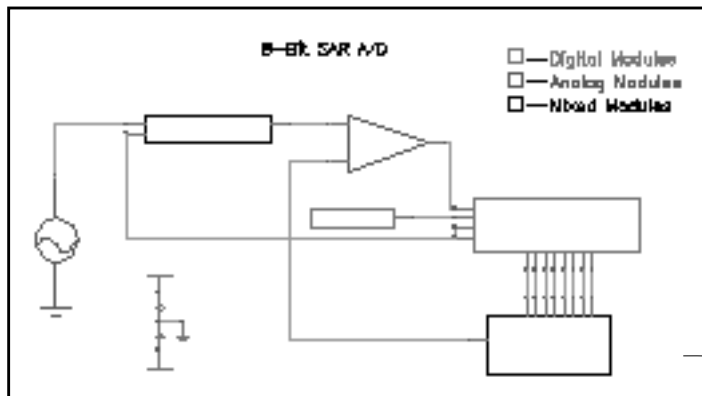
To examine the views below the symbols while viewing a schematic, choose *Design – Hierarchy – Descend Edit*. For example, there might be three view choices: *symbol*,



## Virtuoso AMS Environment User Guide

### Working with Schematic Designs

*verilogAMS*, and *daconv\_behav*. If you choose *verilogAMS*, a text window appears, as shown in the following figure.



```
/opt/mnt4/lorenp/alpha3/AMS_lib/amsLib/samplehc
[✓] Verilog-AMS HDL for "amslib", "samplehold" "
verilogAMS"

module samplehold ( inSig, trigger, holdSig );

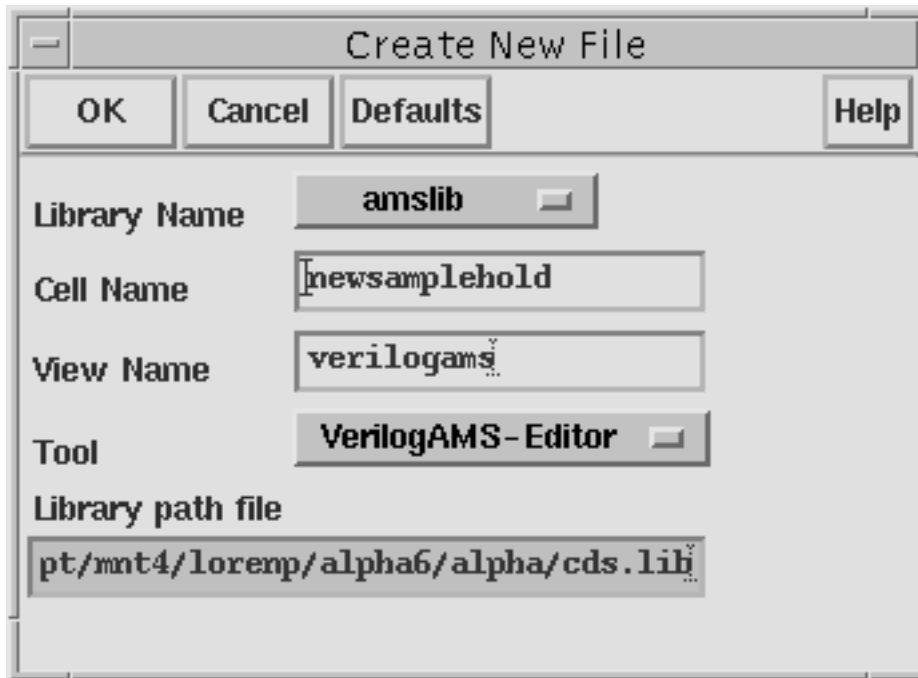
    input inSig;
    input trigger;
    output holdSig;
```

### Creating a Verilog-AMS Cellview

To create a new component with only a Verilog-AMS cellview,

1. From the CIW, choose *File – New – Cellview*.

The Create New File form appears.



2. Specify the *Cell Name* (component).
3. Set the *Tool* cyclic field to *VerilogAMS-Editor*.
4. In the *View Name* field, type the name for the new cellview. To comply with AMS guidelines, the name should be all lower-case.
5. Click *OK*.

## Virtuoso AMS Environment User Guide

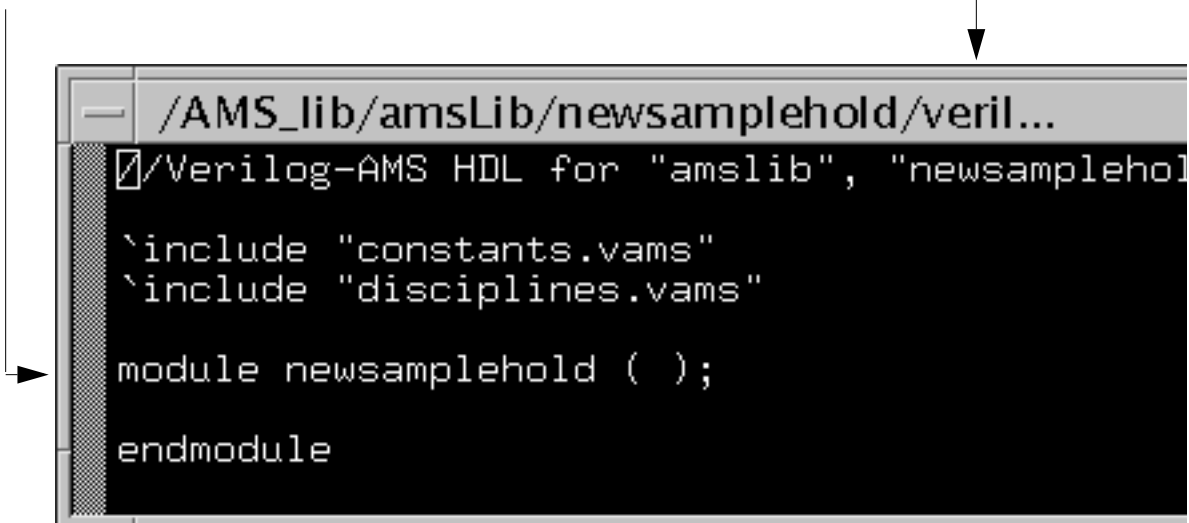
### Working with Schematic Designs

---

A text editor window appears for the new module. If the cell name you typed in the *Cell Name* field is new, an empty template appears. If the name you typed already has available views, a template appears with pin and parameter information in place.

The module must have the same name as the cell.

The UNIX file directory has the same name as the view name.



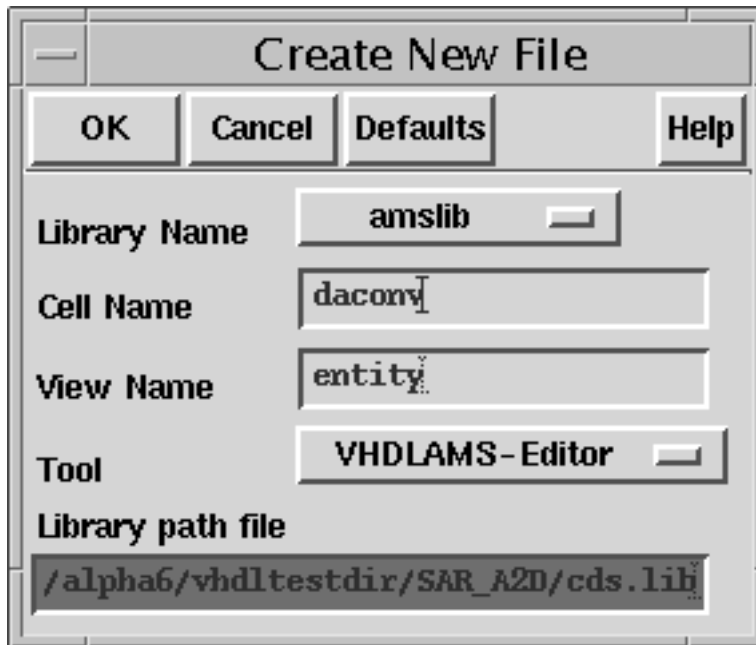
6. Modify any existing pin or parameter information as necessary. You can add unique or shared parameters as required by your design.
7. Complete the module. Be sure that the module name remains the same as the name of the cell.
8. Save the module, and quit the text editor window.

## Creating a VHDL-AMS Cellview

To create a new component with only a VHDL-AMS cellview,

1. From the CIW, choose *File – New – Cellview*.

The Create New File form appears.



2. Specify the *Cell Name* (component).
3. Set the *Tool* cyclic field to *VHDLAMS-Editor*.
4. In the *View Name* field, type the name for the new cellview. To comply with AMS guidelines, the name should be all lower-case. If the view name you enter is *entity*, the created view is a VHDL-AMS entity. If the view name you enter is anything other than *entity*, the created view is an architecture.
5. Click *OK*.

A text editor window appears for the new module. If the cell name you typed in the *Cell Name* field is new, an empty template appears. If the name you typed already has

available views and you are creating an entity view, a template appears with pin and parameter information in place.

The entity has the same name as the cell.

The UNIX file directory has the same name as the view name.

```
library ieee, std;
use ieee.std_logic_1164.all;
entity daconv is
port ( \compSig\ : out ;
      b0 : in ;
      b1 : in ;
      b2 : in ;
      b3 : in ;
      b4 : in ;
      b5 : in ;
      b6 : in ;
      b7 : in );
end daconv;
```

6. Modify any existing pin or parameter information as necessary. You can add unique or shared parameters as required by your design.
7. Complete the module. Be sure that the architecture or entity name remains the same as the name of the cell.
8. Save the module, and quit the text editor window.

## Creating a Symbol Cellview from a Verilog-AMS Cellview

If you created a Verilog-AMS cellview without creating a symbol, or if you have a component with only a Verilog-AMS cellview, you can add a symbol view by following these steps:

1. Choose *File – Open* from the CIW.

The Open File form appears.

2. Open any schematic or symbol cellview.

The editor appears.

**3.** Choose *Design – Create Cellview – From Cellview*.

The Cellview From Cellview form appears.

The screenshot shows the 'Cellview From Cellview' dialog box. The title bar reads 'Cellview From Cellview'. Below the title bar are buttons for 'OK', 'Cancel', 'Defaults', 'Apply', and 'Help'. The main area contains the following fields and controls:

- Library Name:** A text field containing 'amslih' and a 'Browse...' button to its right.
- Cell Name:** A text field containing 'newsamplehold'.
- From View Name:** A dropdown menu currently showing 'verilogAMS'.
- To View Name:** A text field containing 'symbol'.
- Tool / Data Type:** A dropdown menu currently showing 'Composer-Symbol'.
- Display Cellview:** A checked checkbox.
- Edit Options:** A checked checkbox.

**4.** Fill in the *Library Name* and *Cell Name* fields.

If you do not know this information, click *Browse*, which opens the Library Browser, so you can browse available libraries and components.

**5.** In the *From View Name* cyclic field, select the Verilog-AMS view.

**6.** In the *Tool / Data Type* cyclic field, choose *Composer-Symbol*.

**7.** In the *To View Name* field, type `symbol`.

**8.** Click *OK*.

The Symbol Generation Options form appears.

**9.** Click *OK*.

A Symbol Editor window appears.

**10.** Edit the symbol, save it, and close the Symbol Editor window.

## Inherited Connections

The inherited connections solution allows you to selectively override global signals in designs originated in the Virtuoso schematic editor. This solution also allows you to create special global signals and override their names for selected branches of the design hierarchy. The inherited connection capability is not supported for either VHDL (digital) or VHDL-AMS design units.

With the inherited connections feature you can use

- Multiple power supplies in a design
- Overridable substrate connections
- Parameterized power and ground symbols

The inherited connections feature is recognized by all tools throughout the IC design flow. To learn about using inherited connections and net expressions with various Cadence tools in the design flow, refer to the *Inherited Connections Flow Guide*. For more information about using inherited connections in schematics, see the “[Inherited Connections](#)” section, of the “Understanding Connectivity and Naming Conventions” chapter in the *Virtuoso Schematic Editor User Guide*.

This section describes the following information about inherited connections:

- [Global Signals in the Schematic Editor](#) on page 167
- [Inherited Connections in a Hierarchy](#) on page 168
- [Defining Inherited Connections](#) on page 170
- [How Net Expressions Evaluate](#) on page 171

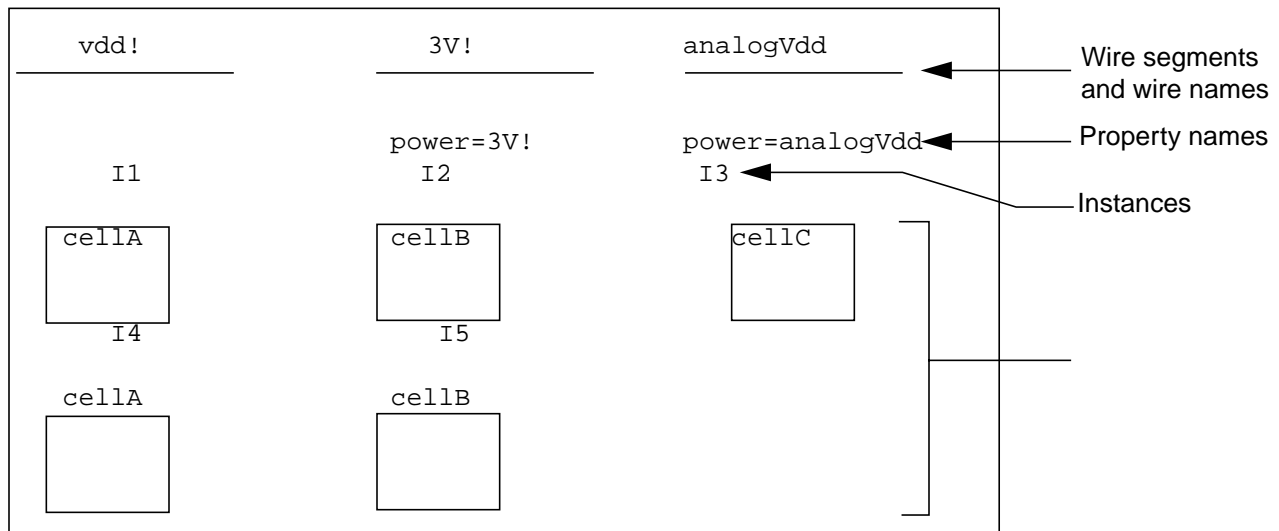
### Global Signals in the Schematic Editor

A global signal is a signal that is connected by name across all levels of a design hierarchy without using pins. In schematics, a global signal is determined by name; if the signal name ends with an exclamation point ( ! ), it is considered global. A signal that is explicitly passed everywhere in a design is not considered global. A global signal connects to other signals with the same name elsewhere in the hierarchy without requiring an explicit connection through the hierarchy.

## Inherited Connections in a Hierarchy

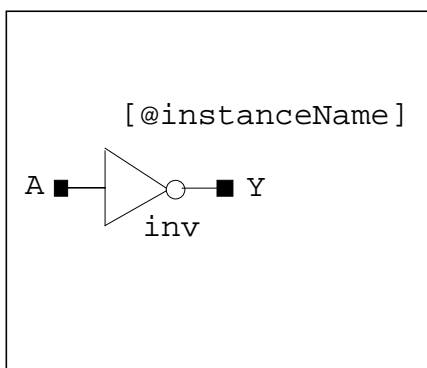
You use inherited connections to selectively override global signals within the Verilog (digital) and Verilog-AMS sections of your design. Inherited connections are not supported for either VHDL (digital) or VHDL-AMS design units. Consider the following example:

### High-Level Schematic

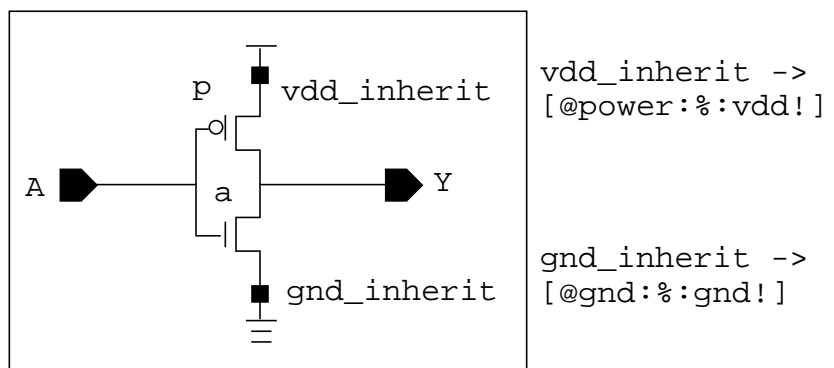


### Lower-Level Schematic

#### Inverter Symbol



#### Inverter Schematic



The high-level schematic shows five instances: `I1`, `I2`, `I3`, `I4`, and `I5`. Each instance represents a portion of the design, each of which eventually references the inverter shown below the high-level schematic.



## Virtuoso AMS Environment User Guide

### Working with Schematic Designs

---

In the inverter schematic, the power and ground wires are defined using the net expressions `[@power:%%:vdd!]` and `[@gnd:%%:gnd!]`, respectively. By default, all inverter `pmos` transistors are connected to `vdd!` and all `nmos` transistors are connected to `gnd!`.

In the high-level schematic, all the inverter `pmos` transistors below instance `I2` of `cellB` are to use `3V!` as the power supply, and the inverters in instance `I3` of `cellC` are to use `analogVdd`.

To selectively override the default global signal `vdd!`, you create a `netSet` type property on instance `I2` named `power` with the value `3V!` and a `netSet` type property on instance `I3` named `power` with the value `analogVdd`. Notice that the inverters below instances `I1`, `I4`, and `I5` are still connected to `vdd!`.

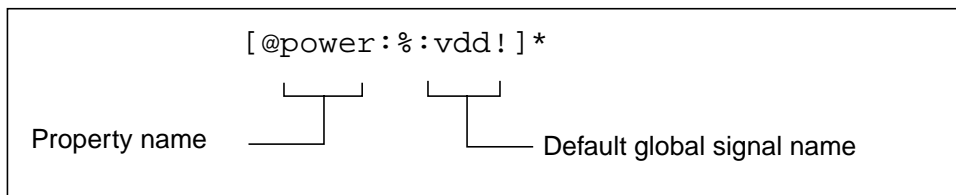
In other words, you place a `netSet` property on an instance representing the branch of hierarchy where it is to be applied. You can create the `netSet` property on any instances at any level above the cellviews with net expressions. For example, if a large hierarchical design has seven levels of hierarchy, you can place a `netSet` property `power = 3V!` on an `I2` instance in the top-level schematic. This affects all the logic below instance `I2` all the way to the bottom in all cellviews that contain a net expression, such as `[@power:%%:vdd!]`. All cellviews that contain this net expression use `3V!` instead of `vdd!` for that branch of the design.

However, if on a lower-level cellview there is an `I7` instance with a `netSet` property `power = 2V!`, then `2V!` is always used below the `I7` instance.

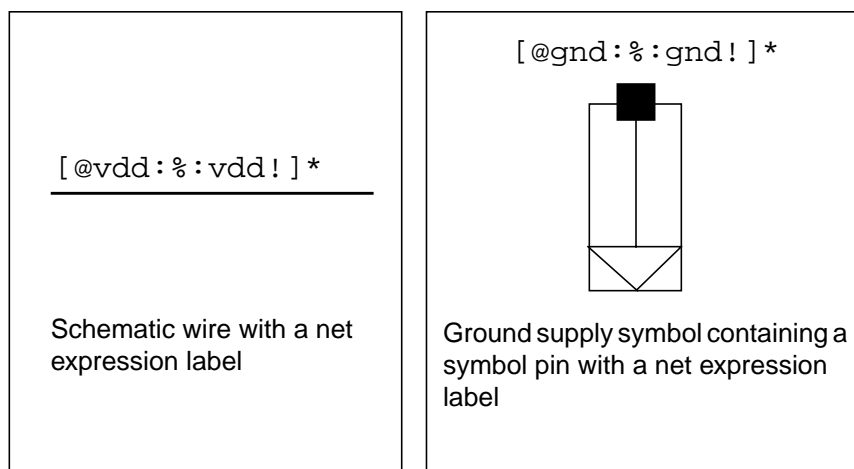
## Defining Inherited Connections

In the schematic editor, you define an inherited connection by adding a net expression label to either a wire or a pin. A net expression defines the default global signal name for the connection and the name of a property that can be used to override that default global signal name.

### Example Net Expression



### Example Net Expression Labels

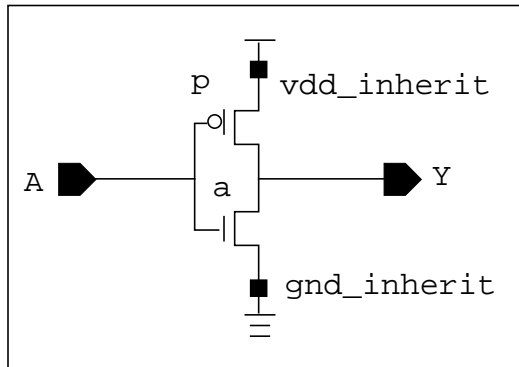


The asterisk ( \* ) after a global name shows that this is not a regular wire name but a name that is an overridable net expression. The default global signal name specifies what the wire or pin is connected to by default.

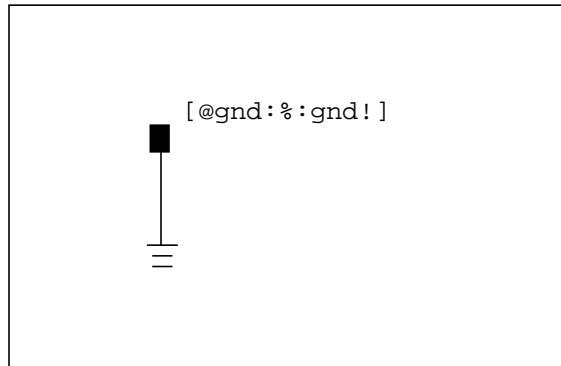
You can create an inherited connection in a schematic by placing an instance of a symbol where one of the symbol pins has a net expression label. When you run the checker program

on the schematic, the net expression label from the symbol pin propagates onto the net within the schematic.

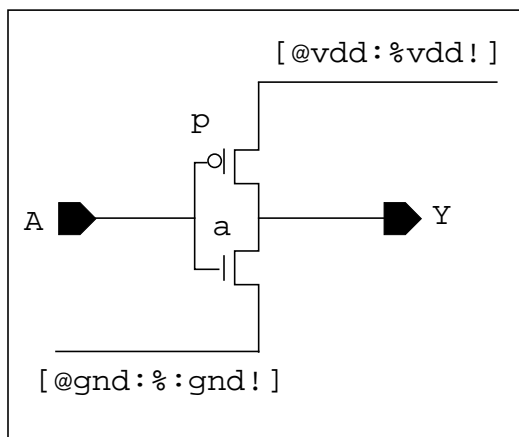
**Inverter Schematic**



**gnd\_inherit Symbol**



**Inverter Schematic**

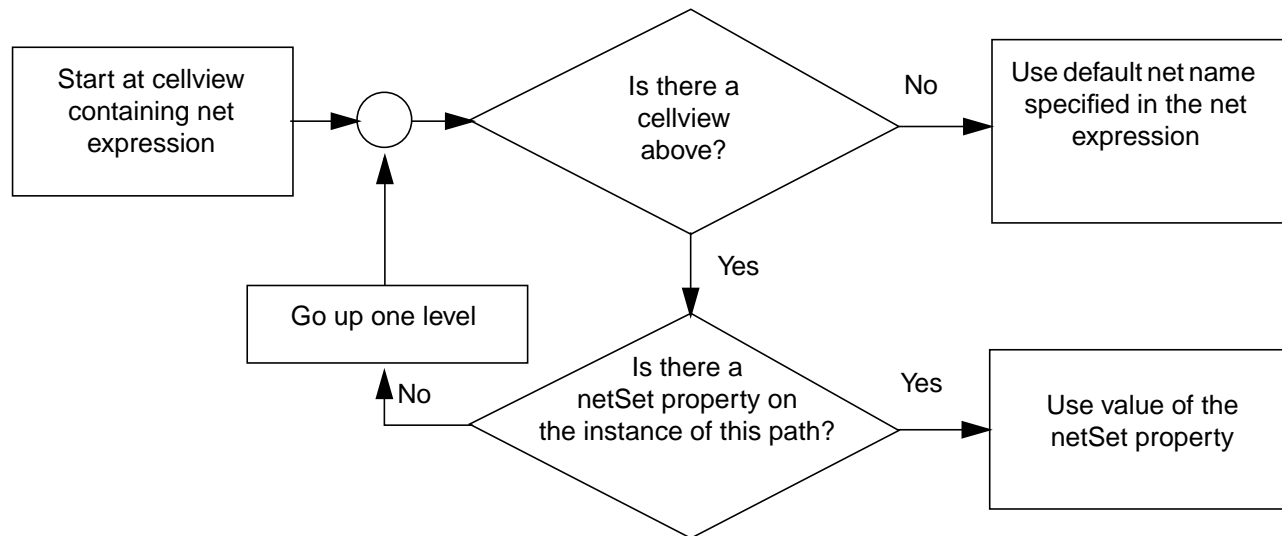


Net expression labels can also be placed on wires to achieve the same effect.

## How Net Expressions Evaluate

Net expressions are evaluated for each hierarchical path. The system uses the property name specified in the net expression label to search upward from the cellview containing the net expression label (one instance at a time) to the top of the design. The first instance that is found that has a matching property name terminates the search. If the property is of type

`netSet` and is a legal net name, its value is used as the connecting net rather than the default net name specified in the expression.



The `netSet` property can be placed on any instance and it affects all net expression labels with matching property names at all levels below that instance unless overridden by another `netSet` property on a lower-level instance.

## Net and Pin Properties

The AMS environment supports the following properties. The last three are in addition to those listed in “[Properties](#)” on page 610.

<code>netType</code>	Specifies the type of a net. The type must be one of: <code>supply0</code> , <code>supply1</code> , <code>tri</code> , <code>tri0</code> , <code>tri1</code> , <code>triand</code> , <code>trior</code> , <code>triereg</code> , <code>wand</code> , <code>wire</code> , <code>wor</code> , <code>wreal</code> . For example, using the property <code>netType=wand</code> on net <code>a</code> , results in net <code>a</code> being declared in the netlist as  <code>wand a;</code>
<code>netDiscipline</code>	Specifies the discipline used to declare a net. This is sometimes called net coercion. For example, using the property <code>netDiscipline=electrical</code> on net <code>a</code> , results in net <code>a</code> being declared in the netlist as  <code>electrical a;</code>
<code>groundSensitivity</code>	For information, see “ <a href="#">groundSensitivity and supplySensitivity Properties</a> ” on page 173

`supplySensitivity` For information, see “[groundSensitivity and supplySensitivity Properties](#)” on page 173

## 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, sig2\_sensitive\_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.

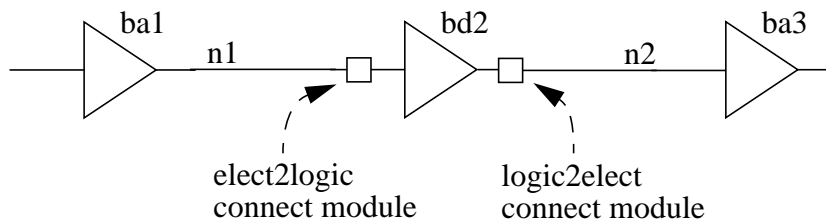
## Basic Principles for Using the Sensitivity Properties

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
        if(V(aVal) > 3.0)           // Digital, do this always.
            #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;
```

## Virtuoso AMS Environment User Guide

### Working with Schematic Designs

---

```
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 Inherited Connection Values

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



## Virtuoso AMS Environment User Guide

### Working with Schematic Designs

---

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.

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! "`.

## Virtuoso AMS Environment User Guide

### Working with Schematic Designs

---

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.

---

## Using External Text Designs

---

AMS Designer provides an efficient environment for developing and simulating designs. In addition, AMS Designer facilitates using existing and new designs developed outside of the AMS environment. This chapter describes how you can use these external designs in the AMS environment.

This chapter contains the following sections:

- [Overview of Steps for Using External Text Designs](#) on page 180
- [Bringing Modules into a Cadence Library](#) on page 180
- [Using Text Blocks in Schematics](#) on page 184
- [Using Modules Located in a Cadence Library](#) on page 186

**Note:** There are a few differences between using HDL modules and design units standalone and using them in the AMS environment.

- To avoid problems reading and writing a file, always use a full path when opening files inside a module using `$fopen`. The AMS environment might use a run directory that is in a different location than what you expect.
- When you are using the AMS environment, editing HDL files directly might cause problems. For more information, see [“Editing Verilog-AMS and VHDL-AMS Cellviews Outside of the AMS Environment”](#) on page 160.

## Overview of Steps for Using External Text Designs

The AMS environment works most efficiently with modules that are available in Cadence `Library.Cell:View (5X)` libraries. As a result, the strategy for using external text designs is to bring them into a library. A typical simplified series of steps looks like the following:

1. Specify the working library.
2. Use the `-use5x` option to compile modules into a library.
3. Create symbols for text modules that you plan to place in schematics.
4. Create a configuration that uses the text modules.
5. Run AMS Design Prep to generate a `cds_globals` module.
6. If necessary, edit the `cds_globals` module to add information about global variables and design variables found in the text modules.
7. Run the simulation.

These steps are described in more detail in the following sections.

## Bringing Modules into a Cadence Library

Bringing modules into Cadence libraries does not mean copying modules into the libraries. Rather, it means 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.

### Specifying the Working Library

Some of the `ncvlog` commands discussed in this chapter compile modules into the working library. You must specify the working library by using the command

```
DEFINE WORK libraryName
```

in your `hdl.var` file.

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, see [“The hdl.var File”](#) on page 58.

If the access permissions for the working library do not allow writing, you can establish a corresponding temporary library that does allow writing. For more information, see [“Compiling into Temporary Libraries”](#) on page 182.

## Compiling into Libraries

To bring modules into the working library, compile them with the `ncvlog` or `ncvhdl` commands using the `-use5x` option. For example, the command

```
ncvlog -ams -use5x master.vams
```

adds a cell named `master`, with the default view name `module`, to the current working library.

The command

```
ncvhdl -v93 -ams -use5x daconv2.vhd
```

where the `daconv2.vhd` file contains specifications for an entity named `daconv2` and an architecture named `daconv2_behav`, adds a cell named `daconv2` with views named `entity` and `daconv2_behav`, to the current working library.

The module to be added to the library can be located in any directory. The following example illustrates this usage by specifying the path to the module.

```
ncvlog -ams -use5x /mnt4/lgp/master.vams
```

If the file you are working with contains more than one module, you can specify the module to be compiled. For example, the following command compiles the `master1` Verilog-AMS module, which is located in the file `master.vams`.

```
ncvlog -ams -use5x -specificunit ncvlog_lib.master1:behavioral master.vams
```

This command adds a cell called `master1`, with a view called `behavioral`, to the `ncvlog_lib` library. The new `behavioral` view contains the `master1` module from the `master.vams` file.

If you need to bring primitives into a 5X library system, compile them into a library that is used only by primitives. As described in [“Using Modules Located in a Cadence Library”](#) on page 186, 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.

For detailed information about compiling into libraries, see Chapter 7 of *Cadence NC-Verilog Simulator Help* and Chapter 6 of *Cadence NC-VHDL Simulator Help*.

## 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. For more information, see [“Cadence Library Structure”](#), in the *Cadence Application Infrastructure User Guide*.

By default, only derived data is allowed in TMP libraries. This means that if the non-writable library already contains a particular cellview, you can compile it into a TMP library. Even with an associated temporary library, though, you are not allowed to add new views or new cells to a non-writable library. That restriction presents no problem when you want to use only cells and views that already exist in the library. However, if you want to add a new cell or view to the library, perhaps to substitute a new behavioral view for an existing library schematic view, the derived data restriction becomes an obstacle.

To circumvent this restriction, Cadence provides the `CDS_BIND_TMP_DD` shell environment variable.

### CDS\_BIND\_TMP\_DD Shell Environment Variable

The `CDS_BIND_TMP_DD` variable allows cells and views to be created in TMP libraries, even when corresponding master data does not exist in the master library. Using this variable allows some Cadence tools to create cell and view master data in TMP libraries, a behavior that is otherwise prohibited. The `CDS_BIND_TMP_DD` variable is most useful when the master library is read-only and you need to make new cells or views available for use in your design.

The following table describes the effects of the `CDS_BIND_TMP_DD` variable values, which are case-insensitive. Setting the `CDS_BIND_TMP_DD` variable to a value other than those listed has the same effect as not setting the variable at all.

---

Value	Effect
both, cell, true, or yes	Allow the creation, in a TMP library, of both new cells and new views, even when the corresponding master data does not exist in the master library.
view	Allows the creation, in a TMP library, of new views (only), even when the corresponding master data does not exist in the master library.

---

For example, you have a read-only working library called `amslib`. You want to assign that library to a TMP library and you want to be able to create new cells in the TMP library. In the `cds.lib` file for the library, you add an `ASSIGN` statement so the file contains the following statements:

## Virtuoso AMS Environment User Guide

### Using External Text Designs

---

```
DEFINE amslib ./AMS_lib/amsLib
ASSIGN amslib TMP /tmp/amslib_tmp
```

To permit the creation of new cells in the TMP library associated with the read-only `amslib` library, you set the environment variable on the command line.

```
setenv CDS_BIND_TMP_DD both
```

Then you use a command such as

```
ncvlog -ams -use5x -work amslib -view module compar6.vams
```

to compile the new `compar6.vams` module into the `amslib` TMP library.

## Listing Compiled Modules

AMS Designer keeps a list of what has been compiled, which you can query with the UNIX command

```
ncls -library library_name
```

For example, assuming that you have a working library named `amslib`, the command

```
ncls -library amslib
```

might produce a list like the following.

```
ncls: 04.20-a001: (c) Copyright 1995-2002 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.ELECTRICALSYSYSTEM (AST)
      package AMSLIB.ELECTRICALSYSYSTEM (COD)
      connect amslib.mixedsignal:connect (VST)
```

The first entry in this list, for example, says that in the `amslib` library, there is a cell called `comparator`, that has a view called `module`.

For more information on the `ncls` command, see the “`ncls`” section of the “Utilities” chapter, in the *NC Verilog Simulator Help*.

## Using Text Blocks in Schematics

The description in this section assumes that the text blocks to be instantiated in a schematic are already included in an accessible library. If this is not the case, you need to compile the blocks into a library. For more information, see [“Bringing Modules into a Cadence Library”](#) on page 180.

You must have a corresponding symbol before you can add a component to a schematic. If you need to create a symbol for a text view, the easiest way is to

1. Reopen the text cellview.

For example, you might select the cellview in the library manager and choose *File – Open* from the menu.

2. Save the text (as you do after editing the code), and then close the cellview.

A dialog box appears, asking whether you want to create a symbol for the component.

3. Click *Yes*.

Alternatively, you can also add a symbol view by following these steps:

1. Select *File – Open* from the CIW.

The Open File form appears.

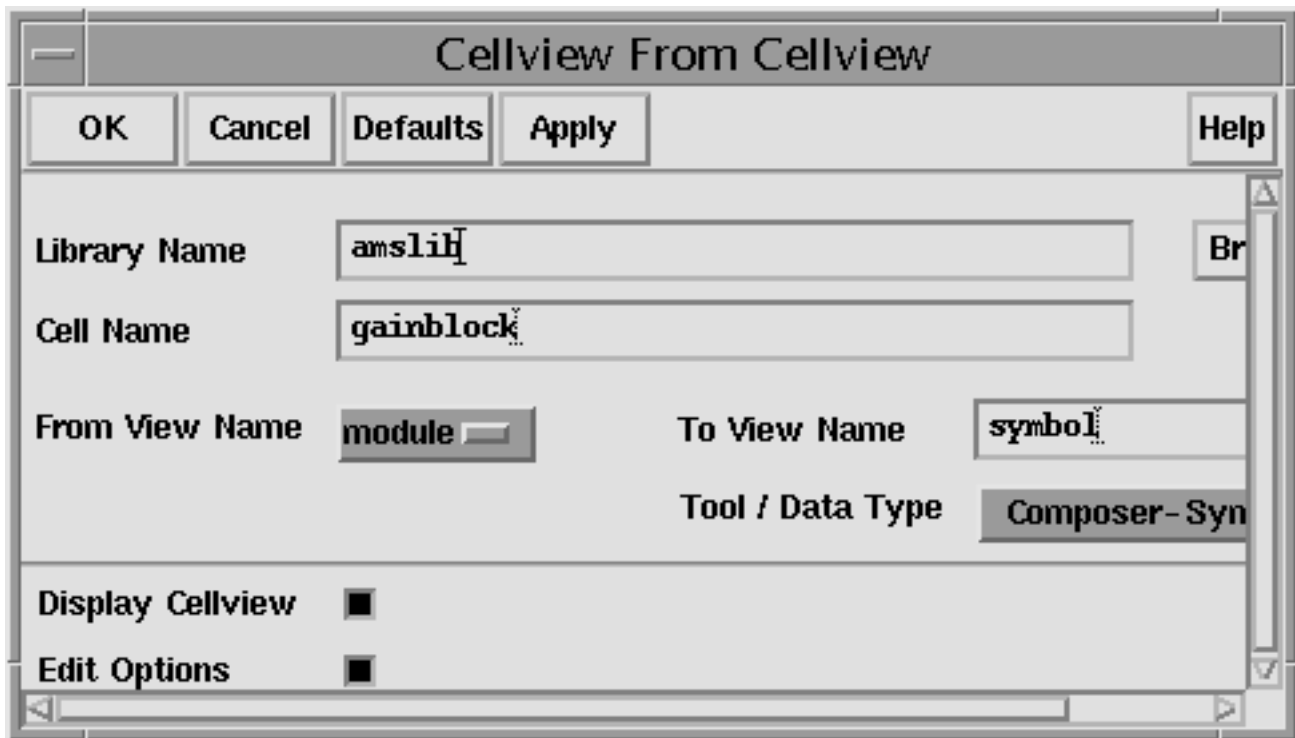
2. Open any schematic or symbol cellview.

The Symbol Editing window opens.

3. From the Symbol Editing window, select *Design – Create Cellview – From Cellview*.



The Cellview From Cellview form appears.



4. Type in the *Library Name* and *Cell Name* of the text block that lacks a symbol.
5. In the *From View Name* field, select the cellview of the text block.
6. In the *Tool / Data Type* field, select *Composer-Symbol*.

The *To View Name* field changes to symbol.

7. Click *OK*.

The Symbol Generation Options form opens.

8. Click *OK* in that form.

A Symbol Editing window appears.

9. Edit the symbol if necessary, save it, and close the Symbol Editing window.

At this point, you have a symbol representing the text block that you can place in a schematic.

## Using Modules Located in a Cadence Library

Once your modules are located in Cadence libraries, there are many tools that you can use to analyze and manipulate the modules. Perhaps the most important tool for the AMS environment is the Cadence hierarchy editor. With the hierarchy editor, you can create a configuration for your design and then run through the simulation process.

### Creating a Configuration

The Cadence hierarchy editor greatly facilitates the process of building components into your design by allowing you to create configurations that specify the view to be used for each cell. If you need more information about creating a configuration, see [“Creating a Config Cellview”](#) on page 200.

If you use the *AMS* template to create a configuration, the resulting configuration selects the views verilogams, veriloga, behavioral, functional, schematic, and symbol. However, if your design was created outside of the AMS environment, the *AMS\_Compatibility* template might do a better job of selecting appropriate views. Note that any selected symbol views must be associated with a model that the AMS simulator can use to describe the device.

If a newly created configuration does not automatically select the views that you need for each of the cells, you can select views manually. For guidance, see the “Defining Cell-Based Bindings and View Lists” section, in the “Overview” chapter of the *Hierarchy Editor Help*. The goal is to be sure that the views you want to simulate are actually specified in the configuration.

### Preparing for Simulation

The *AMS* entry, in the Cadence hierarchy editor menu, provides several tools that can help you prepare for and run a simulation that uses text modules.

- **AMS Design Prep**, which you run by choosing *AMS – Design Prep*, generates a `cds_globals` module. This module contains information about global signals and design variables contained in any translated schematics but not about those within text modules. If your text modules contain global signals or design variables, you can edit the `cds_globals` module to add them.
- Choosing *AMS – Options* opens a form that allows you to set options for the elaborator and the simulator.
- Choosing *AMS – Run Simulation* opens a form that allows you to run the simulator.

---

## Using Existing Designs in the AMS Environment

---

This chapter explains how to use analog primitives and SPICE and Spectre<sup>®</sup> netlists and subcircuits.

This chapter contains the following sections:

- [Using Analog Primitives](#) on page 188
- [Using SPICE and Spectre Netlists and Subcircuits](#) on page 188

## Using Analog Primitives

Analog primitives must be converted for use in AMS Designer. To do the conversion, you use the Conversion Tool Box, which is opened by choosing *Tools – Conversion Tool Box* from the CIW and then clicking *AMS simInfo from Spectre*. The tool box operates on an entire library at a time so the library must contain only analog primitives. For more information about using the Conversion Tool Box, see [“Converting an Existing Analog Primitive Library”](#) on page 598

This tool box assumes that the primitives to be converted are already prepared for use with Spectre Direct. If this is not true, you can

- Use another conversion tool to prepare the primitives for use with Spectre Direct and then do the conversion for AMS. For details about preparing primitives for use with Spectre Direct, see the “Preparing Files for Direct Simulation” section, in the “Converting Design Libraries and Technology Data” chapter of the *Compatibility Guide*.
- Manually create AMS simulation information for the primitives according to the definitions in [Appendix B, “Updating Legacy SimInfo for Analog Primitives,”](#) then use the steps in [“Placing SPICE and Spectre Netlists and Subcircuits in a Schematic”](#) on page 189 to edit the CDF simulation information.

## Using SPICE and Spectre Netlists and Subcircuits

This section describes briefly what you must do to use subcircuits, models, and netlists written in SPICE or Spectre. For additional information, see the [“Using Subcircuits and Models Written in SPICE or Spectre”](#) section, in the “Instantiating Analog Primitives and Subcircuits” chapter of the *Virtuoso AMS Simulator User Guide*.

### Preparing to Use SPICE and Spectre Netlists and Subcircuits

To prepare to use a subcircuit or a model written in SPICE or Spectre, you must

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 should start with `simulator lang=spectre` and SPICE files should start with `simulator lang=spice` and end with `simulator lang=spectre`. This practice protects against an inadvertent language change.

2. Give the elaborator the location of the model file.

You can meet this requirement either manually or by using the windows provided by the AMS environment.

- ❑ You can give the location manually by defining the `MODELPATH` variable in the `hdl.var` file, or by using the `-modelpath` option for `ncelab`. For example, an entry in the `hdl.var` file might look like this:

```
define MODELPATH model_filename
```

You can use the Analog Model Files form. For more information, see [“Specifying Model Files to Use During Elaboration”](#) on page 219.

## Placing SPICE and Spectre Netlists and Subcircuits in a Schematic

After preparing as described in the previous section, you can add a SPICE or Spectre netlist or subcircuit to a schematic. To do so, you

1. Create a symbol for the netlist or subcircuit.

You can use the Library Manager to copy an existing symbol, then modify the new symbol as necessary.

2. Add the CDF *termOrder* field simulation information for the symbol.

- a. From the CIW, choose *Tools – CDF – Edit*.

The Edit Component CDF form opens.

- b. In the Edit Component CDF form, select *Base* for the *CDF Type*.

- c. Type in the names of the library and cell that hold the newly created symbol view.

- d. Scroll down to the Simulation Information section and click *Edit*.

The Edit Simulation Information form appears.

- e. For the *Choose Simulator* field, select *ams*.

- f. Type the terminal names into the *termOrder* field.

Neglecting this step causes compilation errors such as

```
In AMS, named port association for analog  
primitives is not supported.
```

3. If your SPICE or Spectre netlist or subcircuit supports models (as, for example, the *bjt* and *resistor* do),

## Virtuoso AMS Environment User Guide

### Using Existing Designs in the AMS Environment

---

- a. Type `model` into the *instParameters* and *otherParameters* fields.

This tells the netlister that there is a model associated with the symbol.

- b. Type a `t` into the *isPrimitive* field.

This tells the netlister to translate the cell to a Spectre primitive that supports models.

At the end of this series of steps, the Edit Simulation Information form might look like this.

The screenshot shows the 'Edit Simulation Information' dialog box with the following fields and values:

Field	Value
Choose Simulator	ams
otherParameters	model
instParameters	w l as ad ps pd ld ls m trise model
enumParameters	region
referenceParameters	
stringParameters	
arrayParameters	
componentName	
termOrder	D G S B
propMapping	
extraTerminals	
isPrimitive	t

4. If you are placing a Spectre built-in primitive that does not support models (such as vdc or vsin), type the name of the primitive into the *componentName* field.
5. Click *OK* to close the Edit Simulation Information form.
6. In the Edit Component CDF form, click *Add* to open the Add CDF Parameter form.

Type in any required parameters. When you close the form, the added parameters appear in the *Component Parameters* pane of the Edit Component CDF form.

7. Close the Edit Component CDF form, by clicking *OK*.
8. Place the symbol in your schematic.

For more information about this step, see the [“Creating Schematics”](#) chapter of the *Virtuoso Schematic Editor User Guide*.

# **Virtuoso AMS Environment User Guide**

## Using Existing Designs in the AMS Environment

---



---

## Using Test Fixtures

---

Usually, a component is not complete in itself but requires stimuli to drive it. In response to the stimuli, the component produces outputs. To ensure that the outputs are what you expect, you can use a test fixture that instantiates the component, provides appropriate stimuli, and checks the outputs.

This chapter contains the following sections:

- [Creating and Using a Textual Test Fixture](#) on page 194
- [Example: Creating and Using a Test Fixture](#) on page 195

## Creating and Using a Textual Test Fixture

A schematic test fixture is structural only, instantiating the top-level module, analog stimuli, and analog sources that deliver analog stimuli. However, the approach described here uses textual test fixtures. Textual test fixtures often include behavioral code along with whatever instantiations are required.

Either approach assumes that you have a top-level view of the design to be tested. For example, the top-level view might be a VHDL or Verilog-AMS text module. Or the top-level design might be a schematic with, for example, hierarchical connectors that represent the pins of an integrated circuit. AMS Designer netlists schematics into Verilog-AMS modules so, in either case, the task ultimately becomes one of instantiating a textual module in the test fixture.

Test fixtures vary according to the kind of high-level view being tested and according to the different practices of designers. The test fixtures described in this chapter contain both structural and behavioral code.

### Creating a Textual Test Fixture

Most of the steps involved in creating a textual test fixture are the same as those used to create any other text module. However, the contents of the module reflect its position as the highest module in the hierarchy. You can create the test fixture either inside the AMS environment or outside of the environment.

You can use any of the AMS Designer supported languages for a test fixture, but differences in the way the languages are supported might tend to push you to one language in preference to another. The examples in this chapter use Verilog-AMS.

- To create a Verilog-AMS test fixture inside the AMS environment,
  - a. Select *File – New – Cellview* from the menu in the CIW.
  - b. In the Create New File dialog box, choose *VerilogAMS-Editor* in the *Tool* field.
  - c. Choose the library to hold the test fixture.
  - d. Type in the cell name and view name to be used for the test fixture. Remember that the cell name must be the same as the module name that you plan to use.
  - e. Click *OK*.
  - f. In the editor window that appears, compose the test fixture module. Save it when you are done.

## Virtuoso AMS Environment User Guide

### Using Test Fixtures

---

- To create the fixture outside of the AMS environment,
  - a. Use a text editor to create the module. When you are finished, save the file that contains the module.
  - b. Compile the module, being sure to use the `-use5x` option. This option creates a new cell and view for the test fixture in the working library.

Because the test fixture module is the highest simulation level, the module has no ports. The other content of the test fixture module depends on the inputs and outputs that you need to provide and examine. A typical Verilog-AMS structure looks 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
```

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 then react by providing new stimuli that depend on the outputs.

## Using a Test Fixture

To use the test fixture, you specify it as the highest level in a configuration, with the top-level module (the one being tested) at the next level below. The view to use for a Verilog-AMS test fixture is the Verilog-AMS view. For information on creating a configuration, see [“Creating a Config Cellview”](#) on page 200.

## Example: Creating and Using a Test Fixture

This simple example uses only two modules, both of them Verilog-AMS text modules. The first, `testfixture`, instantiates the other module `myswitch`. 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 goal of the example is to verify the operation of the `myswitch` module by instantiating the switch in the test fixture and then running a simulation.

First create the modules and bring them into the library.

1. Assume that the `myswitch` module exists in the file `switchcomps.vams`. The module (which is only one of the modules in the file) looks like this.

## Virtuoso AMS Environment User Guide

### Using Test Fixtures

---

```
//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
```

2. Compile the module into the `verilogams` view, of the `myswitch` cell, of the `amslib` library. The shorthand way of referring to this view is `amslib.myswitch:verilogams`. The appropriate command is:

```
ncvlog -ams -use5x -specificunit amslib.myswitch:verilogams switchcomps.vams
```

3. Create the test fixture in the AMS environment so it is located in the `amslib` library too. This module provides the inputs for the instantiated `myswitch` module and then reads the outputs to ensure that the module operates as it should.
  - a. Select *File – New – Cellview* from the menu in the CIW.
  - b. In the Create New File dialog box, choose *VerilogAMS-Editor* in the *Tool* field.
  - c. Choose the `amslib` library to hold the test fixture.
  - d. Type in the cell name `testfixture` and the view name `verilogams`. Remember that the cell name must be the same as the module name that you plan to use.
  - e. Click *OK*.
  - f. In the editor window that appears, type in the code for the module. Save it when you are done.

The code to use for the module is

```
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
```

## Virtuoso AMS Environment User Guide

### Using Test Fixtures

---

```
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
```

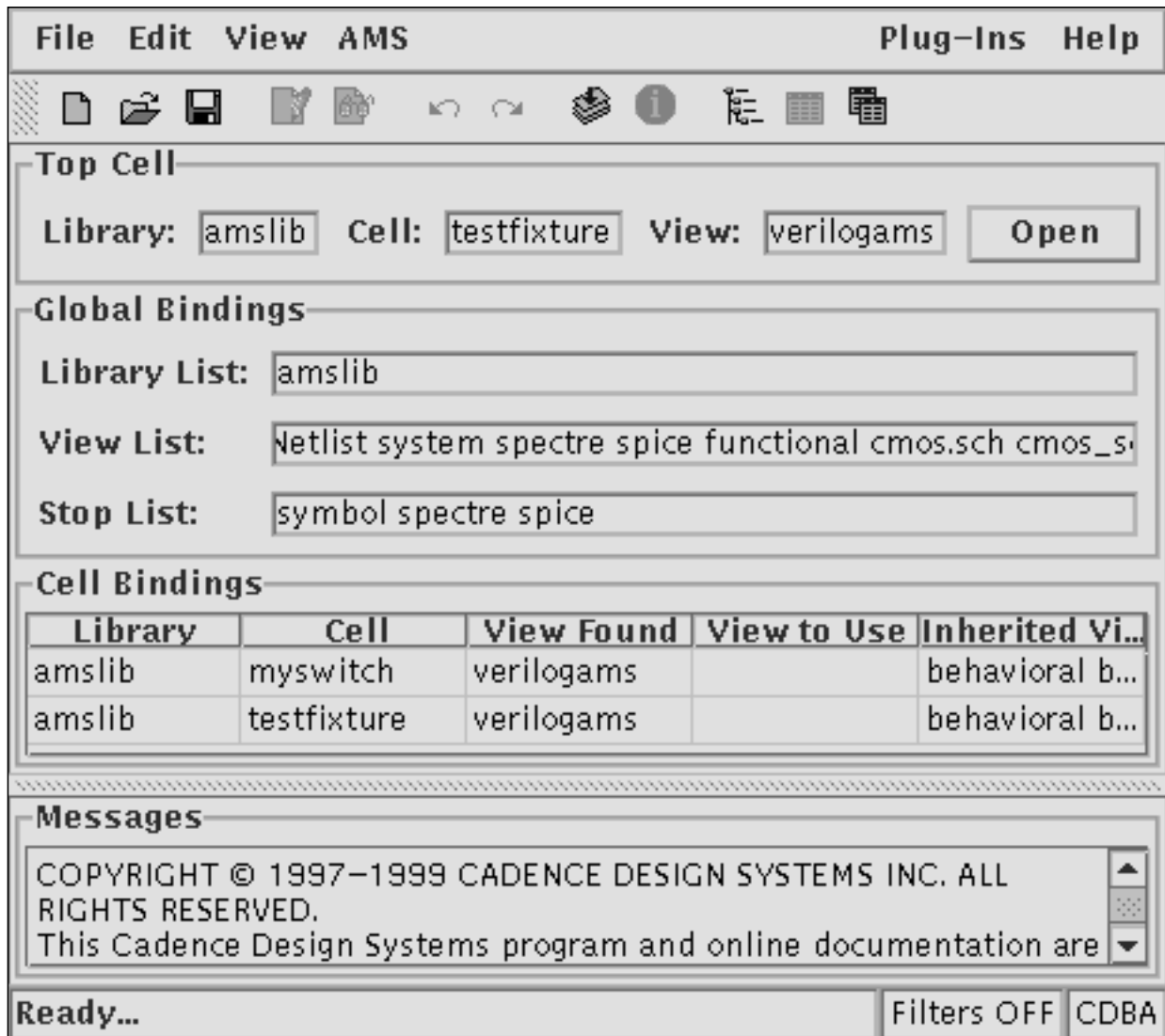
At this point both modules parse successfully and exist in the library. The next steps match the steps used to run any simulation in the AMS environment.

1. Create a config for the test fixture.

## Virtuoso AMS Environment User Guide

### Using Test Fixtures

Displayed in the hierarchy editor, it might look like this:



2. Use the selections in the AMS menu to prepare the design and simulate.

You can use SimVision to examine waveforms and use the information printed by the test fixture to determine whether your instantiated component works as desired.

---

## Using Design Configurations

---

This chapter describes how to use the Cadence® hierarchy editor to specify the cellviews that you want to use in your design. The set of rules that defines which cellviews under a cell are part of the design for a given purpose (such as netlisting), is called a configuration. The Cadence hierarchy editor lets you see and helps you understand the hierarchy of cellviews specified by the configuration rules.

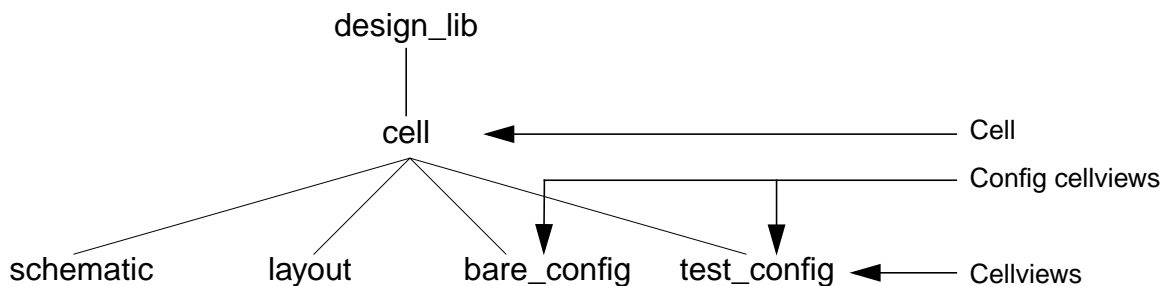
This chapter contains the following sections:

- [Overview of Configurations](#) on page 200
- [Creating a Config Cellview](#) on page 200
- [Using VHDL Modules in a Configuration](#) on page 206
- [Ensuring HDL Design Unit Information Is Current](#) on page 206
- [Using a Configuration](#) on page 206

## Overview of Configurations

Configurations are a convenient way to work with different cellviews as a design evolves from concept to finish. At the beginning of the design process, for example, you might want to use high-level behavioral modules to describe the overall behavior of a component. Then, you might want to insert modules into test fixtures. Finally, you might want to replace the behavioral description with a detailed schematic. Configurations allow you to do all those tasks easily.

A configuration is stored as a cellview, called a *config*, of the cell. You can have different config cellviews for different purposes.



You can make configuration rules defining what views are to be included in the hierarchy at three different levels:

- At the global level, using a global view list and stop list
- At the cell level, using cell-based view lists, which affect the cell as well as structures below the cell in the hierarchy, and cell bindings.
- At the instance level, using instance-based view lists, which affect the instance as well as structures lower in the hierarchy, and instance bindings.

To simulate your design in the AMS environment, you must have a top-level config cellview. That top-level config can include other config cellviews lower in the hierarchy. (You can, however, run the simulator outside of the environment without using a config cellview.)

## Creating a Config Cellview

Your design hierarchy must be specified by a top-level config cellview, which can contain other config cellviews lower in the hierarchy. You can use the Cadence hierarchy editor to create a config cellview from a schematic cellview by following the simplified steps given below. For details of this procedure, see the [\*Cadence Hierarchy Editor User Guide\*](#).



## Virtuoso AMS Environment User Guide

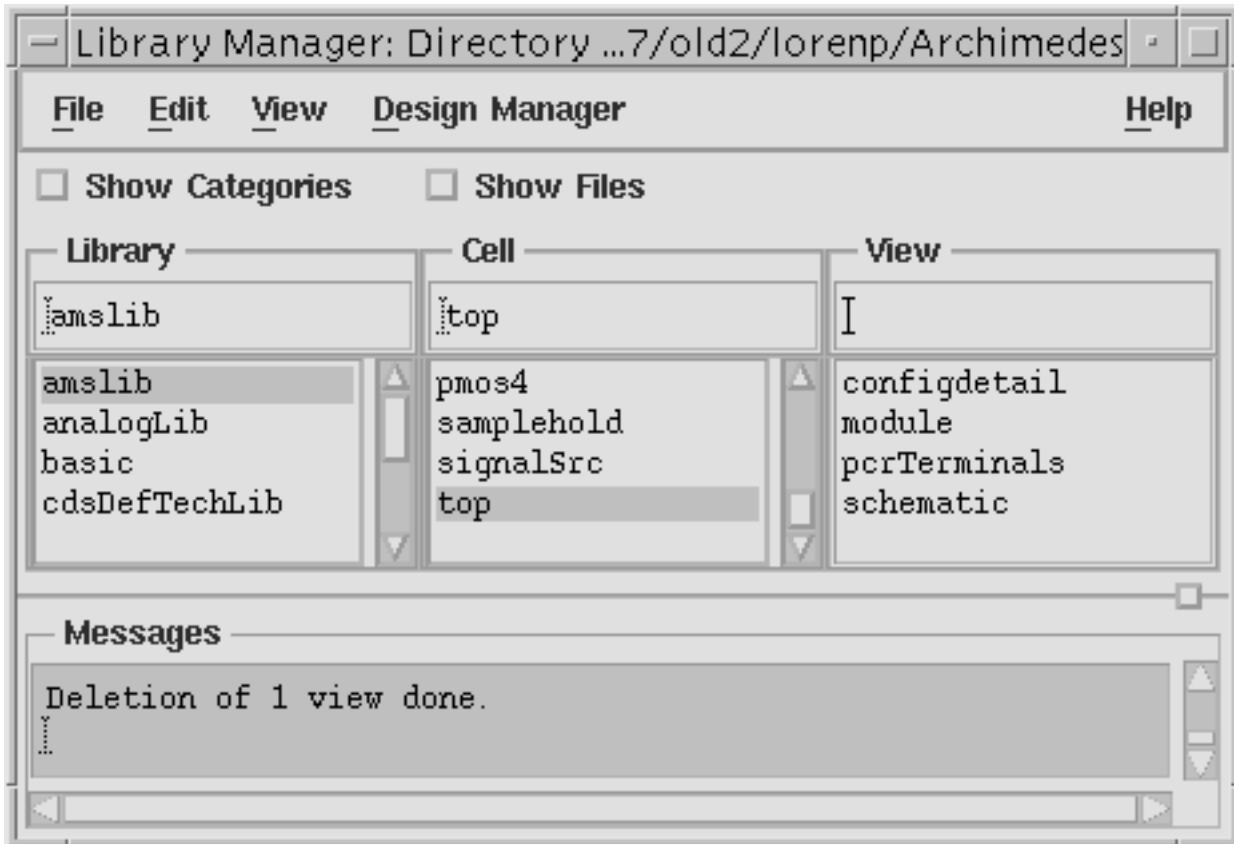
### Using Design Configurations

---

1. From the CIW, choose *Tools – Library Manager*.

The Library Manager window appears.

2. Highlight the library and cell for which you want to create a config cellview.



3. Choose *File – New – Cell View*.

The Create New File form appears.

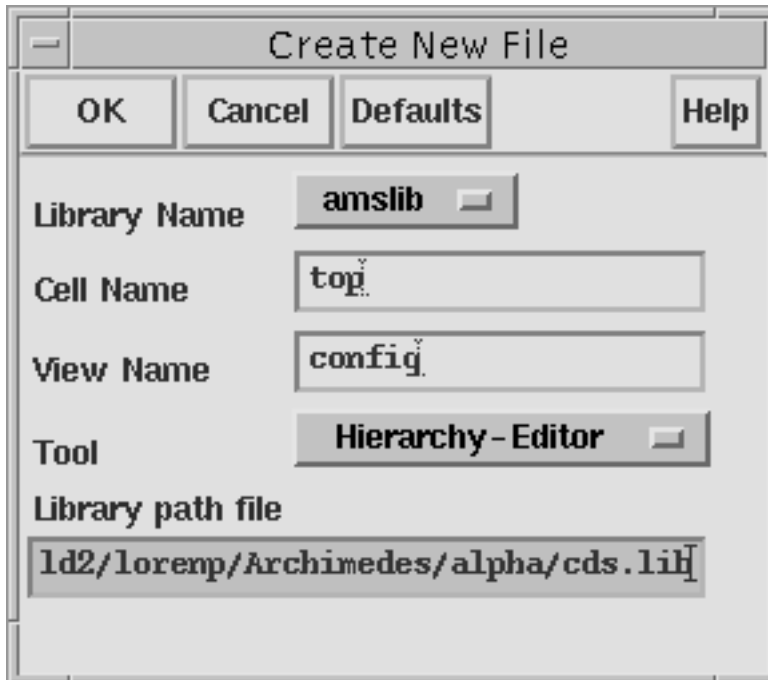
4. In the *Tool* field, choose *Hierarchy–Editor*.

## Virtuoso AMS Environment User Guide

### Using Design Configurations

---

The *View Name* automatically changes to *config*.



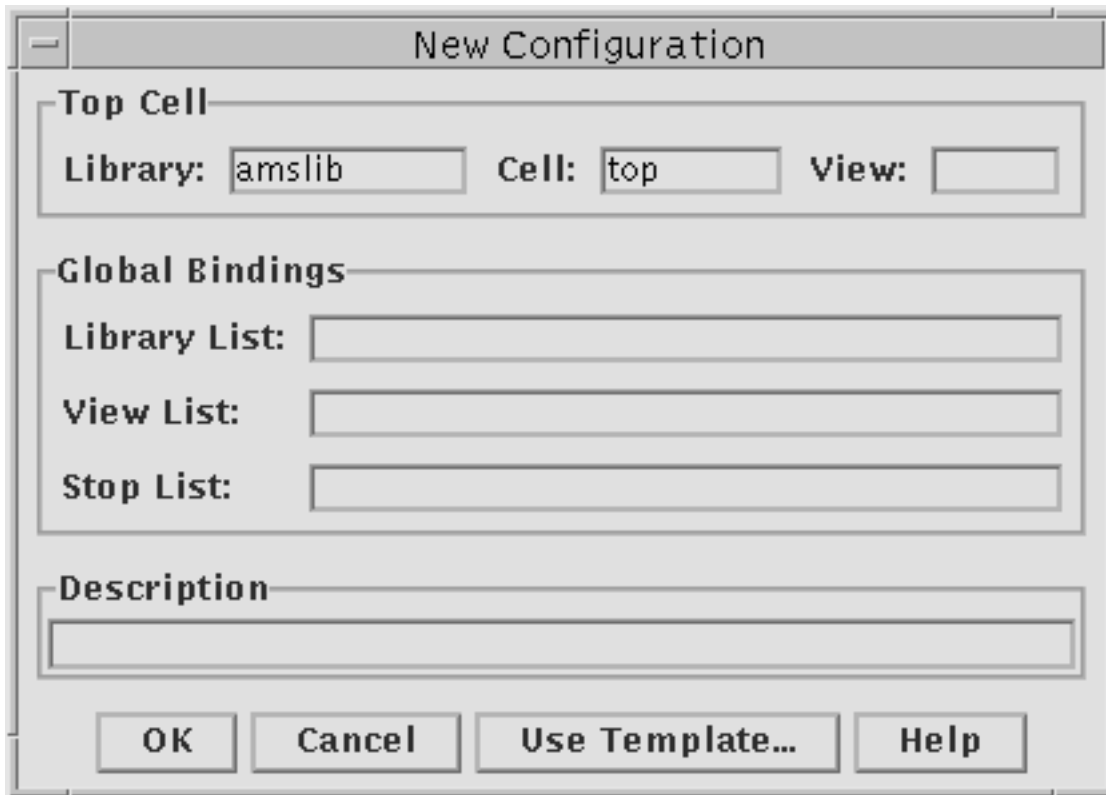
5. Click OK.

## Virtuoso AMS Environment User Guide

### Using Design Configurations

---

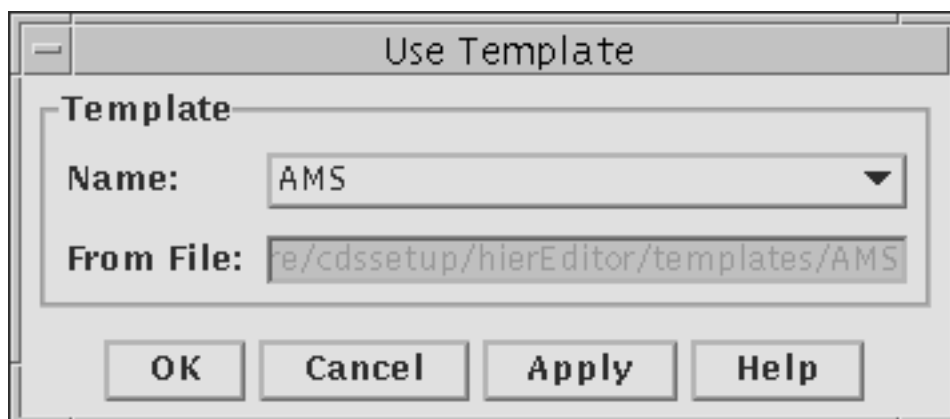
The Cadence hierarchy editor window and the New Configuration form appear.



The 'New Configuration' dialog box is shown. It has a title bar with a minus button and the text 'New Configuration'. The dialog is divided into several sections: 'Top Cell' with fields for 'Library:' (containing 'amslib'), 'Cell:' (containing 'top'), and 'View:' (empty); 'Global Bindings' with fields for 'Library List:', 'View List:', and 'Stop List:' (all empty); and a 'Description' section with a large empty text area. At the bottom are four buttons: 'OK', 'Cancel', 'Use Template...', and 'Help'.

6. In the New Configuration form, click *Use Template*.

The Use Template form appears.



The 'Use Template' dialog box is shown. It has a title bar with a minus button and the text 'Use Template'. The dialog contains a 'Template' section with a 'Name:' field (a dropdown menu showing 'AMS') and a 'From File:' field (containing the path 'fe/cdssetup/hierEditor/templates/AMS'). At the bottom are four buttons: 'OK', 'Cancel', 'Apply', and 'Help'.

7. Choose *AMS* or *AMS\_Compatibility* from the *Name* cyclic field.

- ☐ The *AMS* template provides the global bindings that are likely to be most useful for designers who follow the guidelines in this book.

## Virtuoso AMS Environment User Guide

### Using Design Configurations

---

- ❑ The *AMS\_Compatibility* template provides global bindings that are likely to be of use with designs that originate in other flows, such as flows that use vhdINet, verilogIn, or Verimix.

**8.** Click *OK*.

The New Configuration form redisplayes with the default library list, view list, and stop list for the AMS simulator.

- 9.** Ensure that the *Library List* includes `analogLib` if your design has text views that use components from that library.
- 10.** In the New Configuration form, ensure that the *Library*, *Cell*, and *View* refer to the cellview from which you want to create a configuration.

**New Configuration**

**Top Cell**

**Library:**  **Cell:**  **View:**

**Global Bindings**

**Library List:**

**View List:**

**Stop List:**

**Description**

Default template for AMS

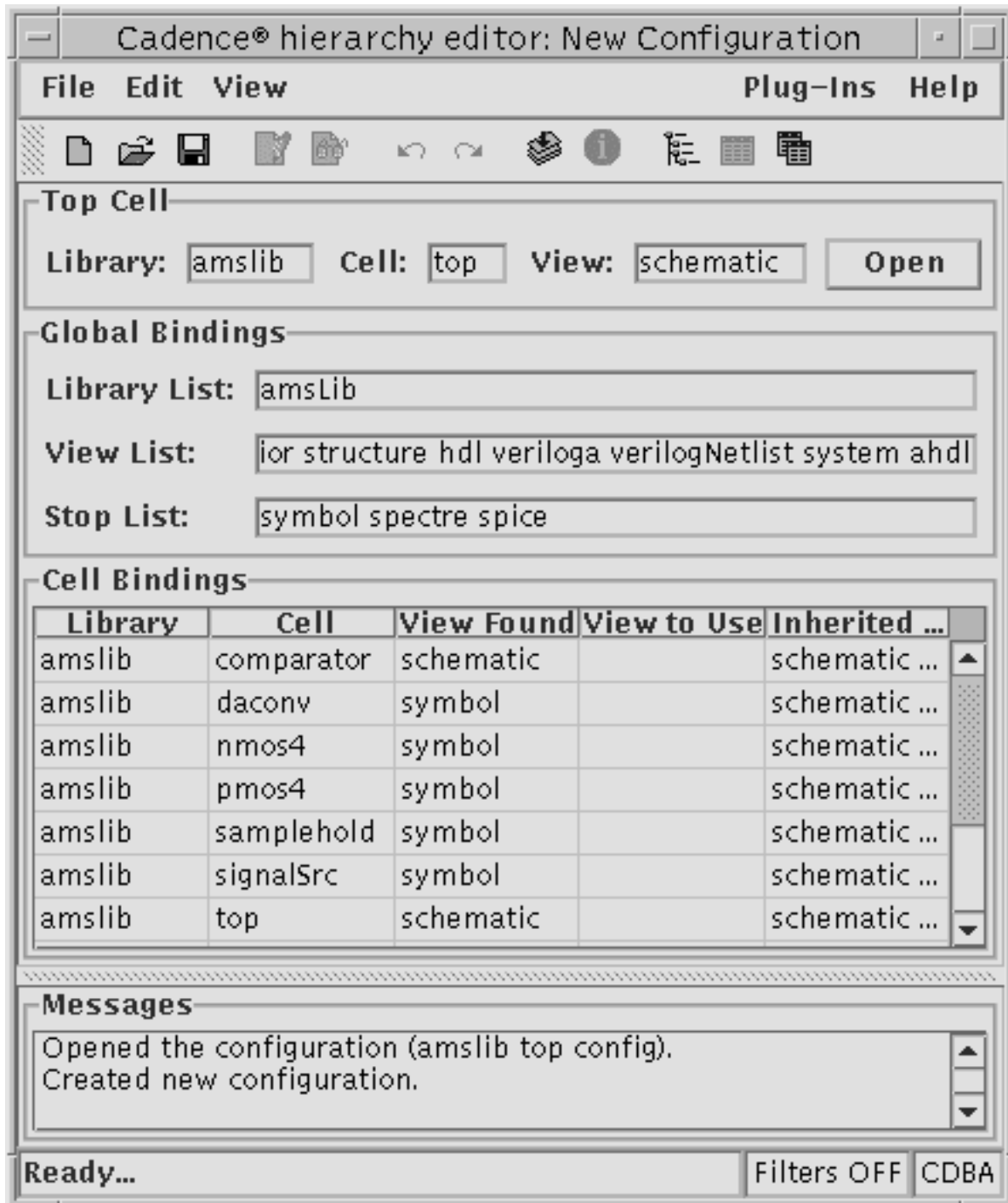
Note:  
Please remember to replace the Library, Cell, and View fields for the Top Cell with the actual names used by your design.

**11.** Click *OK*.

## Virtuoso AMS Environment User Guide

### Using Design Configurations

The Cadence hierarchy editor redisplays with the default cell bindings.



12. If you need to modify any of the *Global Bindings* lists, you can type directly in the fields. For example, VHDL users might want to add a wildcard asterisk to the *View List*.

**13.** To save this new config view, select *File – Save* in the hierarchy editor.

For additional background information and more information about creating configurations, see the *Cadence Hierarchy Editor User Guide*. Continue reading here for information on setting up a configuration for mixed-language designs in the AMS environment.

## Using VHDL Modules in a Configuration

To instantiate a VHDL module in a Verilog®-AMS module, you use the architecture view of the VHDL module in the configuration. For an example that instantiates a VHDL module, see how the `vhdl_clock` is used in [Chapter 2, “Quick-Start Tutorial.”](#)

You can also instantiate Verilog-AMS modules in VHDL modules. For information, see the [“Importing Verilog-AMS Modules into VHDL Modules”](#) chapter, in the *Virtuoso AMS Simulator User Guide*.

## Ensuring HDL Design Unit Information Is Current

If you edit an HDL design unit, such as one of the netlists named `verilog.vams`, using an editor such as `vi`, you must then run AMS Design Prep to ensure that the 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. If you use the AMS environment, for example, the AMS Run Simulator form prepares an `ncelab` command like the following:

```
ncelab amslib.top:config amslib.cds_globals:top_config ConnRules_5V_full  
-discipline logic -timescale 1ns/1ns -noparamerr -use5x4vhdl
```

Notice how, in this command, the first object passed to `ncelab` is the configuration, `amslib.top:config`. Similarly, the command for the simulator is:

```
ncsim amslib.top:config -amslic -analogcontrol top.sce -GUI -input text.tcl
```

In this command too, the first object passed is the configuration.

As you use configurations, be aware of the following guidelines.

- Compile the design with the `-use5x` command line option and ensure that the design is located in a Cadence library. For more information, see the [“ncvlog Command Syntax”](#) section, in the *“Compiling”* chapter, of the *Virtuoso AMS Simulator User Guide*.

## Virtuoso AMS Environment User Guide

### Using Design Configurations

---

- Use the `-use5x4vhdl` command line option when you elaborate the design. This option applies configurations to VHDL modules. For more information, see the “[-USE5x4vhdl Option](#)” section, in the “Elaborating” chapter, of the *Virtuoso AMS Simulator User Guide*.
- By default, `ncelab` places the simulation snapshot in the `cellview` directory of the first design unit specified on the `ncelab` command line. To specify a different location, use the `-snapshot` option.

If you develop and simulate your design within the AMS environment, these guidelines are followed automatically.

## **Virtuoso AMS Environment User Guide**

### Using Design Configurations

---



---

## Preparing a Design for Simulation

---

This chapter discusses what AMS Design Prep does, when to use it, and how to use it to prepare a design for simulation.

The chapter contains the following sections:

- [Overview of AMS Design Prep](#) on page 210
- [Specifying the Behavior of AMS Design Prep](#) on page 211
- [Running AMS Design Prep](#) on page 221
- [How AMS Design Prep Handles Global Signals and Design Variables](#) on page 225

## Overview of AMS Design Prep

Virtuoso® AMS Design Prep prepares a design for simulation by creating Verilog®-AMS netlists for CDBA cellviews in your design. This preparation ensures that the Verilog-AMS netlist and compiled representations generated from a CDBA cellview are up-to-date. View types that are eligible for netlisting are `schematic`, `symbolic`, `maskLayout`, and `netlist`.

When AMS Design Prep prepares a CDBA cellview for simulation, it calls the AMS netlister, which produces a netlist file called `verilog.vams`. AMS Design Prep also creates the `cds_globals` module, which contains global signals and design variables information.

## What AMS Design Prep Does to Prepare a Design for Simulation

AMS Design Prep performs the following tasks to prepare a design for simulation:

- Traverses the design configuration using the hierarchy editor traversal engine.
- Calls the AMS netlister to generate netlists for CDBA cellviews. See [Chapter 4, “Netlisting”](#) for more information.
- Calls `ncvlog` or `ncvhdl` to compile netlists and other HDL views. See [“compileMode”](#) on page 398 for details about how the compilers determine which views to compile.
- Enables the use of CDBA global signals by creating a `cds_globals` module if global signals exist in the design. The user interface that you use to edit global signals is discussed later in this chapter.
- Enables the efficient use of multiple electrical ground references by taking advantage of the Verilog-AMS `ground` declaration. Each CDBA global signal that is specified as an electrical ground reference is associated with the Verilog-AMS global ground reference node.
- Enables the use of design variables by collecting all design variables in the design hierarchy, provides a user interface that you can invoke from the *AMS* plug-in menu in the Cadence hierarchy editor so you can edit their values, and creates `dynamicparam` statements in the `cds_globals` module. See [“Specifying Design Variables”](#) on page 216 for more information on the AMS Design Variables user interface.
- Allows re-netlisting and re-compilation of the entire design to ensure consistency. You can re-netlist all CDBA cellviews and re-compile all HDL units used in the design hierarchy. See [“Specifying the Behavior of AMS Design Prep”](#) on page 211 for more information.

## **When to Use AMS Design Prep**

You need to run AMS Design Prep once to prepare the entire design before the first simulation. After the first simulation, you need to run AMS Design Prep when

- The design needs netlist consistency checks
- You add or change one or more cellviews in your design and netlists are not automatically created for those views when you check and save your design
- You add new global signals or design variables to your design

If you are uncertain whether you need to run AMS Design Prep, go ahead and run it. Although it might cost you additional time, it never hurts to run AMS Design Prep on your entire design.

Not running AMS Design Prep reduces iteration time if you are certain that you have not introduced any new design variables or global signals. If you have introduced new design variables or global signals or have neglected to run the AMS netlister for a schematic, the elaboration step fails. That indicates that you need to run AMS Design Prep on your design.

## **Specifying the Behavior of AMS Design Prep**

AMS Design Prep is a flexible tool that allows you to process your design in various ways. You can change how the tool operates by changing options.

### **Setting Options for Global Design Data**

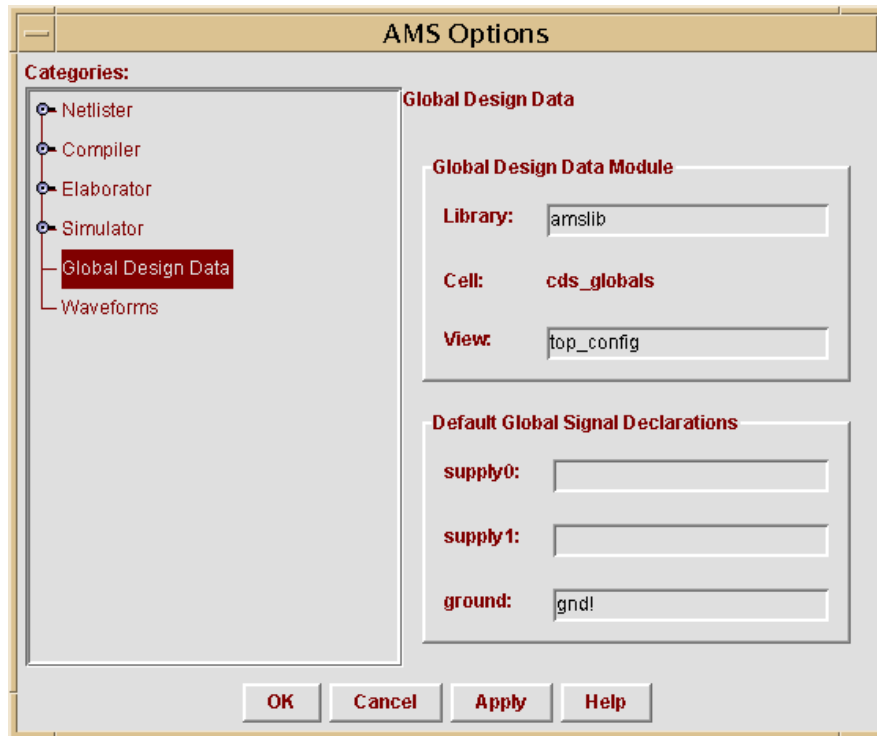
You can access the option settings for the AMS netlister and for AMS Design Prep by choosing *AMS – Options – Global Design Data* in the hierarchy editor. (If you need

## Virtuoso AMS Environment User Guide

### Preparing a Design for Simulation

---

information about installing the AMS menu item in the hierarchy editor, see [“Preparing to Use AMS Designer from the Hierarchy Editor”](#) on page 62.)



This section describes only the *Global Design Data* category. For information about the *Netlister* and *Compiler* categories, see [“Specifying the Behavior of the Netlister and Compilers”](#) on page 74. For information about the *Elaborator* and *Simulator* categories, see [“Specifying the Behavior of the Elaborator, Simulator, and Waveform Viewer”](#) on page 232. For information about the *Waveforms* category, see [“Setting Waveform Selection Options”](#) on page 274.

AMS Design Prep automatically generates the `cds_globals` cell that contains the global signals and design variables. The cell name, `cds_globals`, is fixed, but you can specify a library name and view name in the *Global Design Data Module* pane of the window.

The fields in the *Default Global Signal Declarations* pane contain lists of signals. AMS Design Prep uses the values in these fields to assign default wire types and to select grounds. For example, if the `supply1` field contains `vdd! dvdd!`, then when AMS Design Prep finds a new global signal named `dvdd!`, it declares `supply1` as the wire type for `dvdd!`.

Each list of signals must consist of strings, such as `gnd!`, `vdd!`, or `agnd!`.

Note that a specification in the AMS Global Signals window overrides, for that signal, the default settings from the *Default Global Signal Declarations* pane. The AMS Global Signals window is discussed in the next section.

## Specifying Global Signals

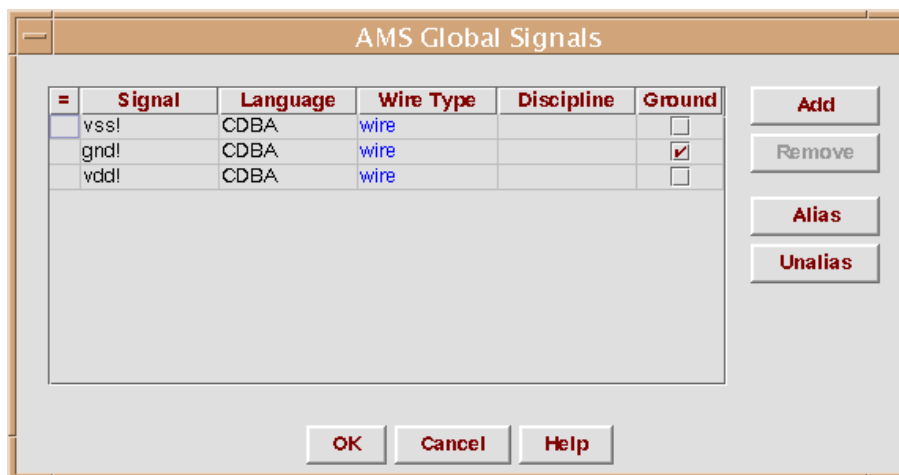
Global signals can come from master CDBA (schematic) data or from out-of-module references in master Verilog (digital) or Verilog-AMS HDL data. AMS Design Prep is aware of only global signals that come from master CDBA data. You can use the AMS Global Signals window to declare a signal that is used as an out-of-module signal reference in the master Verilog (digital) or Verilog-AMS HDL data.

AMS Designer implements global signals as out-of-module references, but does not support out-of-module references either into or out of VHDL-AMS design units. As a result, AMS Designer does not support using global signals in VHDL-AMS design units, even if you enter the VHDL-AMS global signals in the AMS Global Signals window.

When you make changes in the AMS Global Signals window and click *OK*, AMS Design Prep

- Creates and compiles the `cds_globals` module if it does not already exist.
- Regenerates and recompiles the `cds_globals` module if it does already exist.

To display the AMS Global Signals window, select *AMS – Global Signals*.



The display color of the entries in the *Signal* and *Language* fields indicates if AMS Design Prep found the signal, or if you added it. If you added the signal, it is displayed in blue, and if it is found by AMS Design Prep, it is displayed in black.

The black color means that you cannot remove that signal; you cannot remove a global signal found by AMS Design Prep or add a signal with the same name as a global signal found by AMS Design Prep. However, if you added a global signal in a previous design session, and it has the same name as a global signal found by AMS Design Prep during the current design session, then the global signal that you added in the previous session is used, and it is displayed in black, so you cannot remove the signal. This behavior is useful when you change the configuration data while switching back and forth between behavioral and schematic views.

If AMS Design Prep cannot find a global signal in the CDBA data in the current run, but found it in a previous run, the signal is displayed in red. You can remove a signal that is displayed in red because it is not needed for the current design configuration.

If the controls are disabled, AMS Design Prep might be unable to write to the `cds_globals` module where the information in the window is stored. For more information, see [“How AMS Design Prep Handles Global Signals and Design Variables”](#) on page 225.

## Adding a Global Signal

To add a global signal

1. Click *Add*.
2. Click in the *Signal* field for the new line that appears, and type the name of the signal you want to add.
3. Specify the language in which the signal is declared (CDBA, Verilog, Spectre, SpectreHDL, or SPICE) in the *Language* field.
4. Choose a wire type from the drop-down menu that appears when you click on the *Wire Type* field.

You can choose one of the following wire types: `wire`, `tri`, `wor`, `wand`, `trior`, `trand`, `triereg`, `tri0`, `tri1`, `supply0`, or `supply1`.

5. Enter the discipline in the *Discipline* field.

The discipline specification that you enter in this field is written to the `cds_globals` module. If you do not specify a discipline, no discipline specification is written to the `cds_globals` file and the discipline of the signal is determined by discipline resolution during elaboration.

For more information, see the [“Disciplines”](#) section, in the “Data Types and Objects” chapter, of the *Cadence Verilog-AMS Language Reference*.

## Virtuoso AMS Environment User Guide

### Preparing a Design for Simulation

---

6. If you want to use the global signal as a ground reference, select the ground option by clicking in the check box in the *Ground* field for the appropriate signal.

### Deleting a Global Signal

- To remove a global signal, select the signal by clicking on any field associated with the signal you want to remove, and then click *Remove*.

### Aliasing and Unaliasing Global Signals

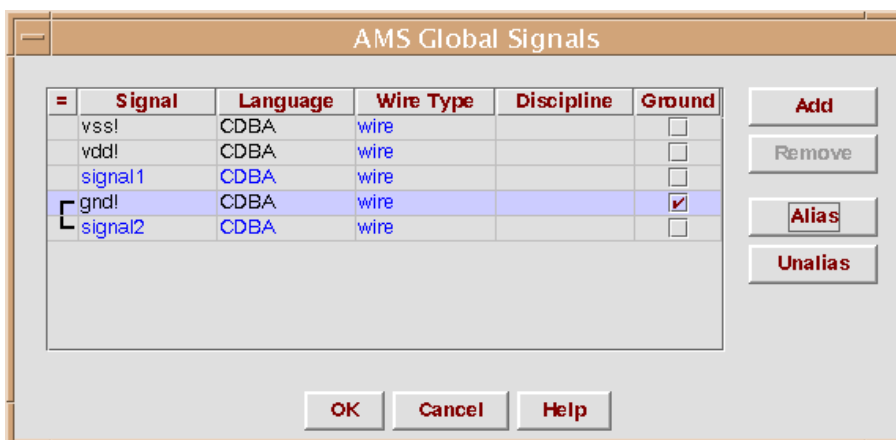
You can alias global signals into groups. Aliased signals in a group are electrically equivalent, as if they are joined by a wire. For more information, see [“How Aliased Signals Are Netlisted”](#) on page 131.

#### Aliasing Global Signals

To alias signals

1. Select the signals to be aliased, and click *Alias*.

To select signals listed consecutively, hold down the shift key while you click on the signal names to be aliased. To select signals that are not listed sequentially, hold down the control key while you click on the signal names. When you alias signals, they redisplay consecutively in the global signal list, joined by a vertical connecting bar, as shown in the example below.



If you alias signals belonging to separate aliased signal groups, all of the signals in the groups are aliased.

2. When you are finished editing the list of global signals, click *OK* for your edits to take effect.

### Unaliasing Global Signals

To unalias signals

1. Select the signals to be unaliased from the group, and then click *Unalias*.
2. When you are finished editing the list of global signals, click *OK* for your edits to take effect.

## Specifying Design Variables

Design variables can be used in master CDBA or master HDL data but AMS Design Prep is aware of only the former. AMS Design Prep is aware of design variables that are copied to CDBA cellviews by the Cadence analog design environment. If a design variable is set in multiple locations in a hierarchy, the effective CDBA value is the one set at the highest level of the hierarchy. That value, in turn, can be overridden by specifying a value in the AMS Design Variables form.

AMS Designer implements design variables as out-of-module references, but does not support out-of-module references either into or out of VHDL-AMS design units. As a result, AMS Designer does not support direct access of design variables in or from within VHDL-AMS design units, even if you enter the VHDL-AMS design variables in the AMS Design Variables window.

In addition, AMS Design Prep assumes that variables are design variables when

- The variables are used in the values of properties or parameters, and
- The variables are not themselves defined as cellview properties or parameters.

For example, a parameter `phase` has the value `sin(x)*3` where the variable `x` is not defined as a parameter. AMS Design Prep assumes that the variable `x` is a design parameter and generates a netlist with the statement `.phase(sin(cds_globals.x)*3)`.

You can use the AMS Design Variables window to add a new design variable or to remove a design variable that is in the master HDL data. You can also set the values of design variables found in master CDBA data and revert back to the original CDBA values when necessary.

When you make changes in the AMS Design Variables window and click *OK*, AMS Design Prep

- Creates and compiles the `cds_globals` module if it does not already exist.



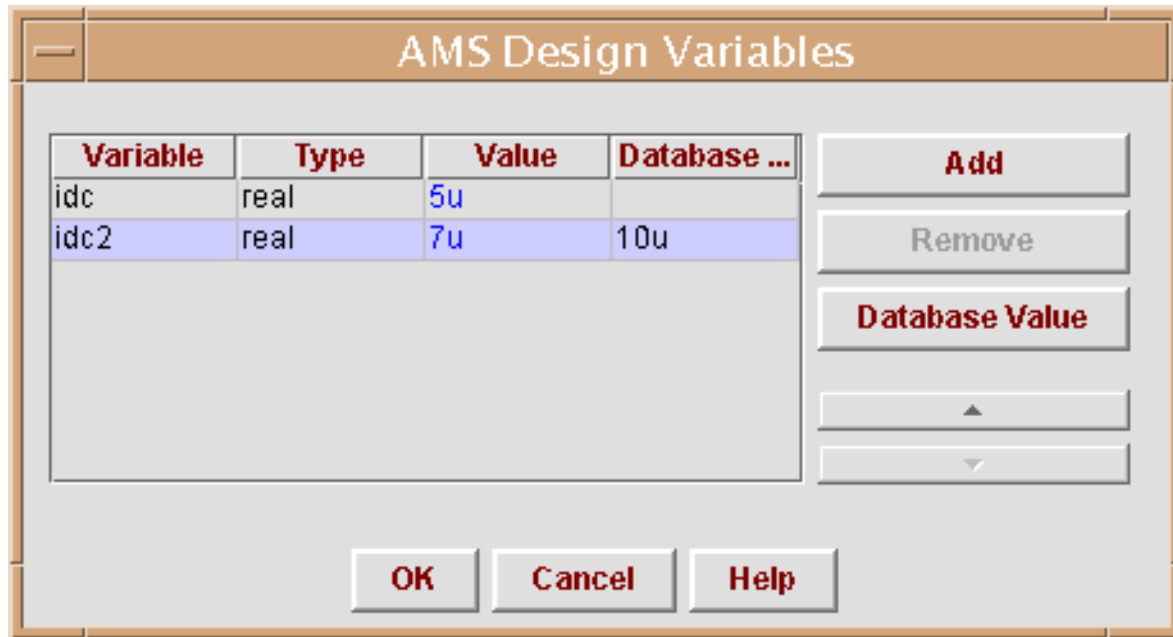
## Virtuoso AMS Environment User Guide

### Preparing a Design for Simulation

---

- Regenerates and recompiles the `cds_globals` module if it does already exist.

To display the AMS Design Variables window, choose *AMS – Design Variables*.



The display color of the values in the table indicates whether AMS Design Prep found the design variable or you added it. If you added the design variable, the value displays in blue. If AMS Design Prep found the design variable, the value displays in black.

The black color means that you cannot remove that variable. You cannot remove a design variable found by AMS Design Prep or add a design variable with the same name as a design variable found by AMS Design Prep because doing so results in an undefined reference when you elaborate the design. However, if you added a design variable in a previous design session, and it has the same name as a design variable found by AMS Design Prep during the current design session, then the design variable that you added in the previous session is used, and its value displays in black, so you cannot remove the design variable. This behavior is useful when you change the configuration data while switching back and forth between behavioral and schematic views. If AMS Design Prep cannot find a design variable in the CDBA data in the current run, but found it in a previous run, the value of the variable displays in red. You can remove a variable that displays a red value because the variable is not used in the current configuration.

If the controls are disabled, AMS Design Prep might be unable to write to the `cds_globals` module where the information in the window is stored. For more information, see [“How AMS Design Prep Handles Global Signals and Design Variables”](#) on page 225.

## Adding a Design Variable

To add a design variable,

1. Click *Add*.

A new row, representing a new design variable, appears in the table.

2. (Optional) Click in the *Variable* field of the new variable and change the default name to the name you want to use.
3. Choose *real* or *integer* for the *Type* field.
4. Type the value of the variable in the *Value* field.

The value you enter can have a scaling factor but must not have a measurement unit. Measurement units are not supported by the Verilog-AMS language.

5. If necessary, highlight a row and click the up or down arrows to rearrange the order of the variables.

You might need to do this if you have a design variable whose value depends on the value of another parameter. For example, if you have a parameter `idc` with the value `5 * myparam`, you must ensure that `myparam` is defined earlier (higher) in the list of design variables.

6. When you finish adding design variables, click *OK*.

## Deleting a Design Variable

- To remove a design variable, click the name of the variable to highlight it, then click *Remove*.

## Editing the Value of a Design Variable

You can edit the value of any design variable. The design variable values are saved verbatim in the `cds_globals` module, and are used during simulation. To edit a design variable,

1. Set the new type and value in the *Type* and *Value* fields of the variable that you want to change. Note, however, that although you can change the value of a variable found in the CDBA data, you cannot change the type.

The value you enter can have a scaling factor but must not have a measurement unit.

2. When you finish editing design variables, click *OK*.

## Reverting to the Value Used in the CDBA

To revert the value of a design variable found in the CDBA to the value used in the CDBA,

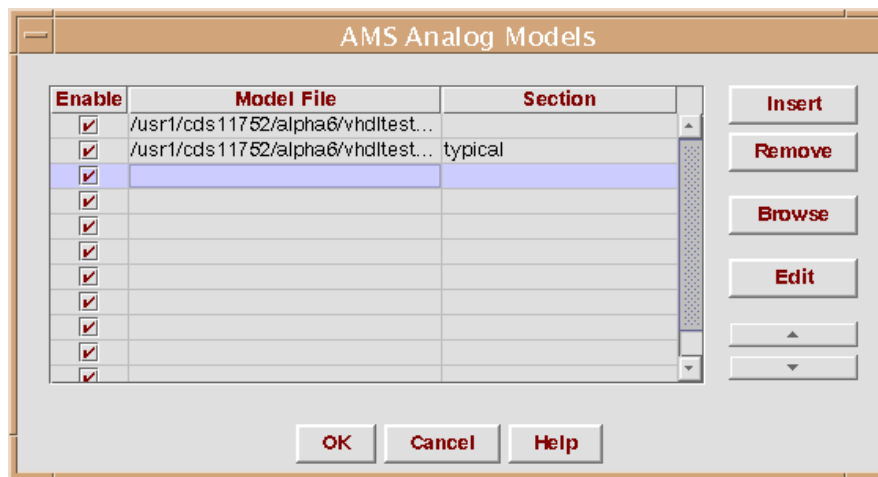
1. Click the name of the design variable in the *Variable* column.
2. Click *Database Value*.

The *Database Value* button is active only when the highlighted variable is found in the CDBA and has a value there.

3. When you finish editing the list of design variables, click *OK*.

## Specifying Model Files to Use During Elaboration

For elaboration, you can specify model files and sections in the AMS Analog Models window. To display the Analog Models window, choose *AMS – Analog Models* in the Cadence Hierarchy Editor window.



## Adding an Analog Model File

To add an analog model file to the table,

1. Do either of the following:
  - ☐ Click on the *Model File* field of an empty row.
  - ☐ Highlight a row and then click *Insert*.

A new row appears in the table above the highlighted row.

2. Type the path and name of the analog model file in the *Model File* field. You can do either of the following:

- ☐ Type the information in the fields.

If you like, you can use environment variables to specify the path. For example, you might type the following into the *Model File* field.

```
$HOME/test/pwr_supply/models/models.scs
```

- ☐ Click *Browse* to open a dialog that allows you to navigate through your directories to identify the appropriate model file.

When you click away from the row, AMS Designer checks for the existence of the file that you specify and highlights the row in red if the file is not found.

3. (Optional) Type the section that you want to use into the *Section* field.

**Note:** AMS Designer does not check for the existence of the section, so be sure that you enter a correct section name.

To specify multiple sections from a single model file, use multiple rows in the table, each with the same model file but with a different section.

You can enable multiple sections only if every model included in those sections is unique. For example, you cannot specify and enable both the section `typical` and the section `slow` in the following analog model file because the `mymos` model is included in both.

```
library mos
  section typical
    model mymos type=n uo=600
  endsection typical
  section slow
    model mymos type=n uo=400
  endsection slow
```

4. If necessary, highlight a row in the table and click the up or down arrows to rearrange the order of the model files.

Files higher in the list are handled by the elaborator before files lower in the list.

5. When you finish adding analog model files, click *OK*.

## Deleting an Analog Model File

- To remove an analog model file, highlight the row, then click *Remove*.

If you want to remove multiple entries, you can use the *Shift* and *Control* keys to highlight multiple entries before you click *Remove*.

## Editing an Analog Model File

To edit the contents of a model file listed in the table,

- Highlight the row that lists the model file you want to edit, and click *Edit*.

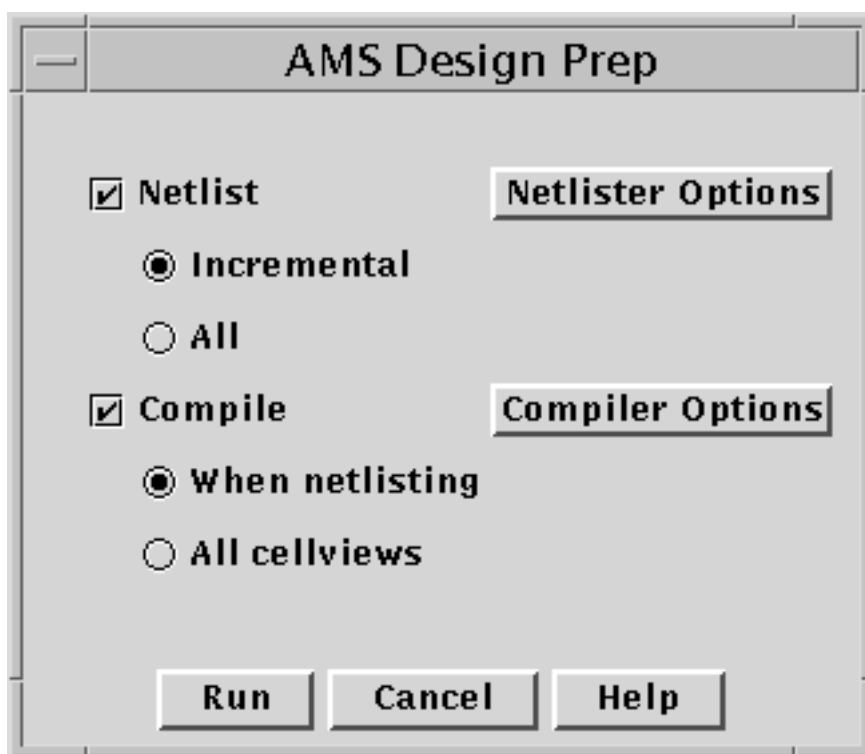
The model file opens in your selected editor.

## Running AMS Design Prep

To run AMS Design Prep,

1. Choose *AMS – Design Prep* from the Cadence hierarchy editor.

The AMS Design Prep window appears.



## Virtuoso AMS Environment User Guide

### Preparing a Design for Simulation

---

The settings in this window determine when netlisting and compilation occur as you work with the CDBA cellviews in your design hierarchy.

Setting	Behavior
<i>Netlist</i>	Enables netlisting
<i>Incremental</i>	Netlists CDBA cellviews in the hierarchy only if their HDL data is not synchronized with their CDBA data
<i>All</i>	Netlists all CDBA cellviews in the hierarchy, regardless of whether their HDL data is synchronized with their CDBA data
<i>Compile</i>	Enables compilation
<i>When netlisting</i>	Specifies that the <code>ncvlog</code> compiler is to create or update the cellview's compiled data only if the cellview is netlisted when you click <i>Run</i>
<i>All</i>	Specifies that, for each cellview used in the design, the <code>ncvlog</code> or <code>ncvhdl</code> compiler is to compile the cellview, whether or not the cell is netlisted in this run.

2. Set the appropriate options.

3. Click *Run*.

In accordance with your selections, AMS Design Prep performs some or all of the following tasks:

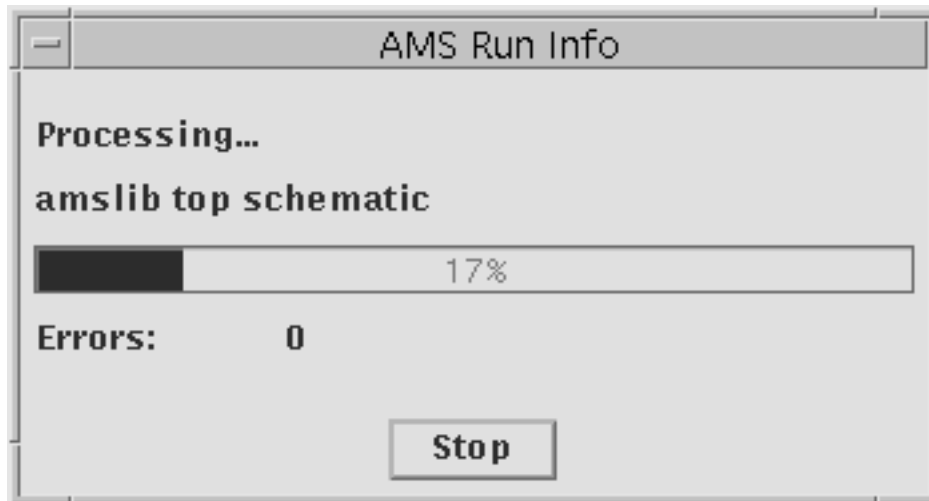
- ☐ Creates or updates netlist data
- ☐ Creates or updates compiled data
- ☐ Creates or updates the `cds_globals` module for global signals
- ☐ Declares or updates design variables
- ☐ Reports information, warnings, and errors in the log file and in the hierarchy editor message pane

## Virtuoso AMS Environment User Guide

### Preparing a Design for Simulation

---

While AMS Design Prep is running, the AMS Run Info progress indicator appears.



You can stop the AMS Design Prep run by clicking *Stop*.

After AMS Design Prep finishes, the AMS Design Prep - Summary window appears, displaying the following information:

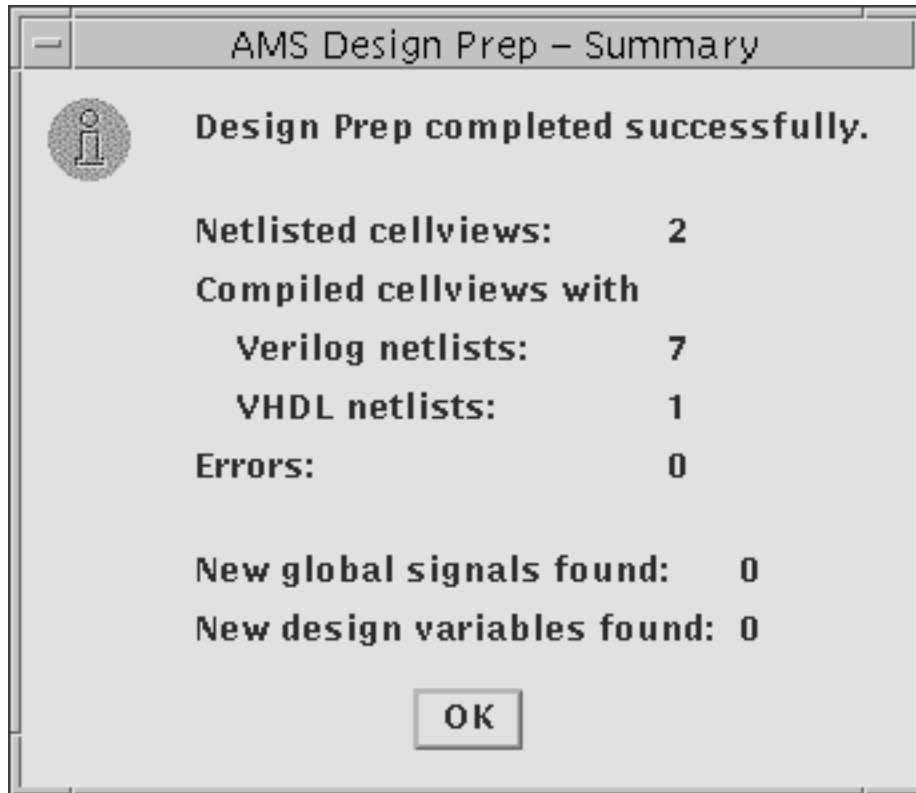
- Success or failure of the design preparation
- How many design units were netlisted
- How many errors were found in the design
- How many new global signals were found

## Virtuoso AMS Environment User Guide

### Preparing a Design for Simulation

---

- How many new design variables were found



AMS Design Prep detects whether you have generated or edited the `cds_globals` module. If you try to run AMS Design Prep when the data in the `cds_globals` module has been edited outside the hierarchy editor or has an error, AMS Design Prep issues a warning and asks for confirmation, as shown below.





## Virtuoso AMS Environment User Guide

### Preparing a Design for Simulation

---

## Messages and Log Files

When AMS Design Prep runs, it displays messages in the Cadence hierarchy editor window. For example, the hierarchy editor messages might look like this.

```
Netlisting amslib top schematic
Compiling verilog.vams in cellview amslib top schematic.
ncvlog: v3.150.(a1): (c) Copyright 1995 - 2000 Cadence Design Systems, Inc.
Netlisting amslib comparator schematic.
Compiling verilog.vams in cellview amslib comparator schematic.
ncvlog: v3.150.(a1): (c) Copyright 1995 - 2000 Cadence Design Systems, Inc.
Creating cds_globals module in amslib cds_globals top_config.
Compiling verilog.vams in cellview amslib cds_globals top_config.
ncvlog: v3.150.(a1): (c) Copyright 1995 - 2000 Cadence Design Systems, Inc.
Design preparation has completed successfully.
```

You can watch the hierarchy editor messages to see current information about the AMS Design Prep run. If errors are reported that you want to fix, you can stop the AMS Design Prep run by clicking *Stop* in the AMS Run Info progress indicator.

AMS Design Prep does not create its own log file. However, tools running under the control of AMS Design Prep do create log files. The AMS netlister creates a log file called, by default, `ams_direct.log`, and the NC Verilog and NC VHDL compilers create the `ncvlog.log` and `ncvhdl.log` files. You can view these log files easily by choosing *AMS – Netlister Log File* or *AMS – Simulator Log Files* from the hierarchy editor menu.

For example, the contents of a netlister (`ams_direct.log`) file might look like this.

```
@(#) $CDS: amsdirect version 4.4.6 04/05/2000 17:39 (machine) $
Copyright (c) 1999 Cadence Design Systems. All Rights Reserved.

Run date: Thu Apr 13 14:09:52 2000
Run time options used:
    -mpshost machine.Cadence.COM -mpssession sudeshna -pid 2443

Info: Processing ("amslib" "top" "schematic") ...
Info: Verilog-AMS netlist successfully written to
    /mnt3/sudeshna/eureka/alpha/AMS_lib/amsLib/top/schematic/verilog.vams.

Info: Found 0 errors and 0 warnings.

Info: Processing ("amslib" "comparator" "schematic") ...
Info: Verilog-AMS netlist successfully written to
    /mnt3/sudeshna/eureka/alpha/AMS_lib/amsLib/comparator/schematic/
verilog.vams.
Info: Found 0 errors and 0 warnings.
```

## How AMS Design Prep Handles Global Signals and Design Variables

The values of global signals and design variables must be available throughout a design, not just within a single module. AMS Design Prep enables this capability by collecting global signals and design variables into a module called `cds_globals`.

## Virtuoso AMS Environment User Guide

### Preparing a Design for Simulation

---

If access permissions are set so that AMS Design Prep cannot write to the `cds_globals` module, the following user interfaces are affected:

User interface	Effect of having a non-writable <code>cds_globals</code>
AMS Global Signals, reached by choosing <i>AMS – Global Signals</i> .	Buttons specific to this window become non-functional. You cannot add, remove, or change information.
AMS Design Variables, reached by choosing <i>AMS – Design Variables</i> .	Buttons specific to this window become non-functional. You cannot add, remove, or change information.
AMS Design Prep, reached by choosing <i>AMS – Design Prep</i> .	A message appears informing you that the <code>cds_globals</code> module cannot be updated with information during this run.

Note these restrictions:

- AMS Design Prep cannot write the `cds_globals` information to a TMP file.
- The creation and updating of the `cds_globals` module is not subject to design management (DM) control, even when access to other objects in the design is controlled by a DM tool.

If the `cds_globals` module cannot be created or is not writable, you can specify a different location for the information. Choose *AMS – Options – Global Design Data* and enter a writable location in the *Global Design Data Module* pane.

## The `cds_globals` Module

For some designs, such as those that combine HDL and CDBA cellviews, you need to understand the kinds of information placed in the `cds_globals` module, and what it means. Do not change the name of the `cds_globals` module because it is automatically specified for simulation. (If your design contains no design variables or global signals, AMS Design Prep creates the `cds_globals` module as an empty module.)

Normally, the `cds_globals` module is written to the file

`configLibName/cds_globals/configCellName_configView/verilog.vams`

(However, you can change the location of the file. For more information, see [“Setting Options for Global Design Data”](#) on page 211.)

For example, consider the following `cds_globals` module, which is located in a file named `amsLib/cds_globals/top_config/verilog.vams`. Notice the declarations of the global signals and the `dynamicparam` declaration of the design variable. For more information, see [“Global Signals”](#) on page 228 and [“Design Variables”](#) on page 229.

```
// Verilog-AMS cds_globals module for top-level cell:
//      amslib/top.
// Generated by AMS Design Prep.
// Cadence Design Systems, Inc.

`include "disciplines.vams"

module cds_globals;
// Global Signals
    electrical \vdd! ;
    electrical \vss! ;
    electrical \gnd! ;
    ground \gnd! ;

// Design Variables
dynamicparam real idc = 20u;
endmodule
```

It is not necessary for the discipline of every signal in the `cds_globals` module to be defined. As elsewhere in the design, the otherwise undefined disciplines are determined by the discipline resolution method that you use. For more information, see the [“Discipline Resolution Methods”](#) section in the “Mixed-Signal Aspects of Verilog-AMS” chapter, of the *Cadence Verilog-AMS Language Reference*.

### Creating Your Own `cds_globals` Module

You can create your own `cds_globals` module, overwriting the `cds_globals` module automatically created by AMS Design Prep. A `cds_globals` module created in this way is not overwritten by AMS Design Prep.

You might want to hand-create this module when

- You want a single or common module that is used by many designs instead of creating a `cds_globals` module for each configuration, which is what AMS Design Prep normally does. By creating a common module, you can avoid cluttering your library with cellviews that contain the same information.
- You want to declare global signals or design variables that are used in HDL cellviews but are not used in the schematic cellviews of your design.

Although Cadence does not recommend it, it is even possible to create many `cds_globals` modules, though each must be uniquely named. The AMS Designer flow only supports a single `cds_globals` module so if you have more than that, the modules must be added in the *Additional arguments* field of the *Elaborator* pane in the AMS Options window.

## Global Signals

A global signal is a signal that is available everywhere in the design hierarchy. AMS Design Prep collects the global signals from each CDBA cellview of the design and stores the collected information in the `cds_globals` module. AMS Design Prep also provides a user interface that allows you to examine, change, and add to the global signals in your design. For information on using the user interface, see [“Specifying Global Signals”](#) on page 213.

Unless you disable the automatic generation of the `cds_globals` module, AMS Design Prep regenerates the global signal information whenever global signals are added to or removed from the design.

## Using Global Signals in Verilog (Digital) and Verilog-AMS HDL Modules

AMS Design Prep does not collect global signals from HDL cellviews. Consequently, if you want global signals in your Verilog (digital) and Verilog-AMS HDL cellviews—VHDL-AMS design units do not support using global signals—to interconnect with global signals in the netlists translated from schematic cellviews, you must use global signals from the `cds_globals` module in your Verilog (digital) and Verilog-AMS HDL cellviews. You do this by using out-of-module references to the global signals.

For example, consider the following `cds_globals` module produced by AMS Design Prep.

```
// Verilog-AMS netlist generated by the AMS netlister
// Cadence Design Systems, Inc.
`include "disciplines.vams"
`include "constants.vams"
module cds_globals;
// Global Signals
    electrical \vdd! ;
    electrical \vss! ;
    electrical \gnd! ;
    ground \gnd! ;
    wire signal1;
    ground signal2;
// Design Variables
    dynamicparam real idc = 20u;
endmodule
```

You can refer to these global signals from within a Verilog-AMS module, for example, by using a hierarchical name such as `cds_globals.signal1`.

There are some complications produced by the fact that not all characters that are legal in CDBA modules are legal in Verilog-AMS modules. For example, a net name of `vdd!`, though legal in CDBA modules, is not allowed in Verilog-AMS and must be escaped. That is why, in the previous module, all of the signals defined with the `electrical` discipline are escaped.

You must use the escaped names if you want to refer to these nets from within an HDL module. For example, you use a hierarchical name of `"cds_globals.\vdd! "` to refer to a net that in the CDBA module is simply `vdd!` (Note the space that terminates the escaped name).

### **The Default Global Signal of an Inherited Signal Expression**

AMS Design Prep treats the default global signal of an inherited signal expression as a regular global signal in CDBA. The signal is also stored in the `cds_globals` module.

## **Design Variables**

A design variable is a global variable that is available everywhere in the design hierarchy. AMS Design Prep collects the design variables found in each CDBA cellview of the design and declares them as dynamic parameters in the `cds_globals` module. AMS Design Prep also provides a user interface that allows you to examine, change, and add to the design variables in your design. For information on using the user interface, see [“Specifying Design Variables”](#) on page 216.

**Note:** The VHDL-AMS language does not support out-of-module references, so you cannot set design variables within VHDL-AMS design units, nor can you, from within a VHDL-AMS design unit, directly access design variables used in other modules or design units.

A design variable can be referenced in the values of instance properties or AEL expressions. (Note, however, that AMS Design Prep does not find design variables that are used in NLP expressions.) When converted to Verilog-AMS, a design variable can be referenced in the value of an expression via an out-of-module reference. For example, assume that `sheetRes` is a design variable declared in the `cds_globals` module with an expression such as

```
dynamicparam real sheetRes = 1.3 ;
```

Then you can refer to that value in a Verilog-AMS module with a statement such as

```
#(.r(cds_globals.sheetRes * area))
```

# **Virtuoso AMS Environment User Guide**

## **Preparing a Design for Simulation**

---

---

## Elaborating, Simulating, and Plotting Results

---

This chapter contains the following sections:

- [Specifying the Behavior of the Elaborator, Simulator, and Waveform Viewer](#) on page 232
- [Creating Probes](#) on page 277
- [Elaborating and Simulating](#) on page 286
- [Viewing Messages](#) on page 289
- [Plotting Waveforms After Simulation Ends](#) on page 290

## Specifying the Behavior of the Elaborator, Simulator, and Waveform Viewer

The elaborator, simulator, and waveform viewer support many different options, which allow you to tailor the behavior of the tools to your needs. All of the supported options can be set on the command line when you start the tools. In addition, you can set the most commonly used options by making selections in the graphical interfaces provided by the AMS environment. The following sections describe how to use the graphical interfaces to set options, elaborate designs, and simulate designs. For information about using the command line controls, see the [Virtuoso AMS Simulator User Guide](#).

For more information, see the following sections.

- [“Setting Elaborator Options”](#) on page 232
- [“Setting Simulator Options”](#) on page 247
- [“Setting Waveform Selection Options”](#) on page 274

### Setting Elaborator Options

You can access the option settings for the elaborator from the Cadence® hierarchy editor.

1. From the hierarchy editor, choose *AMS – Options*.

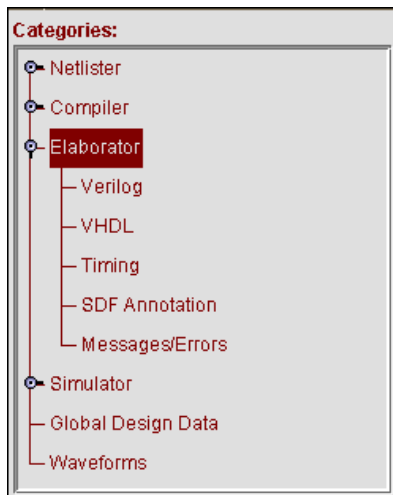
If the *AMS* menu entry is not visible, follow the instructions in [“Preparing to Use AMS Designer from the Hierarchy Editor”](#) on page 62.

The AMS Options window appears.

2. Expand the *Elaborator* category to find the options you want to change.



The *Elaborator* category has the following subcategories.



**3. Highlight the category that you want to change.**

The corresponding pane appears. For information about the fields in each pane, see the following cross-references:

**For information about this category**

**See**

*Elaborator*

[“Specifying General Options for the Elaborator” on page 233](#)

*Verilog*

[“Specifying Verilog Options for the Elaborator” on page 235](#)

*VHDL*

[“Specifying VHDL Options for the Elaborator” on page 237](#)

*Timing*

[“Specifying Timing Options for the Elaborator” on page 239](#)

*SDF Annotation*

[“Specifying SDF Annotation Options for the Elaborator” on page 243](#)

*Messages/Errors*

[“Specifying Message and Error Options for the Elaborator” on page 246](#)

**Specifying General Options for the Elaborator**

1. Choose *AMS – Options* from the hierarchy editor menu.

## Virtuoso AMS Environment User Guide

### Elaborating, Simulating, and Plotting Results

---

If the *AMS* menu entry is not visible, follow the instructions in “[Preparing to Use AMS Designer from the Hierarchy Editor](#)” on page 62.

2. Highlight *Elaborator* to open the AMS Options window to the *Elaborator* pane.

**Elaborator**

Maximum number of errors: 50

Log file: Overwrite log file ▼

☒ Update design units if needed

☐ Ignore source file timestamps when using -update

Default discipline: logic

☐ Use detailed discipline resolution

Default timescale: 1ns/1ns

☒ Allow undefined parameters

☐ Additional arguments:

3. Fill in and select fields as necessary.

The following table briefly describes the fields. For more information, see the “[ncelab Command Syntax and Options](#)” section in the “Elaborating” chapter of the *Virtuoso AMS Simulator User Guide*.

Field	Corresponding ncelab Option	Effect
<i>Maximum number of errors</i>	-errormax	Specifies the maximum number of errors to process.
<i>Log file</i>	-append_log, -nolog	Determines whether a log file is to be created, and, if so, whether the new information is to overwrite any existing log or be appended.
<i>Update design units if needed</i>	-update	Automatically recompiles any out-of-date design units and then re-elaborates the design.
<i>Ignore source file timestamps when using -update</i>	-nosource	Turns off source file timestamp checking when the <i>Update design units if needed</i> field is turned on.

## Virtuoso AMS Environment User Guide

### Elaborating, Simulating, and Plotting Results

---

Field	Corresponding ncelab Option	Effect
<i>Default discipline</i>	-discipline	Specifies the discipline of discrete nets for which a discipline is otherwise undefined.
<i>Use detailed discipline resolution</i>	-dresolution	Specifies the kind of discipline resolution to be used.
<i>Default timescale</i>	-timescale	Sets the default timescale for Verilog modules that do not have a timescale set.
<i>Allow undefined parameters</i>	-noparamerr	Tells the elaborator to allow undeclared parameters to be overridden.
<i>Additional arguments</i>		To pass the elaborator any arguments that cannot be generated by the AMS Options GUI, you can type the options into this field.

4. Click *OK* to save your changes.

### Specifying Verilog Options for the Elaborator

To specify Verilog options for the elaborator,

1. From the Cadence hierarchy editor, choose *AMS – Options*.

2. In the AMS Options window, choose *Elaborator – Verilog* to open the AMS Options window to the *Verilog* pane.

3. Select and fill in fields as necessary.

The following table briefly describes the fields. For more information, see the “[ncelab Command Syntax and Options](#)” section in the “Elaborating” chapter of the *Virtuoso AMS Simulator User Guide*.

Field	Corresponding ncelab Option	Effect
<i>Access visibility</i>	-access	Sets the visibility access for all objects in the design.
<i>Use an access file</i>	-afile	Specifies an access file.
<i>Generate an access file</i>	-genafilename	Generate an access file for PLI and Tcl.

## Virtuoso AMS Environment User Guide

### Elaborating, Simulating, and Plotting Results

---

Field	Corresponding ncelab Option	Effect
<i>OMI checking level</i>	-omicheckinglevel	Specifies the OMI checking level to use.
<i>Run IEEE 1364 lint checker</i>	-ieee1364	Checks for compatibility with the IEEE 1364 standard.
<i>Expand all vector nets</i>	-expand	Expands all vector nets.
<i>PLI Options</i>		
<i>Dynamically load VPI libraries</i>	-loadvpi	Dynamically loads the specified VPI application.
<i>Dynamically load PLI libraries</i>	-loadpli1	Dynamically loads the specified PLI 1.0 application.
<i>Enable delay annotation at simulation time</i>	-anno_simtime	Enables the use of PLI/VPI routines that modify delays at simulation time.
<i>Suppress VPI/PLI warning and error messages</i>	-plinowarn	Disables printing of PLI warning and error messages.
<i>Suppress VPI/PLI messages caused by optimizations</i>	-plinoptwarn	Prints a warning message only the first time that a PLI read, write, or connectivity access violation is detected.

---

4. When you are done changing options, click *OK* to save your changes.

### Specifying VHDL Options for the Elaborator

To specify VHDL options for the elaborator,

1. From the Cadence hierarchy editor, choose *AMS – Options – Elaborator*.

2. In the AMS Options window, choose *Elaborator – VHDL* to open the AMS Options window to the *VHDL* pane.

**VHDL**

☐ Enable VHDL 93 features

☒ Use 5X configs for VHDL

☐ Preserve resolution functions on signals with only one driver

☐ Enable relaxed interpretation of VHDL LRM

☐ Allow mixed-case, escaped identifiers in VHDL

☐ Specify value for top level generic:

Generic Name	Generic Value

Add

Remove

3. Select and fill in fields as necessary.

The following table briefly describes the fields. For more information, see the “[ncelab Command Syntax and Options](#)” section in the “Elaborating” chapter of the *Virtuoso AMS Simulator User Guide*.

Field	Corresponding ncelab Option	Effect
<i>Enable VHDL 93 features</i>	-v93	Enables VHDL-93 features.
<i>Use 5X configs for VHDL</i>	-use5x4vhdl	Specifies that configurations apply to VHDL as well as Verilog-AMS and that configurations take precedence over VHDL default binding and other searches.
<i>Preserve resolution functions on signals with only one driver</i>	-preserve	Preserves resolution functions on signals with only one driver.

Field	Corresponding ncelab Option	Effect
<i>Enable relaxed interpretation of VHDL LRM</i>	-relax	Makes design units visible for default binding when the design units exist in a library that has not been made visible with a <code>LIBRARY</code> declaration in the VHDL source and when the design units do not exist in the library that has been defined as the work library.
<i>Allow mixed-case, escaped identifiers in VHDL</i>	-mixesc	Enables elaboration of mixed-language designs in which escaped names are used for VHDL or VHDL-AMS entities imported into Verilog or Verilog-AMS designs. If you follow the AMS Designer guideline that recommends using lower case identifiers, this option is unnecessary. For more information, see “ <a href="#">ncelabMixEsc</a> ” on page 439.
<i>Specify value for top level generic</i>	-generic	Specifies a value for a top-level generic.

4. When you are done changing options, click *OK* to save your changes.

### Specifying Timing Options for the Elaborator

To specify timing options for the elaborator,

1. From the Cadence hierarchy editor, choose *AMS – Options*.

2. In the AMS Options window, choose *Elaborator – Timing* to open the AMS Options window to the *Timing* pane.

**Timing**

☐ Disable timing checks

☐ Suppress timing check warnings

☐ Print convergence warnings for negative time checks

**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 canceled events to a state

☐ Enable PATHPULSE \$ specparams

☐ Pulse reject limit (path and interconnect): 100 %

☐ Pulse error limit (path and interconnect): 100 %

☐ Pulse reject limit (interconnect only): 100 %

☐ Pulse error limit (interconnect only): 100 %

☐ Relax tcheck data limit by: 0 %

☐ Relax tcheck reference limit by: 0 %

**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 interconnect delays

3. Select and fill in fields as necessary.

The following table briefly describes the fields. For more information, see the “[ncelab Command Syntax and Options](#)” section in the “Elaborating” chapter of the *Virtuoso AMS Simulator User Guide*.

Field	Corresponding ncelab Option	Effect
<i>Disable timing checks</i>	-notimingchecks	Turns off the execution of timing checks.
<i>Suppress timing check warnings</i>	-no_tchk_msg	Turns off the display of timing check warning messages.



## Virtuoso AMS Environment User Guide

### Elaborating, Simulating, and Plotting Results

---

Field	Corresponding ncelab Option	Effect
<i>Print convergence warnings for negative time checks</i>	-ntc_warn	Print convergence warnings for negative timing checks for both Verilog and VITAL if delays cannot be calculated given the current limit values.
<i>Verilog Timing Options</i>		
<i>Delay mode</i>	-delay_mode	Specifies the delay mode to be used for digital Verilog-AMS portions of the hierarchy.
<i>Disable enhanced timing features</i>	-disable_enht	Turns off the enhanced timing features enabled by using special properties in a specify block.
<i>Disallow negative values in checks</i>	-noneg_tchk	Disallows negative values in \$setuphold and \$recrem timing checks in the Verilog description and in SETUPHOLD and RECREM timing checks in SDF annotation.
<i>Enable PATHPULSE\$ specparams</i>	-pathpulse	Enables PATHPULSE\$ specparams, which are used to set module path pulse control on a specific module or on specific paths within modules.
<i>Pulse reject limit (path and interconnect)</i>	-pulse_r	Sets the percentage of delay for the pulse reject limit for both module paths and interconnect.
<i>Pulse reject limit (interconnect only)</i>	-pulse_int_r	Sets the percentage of delay for the pulse reject limit for interconnect only.

## Virtuoso AMS Environment User Guide

### Elaborating, Simulating, and Plotting Results

---

Field	Corresponding ncelab Option	Effect
<i>Relax tcheck data limit by</i>	-extend_tcheck_data_limit	Extends the violation regions established by a pair of \$setuphold or \$recrem timing checks with negative values in which the timing checks contain two different constraints for posedge and negedge of data with respect to the same reference signal and in which the violation regions do not overlap.
<i>Error pulse filtering</i>	-epulse_ondetect, -epulse_onevent	Uses On-Detect or On-Event filtering of error pulses.
<i>Ignore notifiers in timing checks</i>	-nonotifier	Tells the elaborator to ignore notifiers in timing checks.
<i>Filter canceled events to e state</i>	-epulse_neg	Filters cancelled events (negative pulses) to the e state.
<i>Pulse error limit (path and interconnect)</i>	-pulse_e	Sets the percentage of delay for the pulse error limit for both module paths and interconnect.
<i>Pulse error limit (interconnect only)</i>	-pulse_int_e	Sets the percentage of delay for the pulse error limit for interconnect only.
<i>Relax tcheck reference limit by</i>	-extend_tcheck_reference_limit	Extends the violation regions established by a pair of \$setuphold or \$recrem timing checks with negative values in which the timing checks contain two different constraints for posedge and negedge of data with respect to the same reference signal and in which the violation regions do not overlap.

## Virtuoso AMS Environment User Guide

### Elaborating, Simulating, and Plotting Results

---

Field	Corresponding ncelab Option	Effect
<i>VHDL Timing Options</i>		
<i>Disable VITAL acceleration</i>	-novitalaccl	Suppresses the acceleration of VITAL level 1 compliant cells.
<i>Disable X generation in VITAL timing checks</i>	-no_tchk_xgen	Turns off X generation in accelerated VITAL timing checks.
<i>Disable X generation in VITAL path delays</i>	-no_vpd_xgen	Turns off X generation in accelerated VITAL pathdelay procedures.
<i>Suppress VITAL path delay warnings</i>	-no_vpd_msg	Turns off warning messages from accelerated VITAL path delay procedures.
<i>Ignore interconnect delays</i>	-noipd	Turns off recognition of input path delays in a VITAL level 1 cell and uses the non-delayed input signals directly.

4. When you are done changing options, click *OK* to save your changes.

### Specifying SDF Annotation Options for the Elaborator

To specify SDF Annotation options for the elaborator,

1. From the Cadence hierarchy editor, choose *AMS – Options*.

2. In the AMS Options window, choose *Elaborator – SDF Annotation* to open the AMS Options window to the *SDF Annotation* pane.

3. Select and fill in fields as necessary.

The following table briefly describes the fields. For more information, see the “[ncelab Command Syntax and Options](#)” section in the “Elaborating” chapter of the *Virtuoso AMS Simulator User Guide*.

Field	Corresponding ncelab Option	Effect
<i>Use SDF command file</i>	<code>-sdf_cmd_file</code>	Specifies an SDF command file to control SDF annotation.

## Virtuoso AMS Environment User Guide

### Elaborating, Simulating, and Plotting Results

---

Field	Corresponding ncelab Option	Effect
<i>Delay type</i>	-mindelays, -typdelays, -maxdelays	Applies the minimum, typical, or maximum delay value from a timing triplet in the form min:typ:max in the SDF file while annotating to Verilog or to VITAL.
<i>Suppress SDF warnings</i>	-sdf_no_warnings	Suppresses warning messages from the SDF annotator.
<i>Allow unique delays for each source-delay path</i>	-intermod_path	Enables multisource and transport delay behavior with pulse control for interconnect delays.
<i>Include detailed information in SDF log file</i>	-sdf_verbose	Includes detailed information in the SDF log file.
<i>Round precision of timing in SDF file</i>	-sdf_precision	SDF data is modified to this precision.
<b>Verilog SDF Options</b>		
<i>Suppress SDF command file information messages</i>	-no sdfa_header	Turns off the printing of elaborator information messages that display information contained in the SDF command file.
<i>Disable celltype validation between SDF annotator and Verilog description</i>	-sdf_nocheck_celltype	Disables celltype validation between the SDF annotator and the Verilog description.
<i>Allow SDF worst-case rounding</i>	-sdf_worstcase_rounding	For timing values in the SDF file, truncates the min value, rounds the typ value, and rounds up the max value.
<i>Do not run \$sdf_annotate tasks automatically</i>	-noautosdf	Turns off automatic SDF annotation.

Field	Corresponding ncelab Option	Effect
<i>VHDL Timing Options</i>		
<i>Select delay value for VitalInterconnect Delays</i>	-vipdmin, -vipdmax	Selects the typical, minimum, or maximum delay value for VitalInterconnectDelays.

4. When you are done changing options, click *OK* to save your changes.

### Specifying Message and Error Options for the Elaborator

To specify message and error options for the elaborator,

1. From the Cadence hierarchy editor, choose *AMS – Options*.
2. In the AMS Options window, choose *Elaborator – Messages/Errors* to open the AMS Options window to the *Elaborator Messages/Errors* pane.

The screenshot shows a window titled "Elaborator Messages/Errors" with a light gray background. It contains several checkboxes and a text input field, all with red text. The options are:

- ☐ Print informational messages
- ☒ Display runtime status
- ☐ Suppress all warnings
- ☒ Suppress specific warnings:
- ☐ Suppress output to screen
- ☐ Suppress copyright information
- ☐ Print messages about resolving instances (Verilog only)
- ☐ Enable code coverage for entire design (digital only)

3. Select and fill in fields as necessary.

## Virtuoso AMS Environment User Guide

### Elaborating, Simulating, and Plotting Results

---

The following table briefly describes the fields. For more information, see the [“ncelab Command Syntax and Options”](#) section in the “Elaborating” chapter of the *Virtuoso AMS Simulator User Guide*.

---

Field	Corresponding ncelab Option	Effect
<i>Print informational messages</i>	-messages	Prints informative messages during elaboration.
<i>Display runtime status</i>	-status	Prints statistics on memory and CPU usage after elaboration.
<i>Suppress all warnings</i>	-neverwarn	Disables printing of all warning messages.
<i>Suppress specific warnings</i>	-nowarn	Disables printing of the specified warning message.
<i>Suppress output to screen</i>	-nostdout	Suppresses the printing of most output to the screen.
<i>Suppress copyright information</i>	-nocopyright	Suppresses printing of the copyright banner.
<i>Print messages about resolving instances (Verilog only)</i>	-libverbose	Displays messages about module and UDP instantiations.
<i>Enable code coverage for entire design (digital only)</i>	-coverage	Enables coverage instrumentation.

---

4. When you are done changing options, click *OK* to save your changes.

## Setting Simulator Options

AMS Designer provides many options for the simulator, that allow you to tailor the simulator behavior for your needs. You can access the option settings for the simulator from the Cadence hierarchy editor.

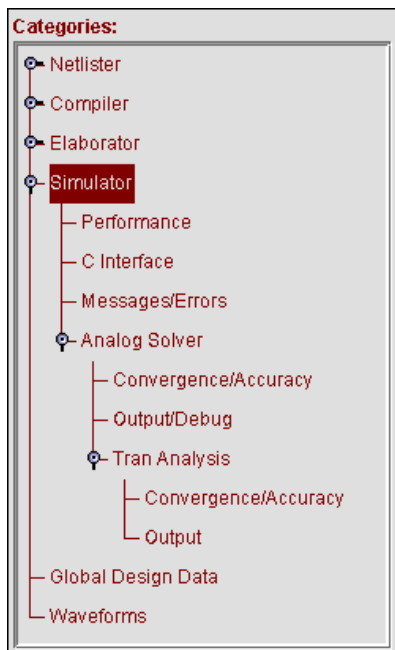
1. From the hierarchy editor, choose *AMS – Options*.

If the *AMS* menu entry is not visible, follow the instructions in [“Specifying the Behavior of the Netlister and Compilers”](#) on page 74.

The AMS Options window appears.

2. Expand the *Simulator* category as necessary to find the options you want to change.

When fully expanded, the *Simulator* category displays the following subcategories.



3. Highlight the category that you want to change.

The corresponding pane appears. For information about the fields in each pane, see the following cross-references:

**For information about this category**

**See**

*Simulator*

[“Specifying Basic Options for the Simulator” on page 249](#)

*Performance*

[“Specifying Performance Options for the Simulator” on page 251](#)

*C Interface*

[“Specifying C Interface Options for the Simulator” on page 255](#)

*Messages/Errors*

[“Specifying Message and Error Options for the Simulator” on page 256](#)

*Analog Solver*

[“Specifying General Options for the Analog Solver” on page 258](#)



**For information about this category**

**See**

*Convergence/Accuracy*

[“Specifying Convergence and Accuracy Options for the Analog Solver”](#) on page 260

*Output/Debug*

[“Specifying Output and Debug Options for the Analog Solver”](#) on page 265

*Tran Analysis*

[“Specifying a Transient Analysis for the Analog Solver”](#) on page 268

*Convergence/Accuracy*

[“Specifying Convergence and Accuracy Options for a Transient Analysis”](#) on page 269

*Output*

[“Specifying Output and Debug Options for a Transient Analysis”](#) on page 273

## **Specifying Basic Options for the Simulator**

To specify basic options for the simulator,.

1. From the hierarchy editor, choose *AMS – Options*.

If the *AMS* menu entry is not visible, follow the instructions in [“Specifying the Behavior of the Netlister and Compilers”](#) on page 74.

The *AMS Options* window appears.

2. Highlight the *Simulator* category.

The *Simulator* pane opens.

**Simulator**

Maximum number of errors: 50

Log file: Overwrite log file ▼

☐ Tcl input script: /usr1/cds11752/alpha6/vhdltestdir/SAR\_A2D/demo.tcl

Browse Edit

☒ Update if needed ☐ Ignore source file timestamps when using -update

☐ Additional arguments:

**3.** Fill in the fields as necessary.

The following table briefly describes the fields and tells you where to go for more information. All of the cross-references in the table are to the “[Simulating](#)” chapter of the *Virtuoso AMS Simulator User Guide*.

Field	Corresponding ncsim Option	Effect
<i>Maximum number of errors</i>	-errormax	Specifies the maximum number of errors to process. Errors caused by Tcl command files or by interactive Tcl commands do not count toward the limit.
<i>Log file</i>	-append_log, -nolog	Determines whether a log file is to be created, and, if so, whether the new information is to overwrite any existing log or be appended.
<i>Tcl input script</i>	-input	Specifies a script file to run at the beginning of the simulation.
<i>Update if needed</i>	-update	Recompiles out-of-date design units as necessary.
<i>Ignore source file timestamps when using -update</i>	-nosource	Turns off source file timestamp checking when using the -update option.

## Virtuoso AMS Environment User Guide

### Elaborating, Simulating, and Plotting Results

---

Field	Corresponding ncsim Option	Effect
<i>Additional arguments</i>		To pass to the simulator arguments that are not available in the GUI, type the arguments into this field.

4. Click *OK* to save your changes.

### Specifying Performance Options for the Simulator

To specify performance options for the simulator,

1. From the Cadence hierarchy editor, choose *AMS – Options*.

## Virtuoso AMS Environment User Guide

### Elaborating, Simulating, and Plotting Results

---

2. In the AMS Options window, choose *Simulator – Performance* to open the AMS Options window to the *Performance* pane.



3. Select fields as necessary.

The following table briefly describes the fields and tells you the corresponding option or option card. For additional information about `ncsim` options, see the “[ncsim Command Syntax and Options](#)” section, in the “Simulating” chapter of the *Virtuoso AMS Simulator User Guide*. For additional information about the `mos_method` options card,

## Virtuoso AMS Environment User Guide

### Elaborating, Simulating, and Plotting Results

---

see the *“Immediate Set Options (options)”* section, in the “Specifying Controls for the Analog Solver” chapter of the *Virtuoso AMS Simulator User Guide*.

Field	Corresponding Option	Effect
<i>Use reduced memory image size</i>	<code>ncsim -redmem</code>	Turns off loading of intermediate objects generated by the compiler.
<i>Do not buffer output</i>	<code>ncsim -unbuffered</code>	Bypasses the file I/O buffer so that data displays immediately.
<i>OMI checking level</i>	<code>ncsim -omicheckinglevel</code>	Specifies the OMI checking level to use.
<i>Generate runtime profile</i>	<code>ncsim -profile</code>	Generates a run time profile of the design. The profiling information is written to the <code>ncprof.out</code> file only after the simulator session is exited. Depending on the simulator you are using, this command might be restricted to digital only.
<i>Allow profiling of threaded processes (digital only)</i>	<code>ncsim -profthread</code>	Allows threaded processes to be profiled.

## Virtuoso AMS Environment User Guide

### Elaborating, Simulating, and Plotting Results

---

Field	Corresponding Option	Effect
<i>Model evaluation</i>		<p>When checked, writes the selected <i>mos_method</i> option, as described for <i>Standard</i> or <i>Accelerated (table model)</i>. When unchecked, writes nothing.</p> <p>This field is disabled when the <i>Use simulation control file</i> field, in the <i>Analog Solver</i> pane, is turned on.</p> <p>(This field appears only if the AMS simulator you are running supports it.)</p>
<i>Standard</i>	<p>The analog simulation control ( <i>.scs</i>) file option card:</p> <pre>amsOptions options + mos_method = s</pre>	Instructs the simulator not to use table models for any instances.
<i>Accelerated (table model)</i>	<p>The analog simulation control ( <i>.scs</i>) file card:</p> <pre>amsOptions options + mos_method = a</pre>	Instructs the simulator to use table models whenever possible. This global option applies to the entire simulated design. You can override this instruction on specific model cards by setting <i>mos_method = s</i> as an option on those cards.
<i>Table model grid size</i>	<p>The analog simulation control ( <i>.scs</i>) file option card:</p> <pre>amsOptions options + mos_vres = value</pre>	Specifies the voltage increment for the mosfet table model interpolation grid. For more information, see <a href="#">“scmosvres”</a> on page 506.
<i>Node reduction threshold</i>	<p>The analog simulation control ( <i>.scs</i>) file option card:</p> <pre>amsOptions options + maxrsd = value</pre>	Specifies the threshold below which parasitic node reduction occurs.

Field	Corresponding Option	Effect
<i>Speed dial</i>	The analog simulation control ( <i>.scs</i> ) file option card: amsOptions options + speed = <i>value</i>	Establishes the tradeoff between simulation performance and accuracy.

4. When you are done changing options, click *OK* to save your changes.

### Specifying C Interface Options for the Simulator

To specify C interface options for the simulator,

1. From the Cadence hierarchy editor, choose *AMS – Options*.
2. In the AMS Options window, choose *Simulator – C Interface* to open the AMS Options window to the *C Interface* pane.

**C Interface**

**Verilog Only**

Dynamically load VPI libraries:

[Text Field] [Add] [Remove]

[List Box] [List Box]

☐ Suppress VPI/PLI warning and error messages

☐ Suppress VPI/PLI messages caused by optimization

**VHDL Only**

☐ Disable constraint checking in VDA applications

3. Fill in and select fields as necessary.

## Virtuoso AMS Environment User Guide

### Elaborating, Simulating, and Plotting Results

---

The following table briefly describes the fields. For additional information, see the “[ncsim Command Syntax and Options](#)” section, in the “Simulating” chapter of the *Virtuoso AMS Simulator User Guide*.

---

Field	Corresponding ncsim Option	Effect
<i>Verilog Only</i>		
<i>Dynamically load VPI libraries</i>	<code>-loadvpi</code>	Dynamically loads a VPI application.
<i>Suppress VPI/PLI warning and error messages</i>	<code>-plinowarn</code>	Disables printing of PLI warning and error messages.
<i>Suppress VPI/PLI messages caused by optimization</i>	<code>-plinoptwarn</code>	Prints a warning message only the first time that a PLI read, write, or connectivity access violation is detected.
<i>VHDL Only</i>		
<i>Disable constraint checking in VDA applications</i>	<code>-nocifcheck</code>	Disables constraint checking in VHDL Design Access (VDA) functions, for increased performance.

---

4. When you are done changing options, click *OK* to save your changes.

### Specifying Message and Error Options for the Simulator

To specify message and error options for the simulator,

1. From the Cadence hierarchy editor, choose *AMS – Options*.

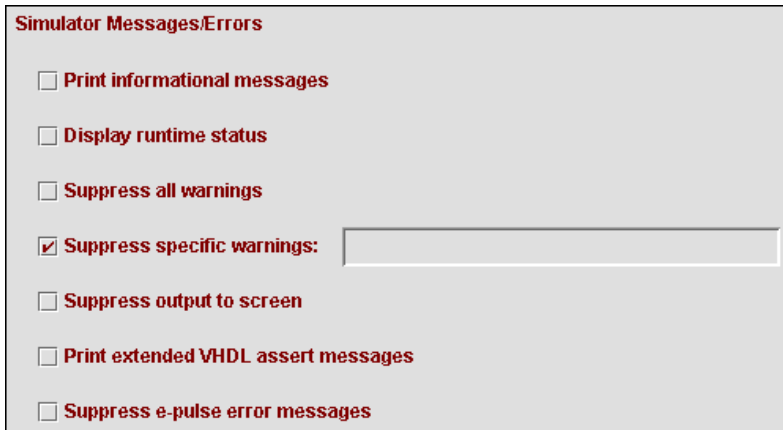


## Virtuoso AMS Environment User Guide

### Elaborating, Simulating, and Plotting Results

---

2. In the AMS Options window, choose *Simulator – Messages/Errors* to open the AMS Options window to the *Simulator Messages/Errors* pane.



**Simulator Messages/Errors**

☐ Print informational messages

☐ Display runtime status

☐ Suppress all warnings

☒ Suppress specific warnings:

☐ Suppress output to screen

☐ Print extended VHDL assert messages

☐ Suppress e-pulse error messages

3. Fill in and select fields as necessary.

The following table briefly describes the fields and tells you where to go for more information. For additional information, see the [“ncsim Command Syntax and Options”](#) section, in the “Simulating” chapter of the *Virtuoso AMS Simulator User Guide*.

Field	Corresponding ncsim Option	Effect
<i>Print informational messages</i>	-messages	Prints informative messages during simulation.
<i>Display runtime status</i>	-status	Prints statistics on memory and CPU usage after simulation.
<i>Suppress all warnings</i>	-neverwarn	Disables printing of all warning messages.
<i>Suppress specific warnings</i>	-nowarn	Disables printing of the specified warning message.
<i>Suppress output to screen</i>	-nostdout	Suppresses the printing of most output to the screen.
<i>Print extended VHDL assert messages</i>	-extassertmsg	Prints extended assert message Information.
<i>Suppress e-pulse error messages</i>	-epulse_no_msg	Suppresses e-pulse error messages.

4. When you are done changing options, click *OK* to save your changes.

## Specifying General Options for the Analog Solver

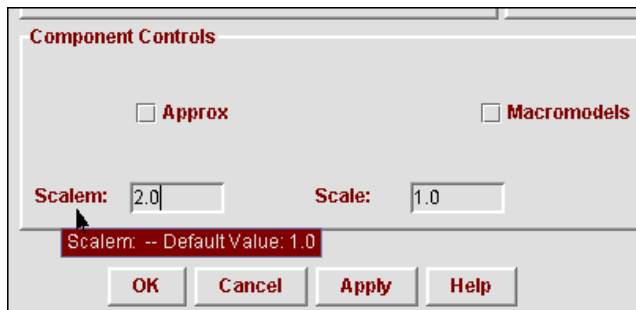
In the analog solver panes, you can specify many options that determine how the analog simulation proceeds.

### When Analog Solver Options Are Written to the Analog Control File

If you specify, for an analog solver option, a value that is not the simulator default, AMS Designer writes the option and value to the analog control file, which controls the simulation. However, when a value in the analog solver panes is the same as the corresponding simulator default value, AMS Designer does not write the option to the analog control file. In this situation, the simulator uses the corresponding value set in the model files, if there is such a value. If there is no corresponding value in the model files, the simulator uses the simulator default value. For more information about how the AMS simulator determines what value to use for options, see the *“Immediate Set Options (options)”* section in Chapter 8, of the *Virtuoso AMS Simulator User Guide*.

### Popping Up Information About Analog Solver Options

Holding the cursor over a label in the analog solver panes pops up information about the default value for that option. For example, in the following screen capture, the pop-up contains information about the default value of the option.



## Specifying Basic Information for the Analog Solver

To specify basic information for the analog solver,

1. From the Cadence hierarchy editor, choose *AMS – Options*.

2. In the AMS Options window, choose *Simulator – Analog Solver* to display the *Analog Solver* pane.

The screenshot shows the 'Analog Solver' pane with three sections:

- Use simulation control file:** An unchecked checkbox. Below it is a text field containing the path `/usr1/cds11752/alpha6/vhdltestdir/SAR_A2D/tutorial2_run/top.scs`. Below the text field are 'Browse' and 'Edit' buttons.
- Include simulation control file:** A checked checkbox. Below it is an empty text field. Below the text field are 'Browse' and 'Edit' buttons.
- Additional options:** A checked checkbox. Below it is an empty text field.

3. Choose how the analog solver is to be controlled.

- ☐ If you want the analog solver to use an existing simulation control file,
  - a. Check the box next to *Use simulation control file*.
  - b. Specify the control file in the *Use simulation control file* field.

The path can contain shell environment variables.

You can type the fully qualified path and name, or click *Browse* to locate the file that you want to use. (If you do not have an existing file, you can open a template by typing a name and clicking *Edit*.)

You can also type in a relative path, which is evaluated relative to the run directory, or a simple file name to refer to a file that is located in the run directory.

The simulation control file contains commands that tell the analog solver how to simulate the design. For more information, see the [“Specifying Controls for the Analog Solver”](#) chapter, in the *Virtuoso AMS Simulator User Guide*.

- ☐ If you want to use the AMS Options GUI to generate a new simulation control file,
  - a. Ensure that the box next to *Use simulation control file* is not checked.
  - b. Use the *Convergence/Accuracy*, *Output/Debug*, and *Tran Analysis* subcategories of the Analog Solver category to specify the behavior of the simulator.

If you use this approach, you must specify a stop time, at least. For guidance, see [“Specifying a Transient Analysis for the Analog Solver”](#) on page 268.

4. (Optional) Type the name of a simulation control file into the *Include simulation control file* field.

The path can contain shell environment variables.

The specified simulation control file is included in the simulation control file generated from the options you specify in the GUI. For example, you type

```
~/fpga.scs
```

into the *Include simulation control file* field. When you click *OK*, AMS Designer checks that the file exists, and if it does, includes it in the generated simulation control file with a statement like this.

```
include "/usr1/mnt4/lorenp/fpga.scs"
```

5. (Optional) Type into the *Additional options* field any options that you want to append to the end of the options card in the simulation control file.

For example, you type

```
rawfile = "/hm/kat/amsAnalysis"
```

into the *Additional options* field. AMS Designer adds the value to the end of the options card, like this.

```
amsOptions options  
+ gmin_check = all  
+ inventory = detailed  
+ rawfile = "/hm/kat/amsAnalysis"
```

6. When you finish setting options, click *OK* to save your changes.

## **Specifying Convergence and Accuracy Options for the Analog Solver**

You can access the convergence and accuracy option settings for the analog solver from the Cadence hierarchy editor.

## Virtuoso AMS Environment User Guide

### Elaborating, Simulating, and Plotting Results

---

1. In the AMS Options window, choose *Simulator – Analog Solver – Convergence/Accuracy* to open the AMS Options window to the *Convergence/Accuracy* pane.

The screenshot shows the 'Convergence/Accuracy' pane of the AMS Options window. It is divided into several sections: 'Tolerances' with fields for Reltol (Value defaulted), Vabstol (1e-6), and labstol (1e-12); 'Temperature' with fields for Temp (27.0), Tnom (27.0), and a Tempeffects dropdown (all); 'Convergence' with Homotopy (all) and Limit (dev) dropdowns; 'Resistance' with Gmin (1e-12), Gmin check (max\_v\_only), and Rforce (1.0) fields; 'Matrix' with Pivrel (1e-3), Pivabs (0.0) fields, and a Pivotdc checkbox; and 'Component Controls' with Approx and Macromodels checkboxes, and Scalem (2.0), Scale (1.0), and Compatible (spectre) fields.

Section	Field/Control	Value/Option
Tolerances	Reltol	Value defaulted
	Vabstol	1e-6
	labstol	1e-12
Temperature	Temp	27.0
	Tnom	27.0
	Tempeffects	all
Convergence	Homotopy	all
	Limit	dev
Resistance	Gmin	1e-12
	Gmin check	max_v_only
	Rforce	1.0
Matrix	Pivrel	1e-3
	Pivabs	0.0
	Pivotdc	<input type="checkbox"/>
Component Controls	Approx	<input type="checkbox"/>
	Macromodels	<input type="checkbox"/>
	Scalem	2.0
Scale	Scale	1.0
	Compatible	spectre

2. Fill in and select fields as necessary.

The following table briefly describes the fields and tells you where to go for more information. For additional information, see the [“Immediate Set Options \(options\)”](#) section, in the “Specifying Controls for the Analog Solver” chapter of the *Virtuoso AMS*

## Virtuoso AMS Environment User Guide

### Elaborating, Simulating, and Plotting Results

---

*Simulator User Guide* and the “Immediate Set Options (options)” section, in the “Analysis Statements” chapter of the *Virtuoso Spectre Circuit Simulator Reference*.

---

Field	Corresponding spectre Option and Parameter	Effect
<i>Tolerances</i>		
<i>Reltol</i>	<code>options reltol</code>	Specifies the maximum relative tolerance for values computed in the last two iterations of a solution.
<i>Vabstol</i>	<code>options vabstol</code>	Specifies the absolute tolerance for differences in the computed values of the voltages in the last two iterations of a solution.
<i>Iabstol</i>	<code>options iabstol</code>	Specifies the absolute tolerance for differences in the computed values of the currents in the last two iterations of a solution.

## Virtuoso AMS Environment User Guide

### Elaborating, Simulating, and Plotting Results

Field	Corresponding spectre Option and Parameter	Effect
<i>Temperature</i>		
<i>Temp</i>	options temp	Specifies the circuit temperature in degrees Celsius.
<i>Tnom</i>	options tnom	Specifies the measurement (nominal) temperature in degrees Celsius.
<i>Tempeffects</i>	options tempeffects	<p>Defines how temperature affects the built-in primitive components. It takes the following three values:</p> <p>vt—Only thermal voltage</p> $V_t = \frac{kT}{q}$ <p>can vary with temperature.</p> <p>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.</p> <p>all—All built-in temperature models are enabled.</p>
<i>Convergence</i>		
<i>Homotopy</i>	options homotopy	Specifies the method to use if convergence fails on the initial DC analysis attempt.
<i>Limit</i>	options limit	Specifies the limiting algorithm used to aid DC convergence.

## Virtuoso AMS Environment User Guide

### Elaborating, Simulating, and Plotting Results

---

Field	Corresponding spectre Option and Parameter	Effect
<i>Resistance</i>		
<i>Gmin</i>	options gmin	Specifies the minimum conductance across each nonlinear device.
<i>Gmin check</i>	options gmin_check	Specifies how the effect of <i>Gmin</i> (if that effect is significant) is to be reported.
<i>Rforce</i>	options rforce	Specifies the resistance to be used when forcing nodesets and node-based initial conditions.
<i>Matrix</i>		
<i>Pivrel</i>	options pivrel	Specifies the relative pivot threshold.
<i>Pivabs</i>	options pivabs	Specifies the absolute pivot threshold.
<i>Pivotdc</i>	options pivotdc	Specifies that numeric pivoting be used on every iteration of DC analysis.



## Virtuoso AMS Environment User Guide

### Elaborating, Simulating, and Plotting Results

Field	Corresponding spectre Option and Parameter	Effect
<i>Component Controls</i>		
<i>Approx</i>	options approx	Specifies that approximate models are to be used. The difference between approximate and exact models is generally very small.
<i>Macromodels</i>	options macromodels	Specifies that the circuit contains macromodels. Sometimes specifying this information improves performance.
<i>Scalem</i>	options scalem	Specifies the scaling factor for models.
<i>Scale</i>	options scale	Specifies the scaling factor for device instances.
<i>Compatible</i>	options compatible	Specifies a simulator. AMS Designer changes device models to improve consistency with the models in the specified simulator.

3. When you are done changing options, click *OK* to save your changes.

### Specifying Output and Debug Options for the Analog Solver

You can access the output and debug option settings for the analog solver from the Cadence hierarchy editor.

## Virtuoso AMS Environment User Guide

### Elaborating, Simulating, and Plotting Results

---

1. In the AMS Options window, choose *Simulator – Analog Solver – Output/Debug* to open the AMS Options window to the *Output/Debug* pane.

The screenshot shows the 'Output/Debug' pane of the AMS Options window. It is divided into two sections: 'Output Options' and 'Debug Options'.

**Output Options**

- Print quantities:** no (dropdown)
- Print time taken for simulation:** detailed (dropdown)
- Print summary of components:** detailed (dropdown)
- ☒ **Narrate the simulation**
- ☒ **Print informational messages**
- ☒ **Print notice messages**      **Maximum per notice message:** 5 (text box)
- ☒ **Print warning messages**      **Maximum per warning message:** 5 (text box)
- ☒ **Print error messages**      ☐ **Print debugging messages**
- Number of digits per number:** 5 (text box)
- Numeric notation:** eng (dropdown)

**Debug Options**

- Analog circuit topology check:** full (dropdown)
- ☐ **Print information to diagnose accuracy and convergence problems**
- ☐ **Silently ignore shorted components**
- ☒ **Check operating point parameters against soft limits**

2. Fill in and select fields as necessary.

The following table briefly describes the fields. For additional information, see the [“Immediate Set Options \(options\)”](#) section, in the “Specifying Controls for the Analog Solver” chapter of the *Virtuoso AMS Simulator User Guide* and the “Immediate Set

## Virtuoso AMS Environment User Guide

### Elaborating, Simulating, and Plotting Results

---

Options (options)” section, in the [“Analysis Statements”](#) chapter of the *Virtuoso Spectre Circuit Simulator Reference*.

Field	Corresponding spectre Option and Parameter	Effect
<i>Output Options</i>		
<i>Print quantities</i>	options quantities	Prints quantities.
<i>Print time taken for simulation</i>	options audit	Prints the time required by various parts of the simulation.
<i>Print summary of components</i>	options inventory	Prints a summary of the components used.
<i>Narrate the simulation</i>	options narrate	Narrates the simulation.
<i>Print informational messages</i>	options info	Prints informational messages.
<i>Print notice messages</i>	options note	Prints notice messages.
<i>Maximum per notice message</i>	options maxnotes	Specifies the maximum number of times any particular notice will be issued per analysis.
<i>Print warning messages</i>	options warn	Prints warning messages.
<i>Maximum per warning message</i>	options maxwarn	Specifies the maximum number of times any particular warning will be issued per analysis.
<i>Print error messages</i>	options error	Prints error messages.
<i>Print debugging messages</i>	options debug	Prints debugging information.
<i>Number of digits per number</i>	options digits	Specifies the number of digits used when printing numbers.
<i>Numeric notation</i>	options notation	Specifies the notation to be used when displaying real numbers.

Field	Corresponding spectre Option and Parameter	Effect
<i>Debug Options</i>		
<i>Analog circuit topology check</i>	options topcheck	Checks the circuit topology for errors.
<i>Print information to diagnose accuracy and convergence problems</i>	options diagnose	Prints information that might help diagnose accuracy and convergence problems.
<i>Silently ignore shorted components</i>	options ignshorts	Tells the simulator to ignore shorted components silently.
<i>Check operating point parameters against soft limits</i>	options opptcheck	Checks the operating point parameters against soft limits.

- When you are done changing options, click *OK* to save your changes.

### Specifying a Transient Analysis for the Analog Solver

You can access the transient analysis option settings for the analog solver from the Cadence hierarchy editor.

- In the AMS Options window, choose *Simulator – Analog Solver – Tran Analysis* to open the AMS Options window to the *Tran Analysis* pane.

**Tran Analysis**

**Analysis title:**

**Stop time:**

**Error preset:**

☒ **Additional options:**

**2. Fill in and select fields as necessary.**

The following table briefly describes the fields. For additional information, see the “[Transient Analysis \(tran\)](#)” section, in the “Specifying Controls for the Analog Solver” chapter of the *Virtuoso AMS Simulator User Guide* and the “[Transient Analysis \(tran\)](#)” section, in the “Analysis Statements” chapter of the *Virtuoso Spectre Circuit Simulator Reference*.

Field	Corresponding spectre Option and Parameter	Effect
<i>Analysis title</i>	tran title	Specifies a title for the analysis.
<i>Stop time</i>	tran stop	Specifies the stop time for the analysis. You must specify a value for this field.
<i>Error preset</i>	tran errpreset	Specifies a collection of parameter settings for the analysis.
<i>Additional options</i>		<p>Specifies additional options that you want to append to the end of the tran card in the simulation control file.</p> <p>For example, you type</p> <pre>outputstart=0.0005</pre> <p>into the <i>Additional options</i> field. AMS Designer adds the value to the end of the tran card, like this.</p> <pre>amsAnalysis tran + stop = 0.001 + method = euler + relref = pointlocal + outputstart=0.0005</pre>

**3. When you are done changing options, click *OK* to save your changes.**

### Specifying Convergence and Accuracy Options for a Transient Analysis

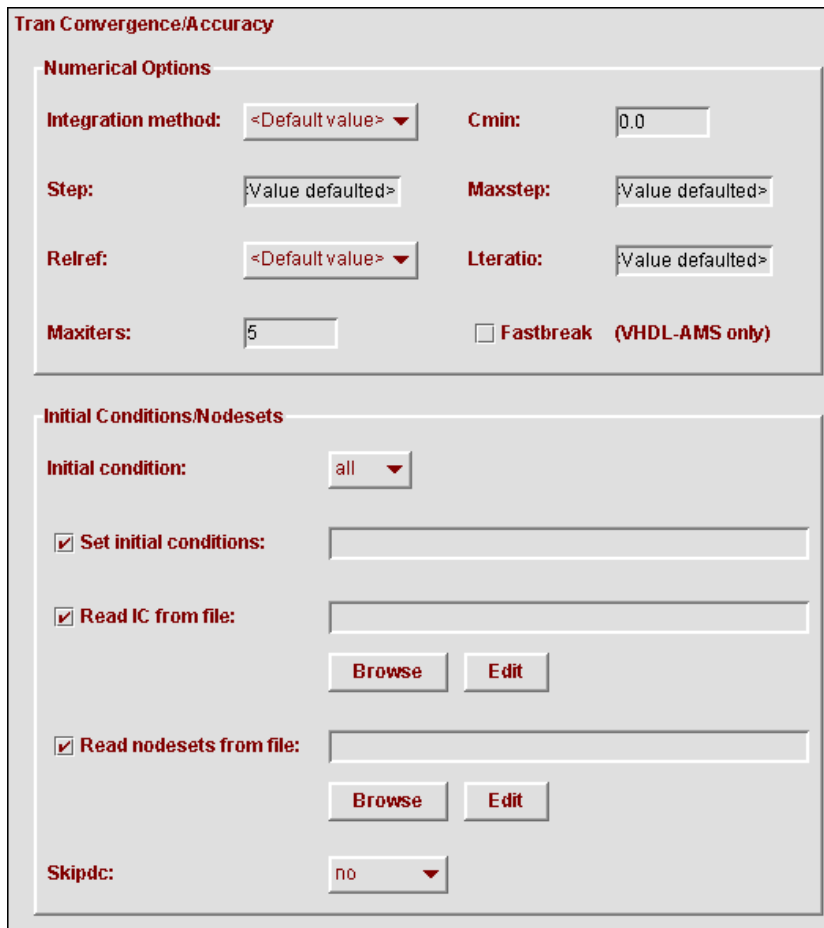
You can access the convergence and accuracy option settings for the transient analysis from the Cadence hierarchy editor.

## Virtuoso AMS Environment User Guide

### Elaborating, Simulating, and Plotting Results

---

1. In the AMS Options window, choose *Simulator – Analog Solver – Tran Analysis – Convergence/Accuracy* to display the *Tran Convergence/Accuracy* pane.



**Tran Convergence/Accuracy**

**Numerical Options**

Integration method: <Default value> Cmin: 0.0

Step: Value defaulted Maxstep: Value defaulted

Relref: <Default value> Lteratio: Value defaulted

Maxiters: 5 ☐ Fastbreak (VHDL-AMS only)

**Initial Conditions/Nodesets**

Initial condition: all

☒ Set initial conditions:

☒ Read IC from file: Browse Edit

☒ Read nodesets from file: Browse Edit

Skipdc: no

2. Fill in and select fields as necessary.

The following table briefly describes the fields. For additional information, see the [“Transient Analysis \(tran\)”](#) section, in the “Specifying Controls for the Analog Solver” chapter of the *Virtuoso AMS Simulator User Guide* and the [“Transient Analysis \(tran\)”](#)

## Virtuoso AMS Environment User Guide

### Elaborating, Simulating, and Plotting Results

---

section, in the “Analysis Statements” chapter of the *Virtuoso Spectre Circuit Simulator Reference*

Field	Corresponding spectre Option and Parameter	Effect
<i>Numerical Options</i>		
<i>Integration method</i>	tran method	Specifies the integration method to use.
<i>Cmin</i>	tran cmin	Specifies the minimum capacitance from each node to ground.
<i>Step</i>	tran step	Specifies the minimum time step to use. You might need to set this value to maintain the aesthetics of computed waveforms.
<i>Maxstep</i>	tran maxstep	Specifies the maximum time step.
<i>Relref</i>	tran relref	Specifies the reference to use for the relative convergence criteria.
<i>Lteratio</i>	tran lteratio	Specifies the ratio to use to compute LTE tolerances from Newton tolerance.
<i>Maxiters</i>	tran maxiters	Specifies the maximum number of iterations per time step.
<i>Fastbreak</i> (VHDL-AMS only)	tran fastbreak	Specifies the evaluation method to use for VHDL-AMS <code>break</code> statements.

## Virtuoso AMS Environment User Guide

### Elaborating, Simulating, and Plotting Results

---

Field	Corresponding spectre Option and Parameter	Effect
<i>Initial Conditions/ Nodesets</i>		
<i>Initial conditions</i>	tran ic	<p>Specifies the objects for which the simulator is to set initial conditions.</p> <p>For example, you specify the value <code>dev</code> for this field. As a result, the generated simulation control file contains the <code>ic = dev</code> option.</p> <pre>amsAnalysis tran + stop = 0.001 + ic = dev + method = euler</pre>
<i>Set initial conditions</i>	ic	<p>Specifies initial conditions for nodes and devices in the design.</p> <p>For example, you type the following into this field.</p> <pre>7=0 out=1 OpAmp1.comp=5 L1:1=1.0u</pre> <p>As a result, the generated simulation control file contains the statement:</p> <pre>ic 7=0 out=1 OpAmp1.comp=5 L1:1=1.0u</pre>
<i>Read IC from file</i>	tran readic	Specifies a file that contains initial conditions.
<i>Read nodesets from file</i>	tran readns	Specifies a file that contains nodesets.
<i>Skipdc</i>	tran skipdc	If yes, there is no DC analysis for transient.

3. When you are done changing options, click *OK* to save your changes.



## Specifying Output and Debug Options for a Transient Analysis

You can access the output and debug option settings for the transient analysis from the Cadence hierarchy editor.

1. In the AMS Options window, choose *Simulator – Analog Solver – Tran Analysis – Output* to display the *Tran Output* pane.

2. Fill in and select fields as necessary.

The following table briefly describes the fields. For additional information, see the “[Transient Analysis \(tran\)](#)” section, in the “Specifying Controls for the Analog Solver” chapter of the *Virtuoso AMS Simulator User Guide* and the “[Transient Analysis \(tran\)](#)” section, in the “Analysis Statements” chapter of the *Virtuoso Spectre Circuit Simulator Reference*

Field	Corresponding spectre Option and Parameter	Effect
<i>Write initial solution to file</i>	<code>tran write</code>	Directs the simulator to write the initial transient solution to the specified file.
<i>Write final solution to file</i>	<code>tran writefinal</code>	Directs the simulator to write the final transient solution to the specified file.

Field	Corresponding spectre Option and Parameter	Effect
<i>Print analysis statistics (stats)</i>	tran stats	Prints analysis statistics.
<i>Skipcount</i>	tran skipcount	Directs the simulator to save only one of every <i>skipcount</i> points.
<i>Skipstart</i>	tran skipstart	Specifies a time. The simulator saves all computed data before this time.
<i>Skipstop</i>	tran skipstop	Specifies a time to stop skipping output data.
<i>Strobeperiod</i>	tran strobeperiod	Specifies an interval. The simulator calculates and saves a data point in each interval.
<i>Strobedelay</i>	tran strobedelay	Specifies an offset time relative to the time specified by <i>Skipstart</i> .

3. When you are done changing options, click *OK* to save your changes.

## Setting Waveform Selection Options

You can access the waveform selection options from the Cadence<sup>®</sup> hierarchy editor. These settings determine the kind and scope of probes that you create by selecting objects. For more information, see [“Creating Probes”](#) on page 277.

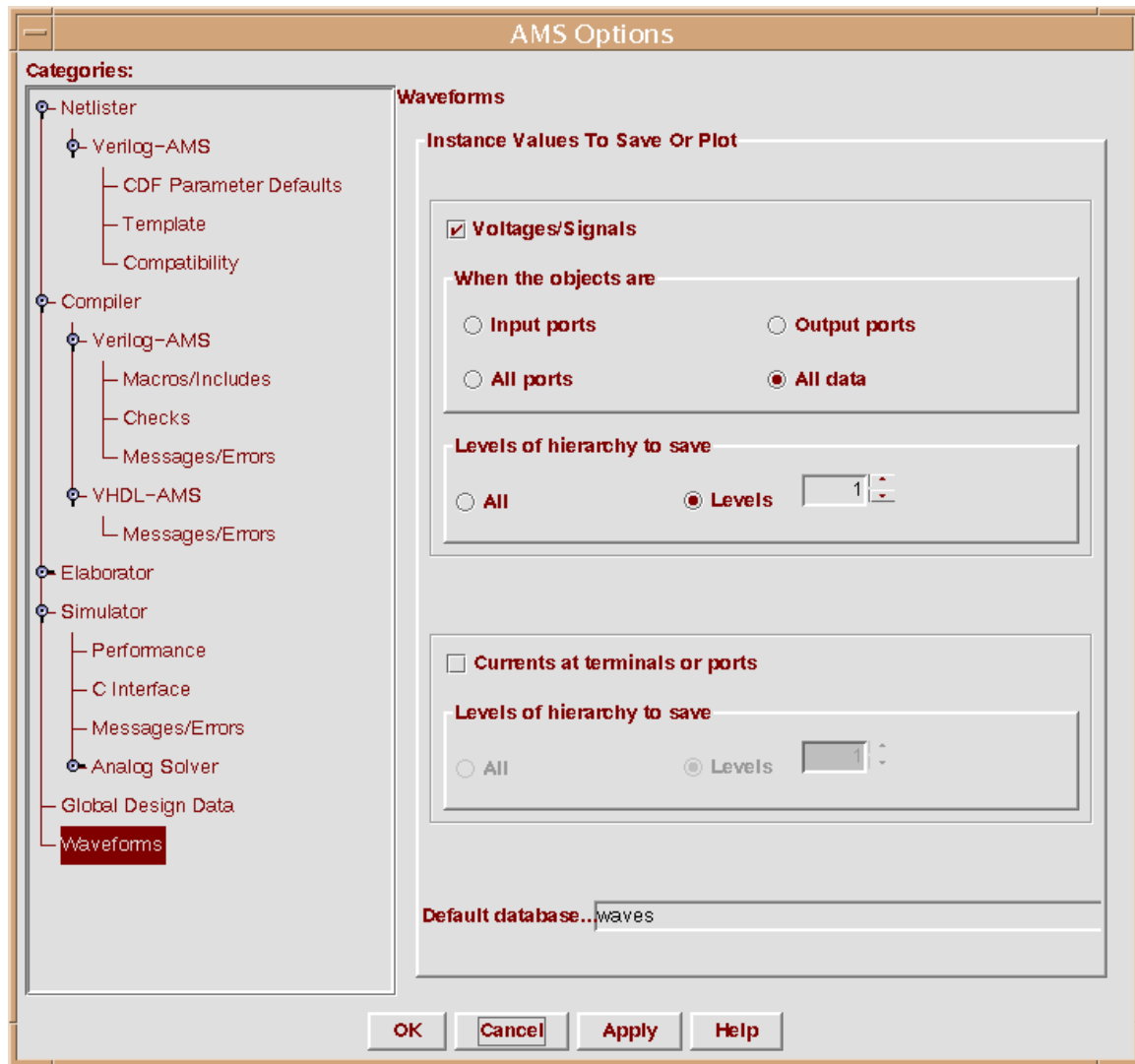
1. From the hierarchy editor, choose *AMS – Options – Waveforms*.

If the *AMS* menu entry is not visible, follow the instructions in [“Preparing to Use AMS Designer from the Hierarchy Editor”](#) on page 62.

## Virtuoso AMS Environment User Guide

### Elaborating, Simulating, and Plotting Results

The AMS Options window appears, showing the *Waveforms* pane.



2. Fill in and select fields as necessary.

## Virtuoso AMS Environment User Guide

### Elaborating, Simulating, and Plotting Results

---

The following table describes the effect of each control and field.

Field	Effect
<i>Instance Values to Save Or Plot</i>	
<i>Voltages/Signals</i>	Specifies whether to create voltage probes. The probes return the voltage of electrical nets or the digital value of nets that have disciplines from the discrete domain.
<i>When the objects are</i>	
<i>Input ports</i>	Specifies that only input ports are probed for voltage.
<i>Output ports</i>	Specifies that only output ports are probed for voltage.
<i>All ports</i>	Specifies that all ports are probed for voltage.
<i>All data</i>	Specifies that all ports and internal signals are probed for voltage.
<i>Levels of hierarchy to save</i>	
<i>All</i>	Specifies that voltages are to be probed and saved for the selected level and all lower levels of the hierarchy.
<i>Levels</i>	Specifies the number of levels (including the selected level) that are probed for voltage. If you probe an instance for both voltage and current, this <i>Levels</i> value is used for both, even if the <i>Levels</i> value for <i>Currents at terminals or ports</i> has a different value.

Field	Effect
<i>Currents at terminals or ports</i>	Specifies whether to create current probes. The probes return the current of electrical terminals or ports but return nothing if the terminals or ports have disciplines from the discrete domain.
<i>Levels of hierarchy to save</i>	
<i>All</i>	Specifies that currents are to be probed and saved for the selected level and all lower levels of the hierarchy.
<i>Levels</i>	Specifies the number of levels (including the selected level) that are probed for current. If you probe an instance for both voltage and current, this <i>Levels</i> value is ignored and the <i>Levels</i> value for <i>Voltages/Signals</i> is used for both.
<i>Default database name</i>	Specifies the name of the default database for signals. For information about customizing the database, see <a href="#">“Defining Databases for Waveforms”</a> on page 281.

3. When you are done changing options, click *OK* to save your changes.

## Creating Probes

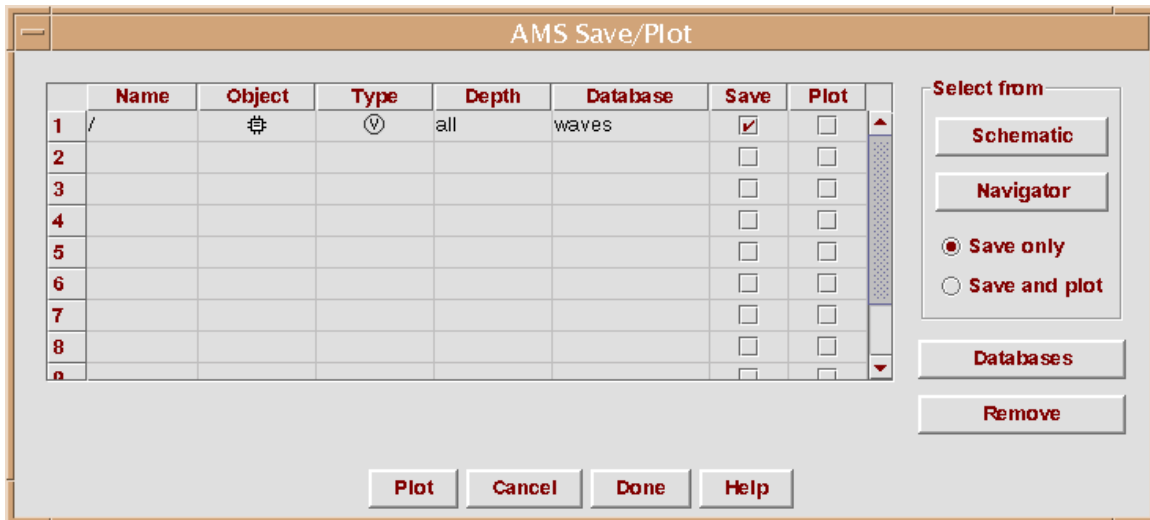
You can use Tcl commands to create voltage and current probes explicitly. The AMS environment, however, eases that task by allowing you to create probes by selecting objects from schematics and hierarchical tree views.

1. Choose *AMS – Save/Plot*.

## Virtuoso AMS Environment User Guide







### Elaborating, Simulating, and Plotting Results

The AMS Save/Plot window appears.



The first line in the table describes the default probe, which saves waveforms for the entire design. You can remove the check mark from the Save column for this line, but you cannot remove the line from the table. If you know which information you need, you can turn off the default probe and use other rows in the table to specify more targeted probes.




The following table describes the controls and fields.

Field	Effect
<i>Name</i>	The name of the probe. When you type in information to create a probe, you must type in the name before filling in the other probe information.
<i>Object</i>	<p>An icon representing the objects to be probed.</p> <ul style="list-style-type: none"> <li> A net or signal. For nets or signals, the <i>Depth</i> column is not applicable (N/A).</li> <li> A terminal. For terminals, the <i>Depth</i> column is not applicable (N/A).</li> <li> All the input ports of the instance.</li> <li> All the output ports of the instance.</li> <li> All the ports of the instance.</li> <li> All the ports and the internal signals of the instance.</li> </ul>

## Virtuoso AMS Environment User Guide

### Elaborating, Simulating, and Plotting Results

---

Field	Effect
<i>Type</i>	<p>An icon representing the type of measurement performed by the probe.</p> <p> Voltage. Returns the voltage of electrical nets or the digital value of nets that have disciplines from the discrete domain.</p> <p> Current. Returns the current of electrical terminals or ports but returns nothing if the terminals or ports have disciplines from the discrete domain.</p> <p> Current and voltage both.</p>
<i>Depth</i>	An integer specifying how many scope levels to descend when searching for objects to probe.
<i>Database</i>	The database to receive the probe data. Which databases are available depends on the specifications in the AMS Databases window. For more information, see <a href="#">“Defining Databases for Waveforms”</a> on page 281.
<i>Save</i>	Specifies whether the waveforms produced by the probe are saved.
<i>Plot (column)</i>	Specifies whether the waveforms produced by the probe are plotted during simulation.
<i>Select from</i>	
<i>Schematic</i>	Opens the Schematic Editing window, if it is not open, so that you can select objects in the schematic. The objects you select are added to the AMS Save/Plot table. For more information, see <a href="#">“Selecting Instances from the Virtuoso Schematic Editing Window”</a> on page 282.
<i>Navigator</i>	Opens the Scope Navigator window so that you can select instances from a tree representation of the design hierarchy. The instances you select are added to the AMS Save/Plot table. For more information, see <a href="#">“Selecting Instances from the Scope Navigator”</a> on page 284.
<i>Save only</i>	Specifies that probes added to the AMS Save/Plot window by selecting objects are added with only the <i>Save</i> column checked. You can switch between <i>Save only</i> and <i>Save and plot</i> at any time during the probe creation process.

Field	Effect
<i>Save and plot</i>	<p>Specifies that probes added to the AMS Save/Plot window by selecting objects are added with both the <i>Save</i> and the <i>Plot</i> columns checked. You can switch between <i>Save and plot</i> and <i>Save only</i> at any time during the probe creation process.</p> <p>The effect of checking both the <i>Save</i> and <i>Plot</i> columns depends on the simulation <i>Run Mode</i> setting.</p> <ul style="list-style-type: none"> <li>■ In the <i>GUI</i> mode, the waveforms are marched.</li> <li>■ In the <i>Batch</i> mode, the waveforms are saved and then plotted at the end of the simulation.</li> <li>■ In the <i>Tcl</i> mode, the waveforms are only saved.</li> </ul>
<i>Databases</i>	<p>Opens the AMS Databases window so that you can specify the characteristics of waveform databases. For guidance, see <a href="#">“Defining Databases for Waveforms”</a> on page 281.</p>
<i>Remove</i>	<p>Removes highlighted rows from the AMS Save/Plot table.</p>
<i>Plot</i> (button)	<p>Plots or replots the waveforms saved by the probes designated by checks in the <i>Plot</i> column. Only waveforms saved during previous simulations can be plotted this way.</p>

**2. (Optional) Use the AMS Databases window to define the databases you need.**

For guidance, see [“Defining Databases for Waveforms”](#) on page 281.

**3. Define the probes.**

You can:

- ☐ Click an empty row and fill in the name and then the other columns to create a probe.
- ☐ Click the *Schematic* button to select objects to be probed from the schematic editing window.

For guidance, see [“Selecting Instances from the Virtuoso Schematic Editing Window”](#) on page 282.

- ☐ Click the *Navigator* button to select instances to be probed from a tree display of the hierarchy.

For guidance, see [“Selecting Instances from the Scope Navigator”](#) on page 284.



## Virtuoso AMS Environment User Guide

### Elaborating, Simulating, and Plotting Results

- ❑ Right-click on a module in the hierarchy editor *Tree* view of a design configuration and select *Save Instance Data*.
- ❑ Copy existing rows to new rows and then modify the new rows as necessary.

For guidance, see [“Copying and Pasting Within Tables”](#) on page 285.

## Defining Databases for Waveforms

The *Database* column in the AMS Save/Plot window specifies the databases to be used for the waveforms returned by the various defined probes. There is a predefined default database named *waves*, but you can also define additional databases by following these steps:

1. In the AMS Save/Plot window, click *Databases*.

The AMS Databases form appears.

	Name	Compress	Maximum Size (bytes)
1	fast_db	<input checked="" type="checkbox"/>	
2	tempdb	<input checked="" type="checkbox"/>	
3		<input checked="" type="checkbox"/>	
4		<input checked="" type="checkbox"/>	

2. (Optional) Click the right-facing arrow at the upper, right-hand corner of the table twice.

The form expands to display additional columns.

	Name	Compress	Maximum S...	Statements	Events
1	fast_db	<input checked="" type="checkbox"/>		<input type="checkbox"/>	<input checked="" type="checkbox"/>
2	tempdb	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input type="checkbox"/>
3		<input checked="" type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>
4		<input checked="" type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>

## Virtuoso AMS Environment User Guide

### Elaborating, Simulating, and Plotting Results

---

The following table describes the information in each column.

Field	Effect
<i>Name</i>	The name of the database.
<i>Compress</i>	Specifies whether the database is compressed to reduce its size.
<i>Maximum Size (bytes)</i>	The maximum size of the database.
<i>Statements</i>	Specifies whether statement trace data is written to the database. If <i>Statements</i> is checked, all value change events are also written to the database, whether <i>Events</i> is checked or not.
<i>Events</i>	Specifies whether all value change events are written to the database. If <i>Events</i> is not checked, the simulator discards multiple value changes for an object during one simulation time and writes only the final value at the end of that simulation time.

3. Highlight a row to change an existing definition or to add a new one.

You can also copy and paste rows. For guidance, see [“Copying and Pasting Within Tables”](#) on page 285.

4. Click *Close* when you are done defining databases.

The AMS Databases form closes and the databases you defined become available as choices for the *Database* column of the AMS Save/Plot form.

## Selecting Instances from the Virtuoso Schematic Editing Window

The Virtuoso Schematic Editing window presents a schematic view of the design. From that view, you can select objects for which you want to create probes. The precise probes created for each object depend on the options specified in the *Waveforms* pane of the AMS Options window. For guidance on setting those options, see [“Creating Probes”](#) on page 277.

To select objects from the Schematic Editing window,

1. Click *Schematic* on the AMS Save/Plot window.

The Virtuoso Schematic Editing window appears with this message:

Select objects from schematic to specify outputs to be saved. Press Esc to stop selections.

**2. Select the objects you want to probe.**

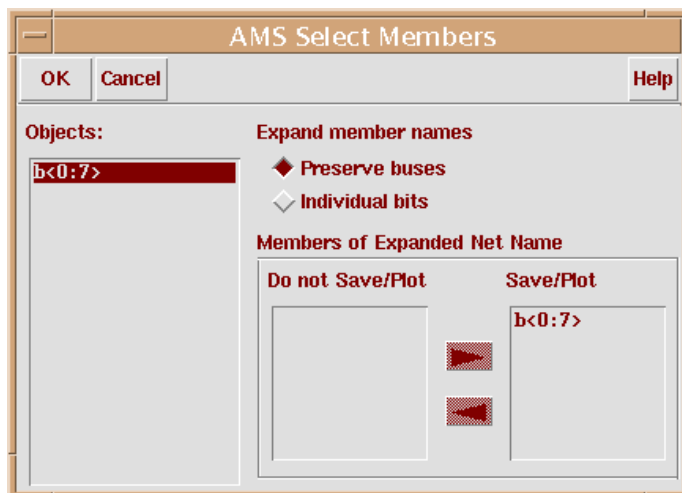
The names of the selected objects appear in the first column of the AMS Save/Plot table and the other columns are filled in according to the options specified in the *Waveforms* pane of the AMS Options window.

Selecting an object that is already in the AMS Save/Plot table removes the object from the table.

**3. Press the `Escape` key when you are done selecting objects.**

## Selecting Buses

When you select a bus to save or plot, AMS Designer displays the AMS Select Members form, which gives you the opportunity to select the bus as a whole or to select from among the bits that compose the bus.



The following table describes the fields.

---

Field	Effect
<i>Objects</i>	Lists the objects or buses you selected.

---

## Virtuoso AMS Environment User Guide

### Elaborating, Simulating, and Plotting Results

---

---

Field	Effect
<i>Expand member names</i>	
<i>Preserve buses</i>	Specifies that buses are to be considered as wholes. As a result, the <i>Save/Plot</i> field lists each bus as a whole.
<i>Individual bits</i>	Specifies that buses are to be considered as individual bits. As a result, the <i>Save/Plot</i> field displays the individual bits that compose each bus so that you can save or plot specific bits.
<i>Members of Expanded Net Name</i>	
<i>Do not Save/Plot</i>	Contains the names of objects that are not to be saved or plotted.
<i>Save/Plot</i>	Contains the names of objects to be saved or plotted.

---

To select the buses or bits to be saved or plotted,

1. Turn on either *Preserve buses* or *Individual bits*.
2. Use the left and right arrows to move buses or bits as necessary between the *Do not Save/Plot* and the *Save/Plot* lists.
3. Click *OK*.

The AMS Select Members form closes and the buses or bits are added to the list in the AMS Save Plot window.

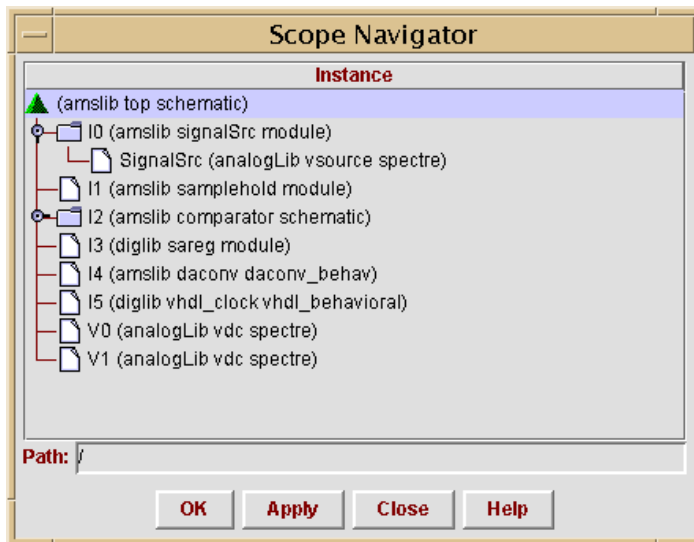
## Selecting Instances from the Scope Navigator

The Scope Navigator window presents a tree view of the design hierarchy. From that view, you can select the instances for which you want to create probes. The precise probes created for each instance depend on the options specified in the *Waveforms* pane of the AMS Options window. For guidance on setting those options, see [“Creating Probes”](#) on page 277.

To select instances from the Scope Navigator,

1. Click *Navigator* on the AMS Save/Plot window.

The Scope Navigator window appears.



2. Expand branches as necessary to reveal the instances you want to probe.
3. Highlight one or more instances.

You can use the `Shift` key and the left mouse button to select a range of instances, or use the `Control` key and the left mouse button to highlight individual instances.

4. If only a single instance is highlighted, the hierarchical path of the instance appears in the *Path* field.
5. Click **OK**.

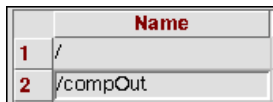
The names of the highlighted instances appear in the first column of the AMS Save/Plot table and the other columns are filled in according to the options specified in the *Waveforms* pane of the AMS Options window.

## Copying and Pasting Within Tables

You can create new rows in a table by copying one or more existing rows, pasting the copy into new rows, and then modifying the content of the new rows as necessary. This approach works for both the AMS Save/Plot table and the AMS Databases table.

- To copy complete rows,
  - a. Be sure that none of the type-in fields in the table, such as *Name*, has focus.

A type-in field with focus, such as the row 2 *Name* field in this example,



restricts the focus to that field. If one of the fields has focus, you can remove the focus by pressing `Escape`.

- b.** Highlight the rows that you want to copy by clicking the row numbers on the left side.

After you highlight one row, you can highlight additional rows by `Shift` clicking (for contiguous rows) or `Control` clicking (for scattered rows).

- c.** Right click to open the pop-up menu and choose *Copy*.
- d.** Position the cursor over the row where you want to paste the copied row (or, if you are pasting multiple rows, over what is to be the topmost pasted row).
- e.** Right click to open the pop-up menu and choose *Paste*.

- To copy all or part of a type-in field,

- a.** Click the type-in field to be copied.

The focus moves to that field.

- b.** Within the type-in field, highlight the part of the field to be copied.
- c.** Right click to open the pop-up menu and choose *Copy*.
- d.** Click the target type-in field.

Focus moves to the target field.

- e.** Right click to open the pop-up menu and choose *Paste*.

## Elaborating and Simulating

To use AMS Designer to simulate a design, you must have a license for the AMS simulator (product number 70001). If you attempt to simulate a design without a license, the simulation appears to run indefinitely. AMS Designer does not produce an obvious message to help you understand the problem, although the tool does place a message in the `ncsim.log` file.

To elaborate and simulate your design,

1. From the Cadence hierarchy editor, choose *AMS – Run Simulation*.

The AMS Run Simulation form appears.

The screenshot shows the 'AMS Run Simulation' dialog box. It is organized into several sections:

- Configuration:** Library: arnslib, Cell: top, View: config.
- Global Design Data Module:** Library: arnslib, Cell: cds\_globals, View: top\_config.
- Connect Rules:** Library: (empty), Cell: ConnRules\_5V\_full, View: (empty).
- Simulation Snapshot:** Library: (empty), Cell: top, View: arns107090957742.

Below these sections are several controls:

- Buttons: Elaborator Options, Simulator Options.
- Tran analysis stop time: 14e-6.
- Buttons: Analog Models Setup, Run Mode: GUI (dropdown).
- Checkboxes: ☒ Run Elaborator, ☒ Run Simulator.
- Buttons: Save, NC commands to runElabSim script.
- Buttons: Run, Cancel, Apply, Help.

2. Ensure that the correct library, cell, and view are specified in the *Configuration* section.
3. Ensure that the correct library and view are specified in the *Global Design Data Module* section.

The `cds_globals` module, which is created by AMS Design Prep, contains information about global signals and design variables. The cell is always named `cds_globals` and, by default, the view name is created from the top-level cell name and the config name, using the format `topCell_configView`.

4. In the *Connect Rules* section, specify the `connectrules` module to be used.

The `connectrules` module tells the elaborator when to insert connection modules. Which default `connectrules` module appears in the form depends on the version of the AMS simulator that you are using. For more information, see [“Using Connect Modules”](#), in the “Mixed-Signal Aspects of Verilog-AMS” chapter of the *Cadence Verilog-AMS Language Reference*.

5. (Optional) Change the information in the *Simulation Snapshot* section.

The default snapshot view name incorporates a time mark to ensure that existing snapshots in the library are not accidentally overwritten.

You might want to save the snapshot under a simpler, more convenient name if you plan to simulate a different existing snapshot, or if you want to resimulate this snapshot from a different run directory.

**Note:** Before simulating, be sure that the *View* field correctly names a valid snapshot view. If you type in a snapshot view that does not exist in the library and attempt to simulate, the simulation fails without issuing an error message.

6. (Optional) Click *Elaborator Options* if you want to change those options.
7. (Optional) Click *Simulator Options* if you want to change those options.
8. (Optional) Enter a stop time in the *Tran analysis stop time* field if the stop time has not been specified in some other way, such as by using a simulation control file.
9. (Optional) Click *Analog Models Setup* if you want to change the models that are used in the design
10. If you want to run the elaborator, ensure that *Run Elaborator* is checked.
11. If you want to run the simulator, ensure that *Run Simulator* is checked.
12. In the *Run Mode* cyclic field, choose whether you want to simulate in GUI (interactive), Tcl, or batch mode.

*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. (For a description of the Tcl commands, see Appendix B, “Tcl-Based Debugging,” in the *Virtuoso AMS Simulator User Guide*.) (Note: If the Tcl window does not appear, verify that the `xterm` command is in your path.)

- If you have not specified a Tcl input script, the window opens and waits for you to enter 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 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 enter a Tcl command.

To close the window, type the `exit` or `finish` commands.



*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.

13. (Optional) Click *Save* to write the `ncelab` and `ncsim` commands to the `runElabSim` script in the run directory.

Writing the commands to a file makes it possible for you to reuse the commands later.

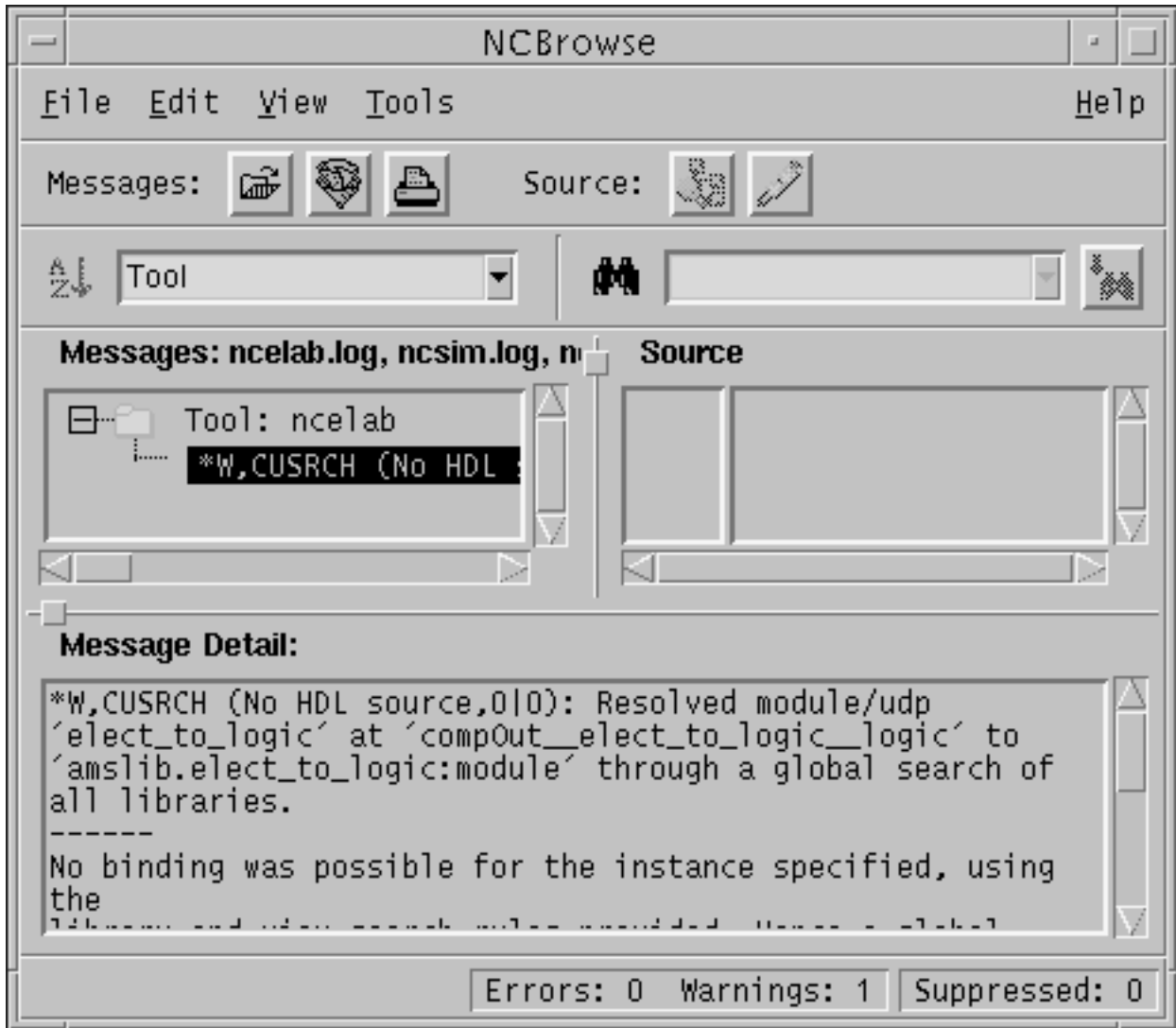
14. Click *Run*.

## Viewing Messages

AMS Designer places the `ncelab.log`, `ncsim.log`, `ncvhdl.log`, and `ncvlog.log` files in the run directory. You can view them there by using a tool such as `vi`. If your primary interest is in viewing warnings and errors found in these log files, you can use the following steps instead.

1. From the Cadence hierarchy editor, choose *AMS – Logfile Viewer – Simulator Log Files*.

The NCBrowse window appears.



2. Highlight one of the messages in the *Messages* pane.

Detailed information about that message appears in the *Message Detail* pane. For more information about the NCBrowse window, see the *NCBrowse Message Browser User Guide*.

## Plotting Waveforms After Simulation Ends

Earlier sections in this chapter describe how to plot waveforms while the simulator is running. The following sections describe how, if the waveforms are saved during the simulation, you

can also plot them after the simulation ends. You might want to do this, for example, to compare the results of earlier and later simulations.

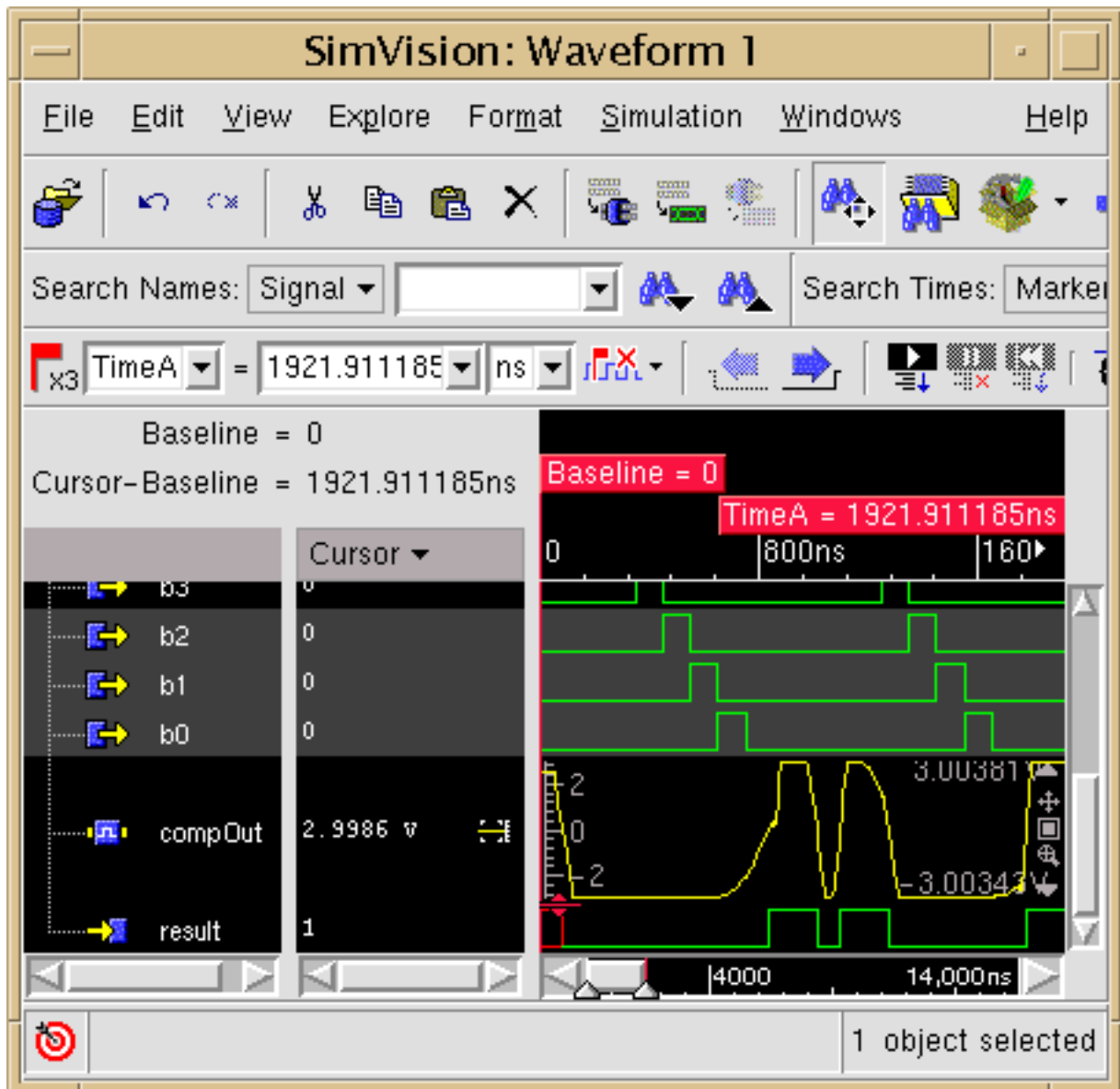
## **Starting the SimVision Waveform Viewer**

To use the capabilities of the SimVision Waveform Viewer to select and plot waveforms,

1. From the Cadence hierarchy editor, choose *AMS – Waveform Viewer*.

A new SimVision Waveform window appears. A waveform window opened in this way does not communicate with AMS Designer and operates independently. To open or

reuse a Waveform window that communicates with AMS Designer, either click *Plot* on the AMS Save/Plot form, or in the hierarchy editor, choose *AMS – Direct Plot*.



- 2. Open the waveform data base.**

The waveform data is located in the `waves.shm` subdirectory of the run directory.

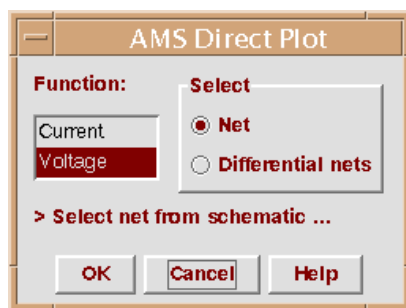
For a brief example of using the SimVision Waveform viewer, see [Chapter 2, “Quick-Start Tutorial.”](#) For detailed information about the viewer, see the *SimVision Waveform Viewer User Guide*.

## Plotting Waveforms Selected on a Schematic (Direct Plot)

With the AMS direct plot capability, you can select nets on schematics and have the corresponding waveforms automatically plotted in the waveform window. This approach works only if the necessary waveforms are saved during the simulation, otherwise the required data are not available to be plotted.

1. In the Cadence hierarchy editor, choose *AMS – Direct Plot*.

The AMS Direct Plot window appears.



The schematic editing window also appears.

2. Use the fields in the form to specify the information to be plotted.

Field	Effect
<i>Function</i>	
<i>Current</i>	Plots the current for the net you select on the schematic.
<i>Voltage</i>	Plots the voltage for the net you select on the schematic.
<i>Select</i>	
<i>Net</i>	Specifies that each selection of a net on the schematic is complete by itself and that either the voltage or current for that net is to be returned.
<i>Differential nets</i>	Specifies that each selection consists of two nets and that the plot is to display the difference of those two nets.

As you switch between *Net* or *Differential nets*, the message in the AMS Direct Plot window changes accordingly.

3. Select nets in the schematic editing window.

The corresponding waveforms appear in the waveform window.

4. Press the `Escape` key when you are done selecting objects.

---

## Using the amsdesigner Command

---

The `amsdesigner` command allows you to run AMS Designer from the command line or from a script. The command includes options for netlisting, compiling, elaborating, and simulating.

```
amsdesigner_command ::=
    amsdesigner [-help | -version]
    | amsdesigner -lib libName -cell cellName -view viewName
      action_option {action_option}
      [-log logFileName]
      [-cdslib filePath]
      [-cdsglobals overwriteEdits | retainEdits]

action_option ::=
    -netlist incremental | all | none
    | -compile whenNetlist | all | none
    | -elaborate
    | -simulate
```

The following table describes the `amsdesigner` command options and values.

amsdesigner Option and Value	Effect
<b>-Lib</b> <i>libName</i>	Specifies the library containing the configuration that you want to process.
<b>-Cell</b> <i>cellName</i>	Specifies the cell containing the configuration that you want to process.
<b>-View</b> <i>viewName</i>	Specifies the cellview name of the configuration that you want to process. The <code>amsdesigner</code> command opens this configuration in read-only mode.

## Virtuoso AMS Environment User Guide

### Using the amsdesigner Command

amsdesigner Option and Value	Effect
<b>-Log</b> <i>logFileName</i>	Tells the amsdesigner tool to write messages to <i>logFileName</i> . Default: <code>./amsdesigner.log</code> <ul style="list-style-type: none"> <li>■ If <i>logFileName</i> is an absolute path, the log file is written to <i>logFileName</i>.</li> <li>■ If <i>logFileName</i> is a relative path, <i>logFileName</i> is placed in a location that is relative to the current directory.</li> </ul>
<b>-CDSLlib</b> <i>filePath</i>	Specifies a cdslib file to load. Default: <code>./cds.lib</code>
<b>-CDSGlobals</b>	You can omit the <code>-CDSGlobals</code> option if the <code>cds_globals</code> module has not been edited by hand. If the <code>cds_globals</code> module has been edited by hand, you must use the <code>-CDSGlobals</code> option and you must specify a value for the option.
<b>overwriteEdits</b>	Tells the amsdesigner tool to regenerate and overwrite the <code>cds_globals</code> module as necessary, even if the module has been edited by hand.
<b>retainEdits</b>	Tells the amsdesigner tool not to overwrite a hand-edited <code>cds_globals</code> module. However, the <code>retainEdits</code> value allows the tool to overwrite the <code>cds_globals</code> module if the module has <i>not</i> been edited by hand.
<b>-Help</b>	Returns a brief description of the amsdesigner command and its options.
<b>-VERSION</b>	Returns version information, including the versions of the amsdesigner, hierarchy editor, amsdirect, and nclog tools and the versions of input and output files used by the hierarchy editor.
<b>-Netlist</b>	
<b>incremental</b>	Tells the amsdesigner tool to produce Verilog-AMS netlists for only new or revised cellviews.
<b>all</b>	Tells the amsdesigner tool to netlist all cellviews in the configuration, whether they have changed since the previous netlisting or not.
<b>none</b>	Turns off netlisting for all cellviews. This is the default value when the <code>-Netlist</code> option is not specified.



## Virtuoso AMS Environment User Guide

### Using the amsdesigner Command

---

amsdesigner Option and Value	Effect
<hr/>	
<b>-CCompile</b>	
<b>whenNetlist</b>	Tells the amsdesigner tool to compile only cellviews netlisted in this run.
<b>all</b>	Tells the amsdesigner tool to compile all cellviews in the configuration, whether netlisted in this run or not.
<b>none</b>	Turns off compilation for all cellviews. This is the default value when the -CCompile option is not specified.
<b>-Elaborate</b>	Tells the amsdesigner tool to elaborate the design.
<b>-Simulate</b>	Tells the amsdesigner tool to simulate the design.

---

The `amsdesigner` command is intended for rerunning designs that have been previously simulated with the AMS environment. The `amsdesigner` command uses setup information that cannot be entered as options on the command line. Before you can run the `amsdesigner` command, you must provide that setup information by using the windows in the AMS environment. To ensure that the setup information is usable, the run directory and input files used during the AMS environment simulation must then remain unchanged for runs of the `amsdesigner` command.

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.

## 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 tools and files used by the `amsdesigner` command and then exits. If you need to communicate with Cadence, you might use a command like this to obtain useful background information.

## Virtuoso AMS Environment User Guide

### Using the amsdesigner Command

---

```
amsdesigner -version
```

The returned information includes information about the tools and files used by the amsdesigner command.

```
@(#) $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)
```

---

## Producing Customized Netlists

---

This chapter contains the following sections:

- [Producing Customized Netlists](#) on page 300
- [Examples: Problems Addressed by Customized Netlists](#) on page 314
- [Data Objects Supported for Netlisting](#) on page 327
- [SKILL Functions Supported for Netlisting](#) on page 333

## Producing Customized Netlists

The capabilities described in this chapter provide a way for you to customize both the format and the content of the netlists produced by the AMS netlister. To enable this customization, AMS Designer provides several kinds of tools, including:

- 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 used by the netlister, so you can read and, in some cases, modify the information stored there

These capabilities, along with the customizations possible by using `ams.env` variables, 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 “[SKILL Functions Supported for Netlisting](#)” on page 333, and to the descriptions of `ams.env` variables in Appendix A, “[Variables for ams.env Files](#).”

## 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 13-1 Sections of a Netlist**

Comment	<code>// Verilog-AMS netlist generated by the AMS netlister, version 0123.</code> <code>// Cadence Design Systems, Inc.</code>	
Header		
Includes list	<code>`include "disciplines.vams"</code>	
Module interface	<code>module \sample-cell ( b,a,d,c );</code>	
Port declarations	<code>input [0:2] b;</code> <code>input a;</code> <code>output [0:1] d;</code> <code>input [1:3] c;</code>	
Signal declarations	<code>wire</code> <code>(* integer inh_conn_prop_name="PWR";</code> <code>integer inh_conn_def_value="cds_globals.\vdd! "; *)</code> <code>\vdd! ;</code>  <code>wire</code> <code>(* integer inh_conn_prop_name="bulk_n";</code> <code>integer inh_conn_def_value="cds_globals.\gnd! "; *)</code> <code>\bulk_n_gnd! ;</code>	Attributes
Parameter declarations	<code>parameter foo=2.23;</code> <code>parameter bar=2.3;</code>	
Instances	<code>nmos4 #(.w(10u), .l(1u)) (* integer library_binding = "analogLib"; *)</code> <code>M0 ( \vdd! , net6, net7, \bulk_n_gnd! );</code>  <code>block1 #(.p(10), .q(3.4)) (* integer library_binding = "netproc"; *)</code> <code>i0 ( .b( d[0:1] ), .a( { a,b[0] } ) );</code>	Instance master  Instance name Instance parameters Instance connections
End module	<code>endmodule</code>	

For a table that lists the netlisting procedures responsible for generating these labeled sections of the netlist, see [Table 13-3](#) on page 329.

## Using `ams.env` Variables to Customize Netlists

As you think about ways to customize AMS Designer netlists, be sure to consider first the capabilities provided by the `ams.env` variables. These variables control a large number of netlisting behaviors, including those listed in the following table. For detailed descriptions, see “Variables for `ams.env` Files” on page 359.

**Table 13-1 Customizations Provided by `ams.env` Variables**

<b><code>ams.env</code> variable</b>	<b>Customization provided</b>
<u><code>aliasInstFormat</code></u>	Specifies the format to be used to create names for instances of the <code>cds_alias</code> module.
<u><code>amsDefinitionViews</code></u>	Specifies a list of views that can be used to determine the vectored terminal range direction and terminal order for cellviews being netlisted.
<u><code>amsExcludeParams</code></u>	Specifies a list of parameters to be excluded from netlists.
<u><code>amsExpScalingFactor</code></u>	Controls the expansion of scaling factors for parameter values.
<u><code>amsLSB_MSB</code></u>	Specifies the direction of a range.
<u><code>amsScalarInstances</code></u>	Determines whether iterated instances are netlisted as arrayed or exploded instances.
<u><code>headerText</code></u>	Inserts headers.
<u><code>ifdefLanguageExtensions</code></u>	Determines whether attributes are enclosed in <code>`ifdef</code> INCA clauses.
<u><code>includeFiles</code></u>	Specifies files to include with <code>`include</code> statements.
<u><code>includeInstCdfParams</code></u>	Determines how the AMS netlister handles CDF parameters.
<u><code>instClashFormat</code></u>	Determines the format to be used to map the names of instances that collide with names of other netlist constructs.
<u><code>iterInstExpFormat</code></u>	Specifies the format to be used for the names of constituent elements generated by the expansion of an iterated instance.
<u><code>modifyParamScope</code></u>	Specifies that the AMS netlister treat <code>atPar</code> and <code>dotPar</code> expressions as <code>pPar</code> and <code>iPar</code> expressions, respectively.

**Table 13-1 Customizations Provided by `ams.env` Variables, *continued***

<b>ams.env variable</b>	<b>Customization provided</b>
<u><code>netClashFormat</code></u>	Specifies the format to be used to map the names of nets that collide with names of other netlist constructs.
<u><code>paramDefVals</code></u>	Supplies default values for parameters.
<u><code>paramGlobalDefVal</code></u>	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 <code>paramDefVals</code> variable.
<u><code>templateFile</code></u>	Embeds files in netlists.
<u><code>templateScript</code></u>	Embeds the output of a script in netlists.
<u><code>useDefparam</code></u>	Specifies whether parameter values are passed with <code>defparam</code> statements.

## Using Netlisting Procedures to Customize Netlists

If changing the values of `ams.env` variables does not provide enough flexibility to generate netlists in the format and with the content you need, then consider using customized netlist procedures. With the power provided by customized netlist procedures, you can fundamentally change the netlists produced by AMS. 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 need a way to define new functions and to have those new functions recognized by the AMS netlister. You also need a way to 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 304
- [Deciding What Kind of Override File to Use](#) on page 305
- [Replacing Default Procedures with Custom Procedures](#) on page 308
- [Loading an Override File](#) on page 308

- [Using Netlisting Procedures for Particular Instance Masters](#) on page 309
- [Using Netlisting Procedures to Customize the simInfo Values of Instances](#) on page 310

### ***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 [“SKILL Functions Supported for Netlisting”](#) on page 333. 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 <a href="#">“Replacing Default Procedures with Custom Procedures”</a> on page 308.  For example, <pre>formatterId-&gt;wiresProc</pre> returns the name of the wire printing procedure: <pre>amsPrintWires</pre>
----	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------



## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

---

->?

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 (CDBA namespace)
type	string	- type of parameter from <i>ams.env</i> (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, and which kind is most appropriate depends on how the netlisting procedures are used.

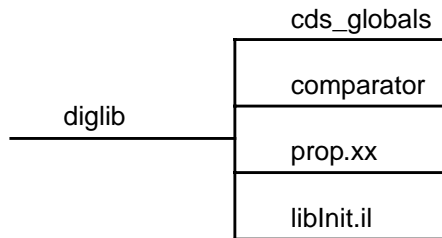
- Use `libInit.il` files to make netlisting procedures available for the cells in one or more libraries. A file named `libInit.il` is read automatically when the library containing it is used.
  - If you are defining a procedure that is specific to the cells in a particular library, place the procedure 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.

## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

---

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:

```
netlistId = amsGetNetlistId()
;; Get the Verilog-AMS formatterId.
vlogFormId = netlistId->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.

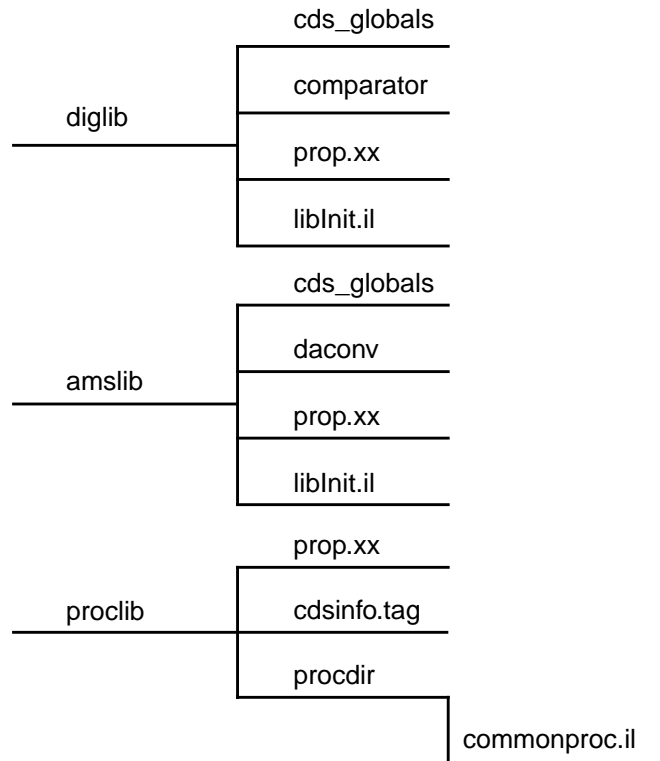
For example, you have a custom netlisting procedure called `mycustproc`, defined in the file `commonproc.il`. You place `commonproc.il` in a subdirectory called

## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

---

`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 308.

Do not use `.cdsinit` files for override procedures because these files are not read when the netlister runs by itself or from the Cadence 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 310.

## ***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.

Notice in the following example, how the `->` operator 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. You can

- Specify the `initFile` variable (described in [“initFile”](#) on page 414) in one of the `ams.env` files loaded by AMS Designer. For example, if your customized procedures are defined in the `custom.il` file, your `initFile` specification might look like this.

```
amsDirect initFile string "custom.il"
```

Before netlisting, the AMS netlister loads the specified override file, making the customized procedures available for use.

- 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 in a changed override file, you can avoid restarting AMS Designer after each change.

## 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 of the Edit Component CDF form, click *Edit*.  
The Edit Simulation Information form appears.
5. Select *Base* in the *CDF Type* field.
6. From the *Choose Simulator* pulldown, select *ams*.
7. Specify the custom procedure or procedures in the *netlistProcedure* field.

The function information for the *netlistProcedure* field can take either of two forms.

## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

---

- ❑ 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, you type the following into the *netlistProcedure* field.

```
myPrintInstance
```

This entry is equivalent, for this instance master only, of setting

```
vlog->instanceProc = 'myPrintInstance
```

To use the DPL syntax, you type, for example, the following into the *netlistProcedure* field.

```
nil masterName myPrintInstMasterName params myPrintInstParams
```

This entry is equivalent, for this instance master only, of setting

```
vlog->instanceMasterNameProc = 'myPrintInstMasterName
vlog->instanceParametersProc = 'myPrintInstParams
```

8. Click **OK** in the Edit Simulation Information form.

9. Click **OK** in the Edit Component CDF form.

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.

## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

---

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. At this point you either enter the procedure directly into the `simInfo` field, using a lambda function, or you 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 Right Customization Approach for Your Needs

As you plan how to customize your netlists, spend some time thinking about how you can achieve your desired results with the least amount of effort. For example, using `ams.env` variables, while simple to do, provides only limited customization. On the other hand, writing your own netlisting functions allows you to radically change the netlists that are produced, but costs more in understanding, development, and maintenance. Or you might find that the middle-of-the-road approach described in the next section provides the greatest power for the least amount of effort.

(If this approach still does not allow you to generate the netlists you need, the most flexible and powerful approach is discussed in [“More Powerful Approach: Using Fully Customized Netlisting Procedures”](#) on page 314.)

### ***Simpler Approach: Changing Object Values and Netlisting With 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 [“Data Objects Supported for Netlisting”](#) on page 327.

Aside from changing the values of `ams.env` variables, the easiest way to modify the netlists that the AMS netlister produces is to change the values of fields in the appropriate objects and then to use 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;
```



## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

---

```
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 [“Data Objects Supported for Netlisting”](#) on page 327.

## **Examples: Problems Addressed by 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 various problems can be resolved by using netlisting procedures.

- [Example: Adjusting Parameter Values to Account for Number of Fingers](#) on page 314
- [Example: Using Symbols that Represent Verilog Test Code](#) on page 318
- [Example: Using CDF Instance Parameters to Define Inherited Connections](#) on page 322

### **Example: Adjusting Parameter Values to Account for Number of Fingers**

The basic problem addressed in this example is the need to pass 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.

## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

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.

The screenshot shows the 'Edit Object Properties' dialog box. At the top, there are buttons: OK, Cancel, Apply, Defaults, Previous, Next, and Help. Below these, the 'Apply To' section has a dropdown menu set to 'only current' and another dropdown set to 'instance'. The 'Show' section has three checkboxes: 'system' (unchecked), 'user' (checked), and 'CDF' (unchecked). Below this is a section with two buttons: 'Browse' and 'Reset Instance Labels Display'. The main area contains a table with three columns: 'Property', 'Value', and 'Display'. The rows are: 'Library Name' with value 'analogLib' and display 'off'; 'Cell Name' with value 'nmos' and display 'off'; 'View Name' with value 'ams' and display 'off'; and 'Instance Name' with value 'M0' and display 'off'. At the bottom, there is a section for 'User Property' with buttons 'Add', 'Delete', and 'Modify'. Below these buttons is a table with three columns: 'User Property', 'Master Value', 'Local Value', and 'Display'. The row is: 'fingers' with an empty 'Master Value' field, '10' in the 'Local Value' field, and 'both' in the 'Display' field.

Property	Value	Display
Library Name	analogLib	off
Cell Name	nmos	off
View Name	ams	off
Instance Name	M0	off

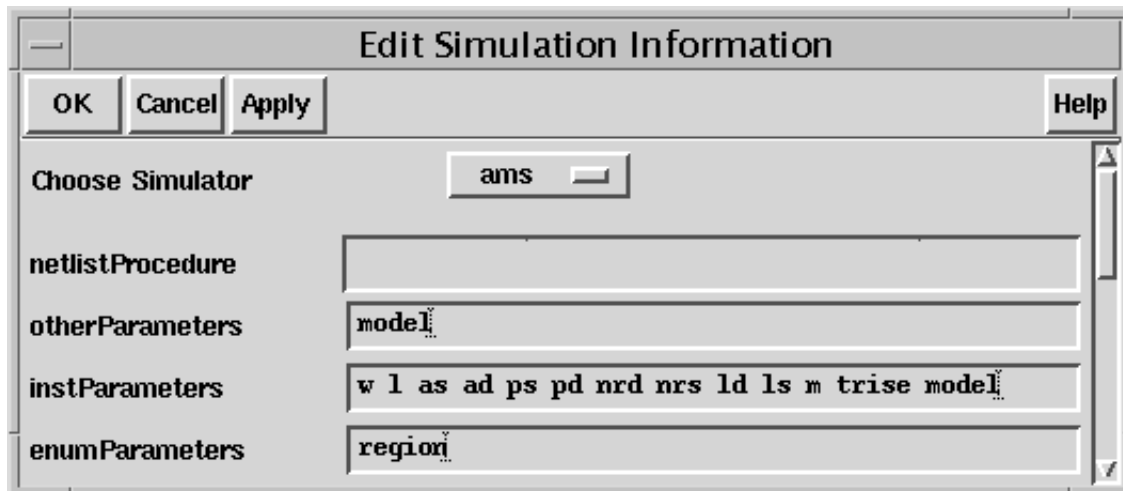
User Property	Master Value	Local Value	Display
fingers		10	both

## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

---

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 of the ams Simulation Information.



Notice, however, that the *fingers* property does not appear in the *instParameters* list. 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* then 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
```

## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

---

```
;; it up. But it is there on the instance as a normal CDBA
;; 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
;; =====

netlistId = amsGetNetlistId()
formatterId = netlistId->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.

## Example: Using Symbols that Represent Verilog Test Code

The goal in this example is to place a symbol representing a piece of Verilog test code in a schematic, and to have the test code inserted into the netlist. The symbol, which has no pins, is just a vehicle for the test code.

There are several steps involved in setting 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;

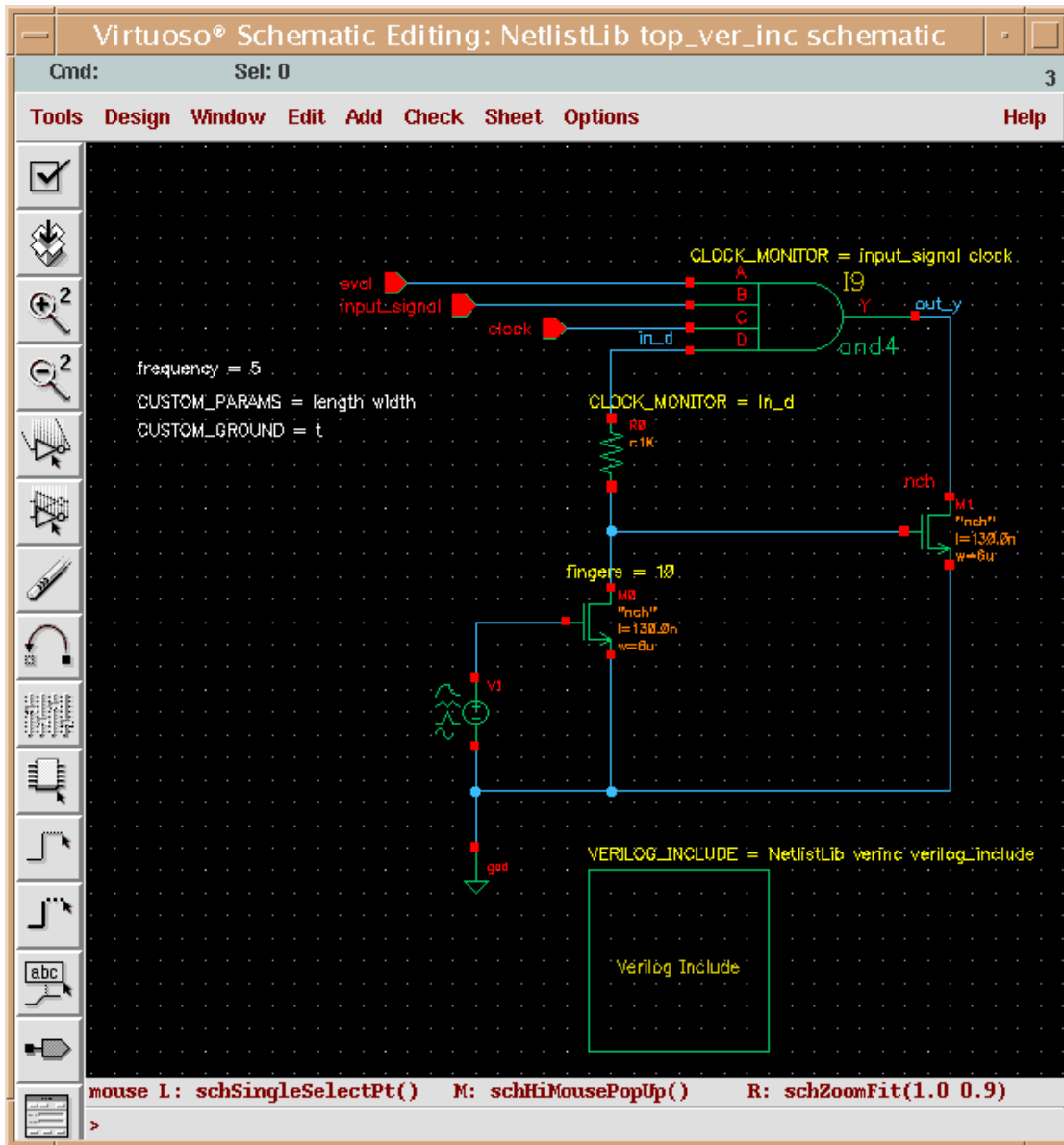
`ifdef CHECK_INPUT_TRANSITIONS
always @(posedge(TCOff||in_d) or negedge(TCOff||in_d))
    if (eval==1'b1)
        $display($stime," WARNING: %m in_d transition (evaluate is active)");
always @(posedge(TCOff||out_y) or negedge(TCOff||out_y))
    if (eval==1'b1)
        $display($stime," INFO: %m out_y 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.

## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

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 simple square symbol labeled Verilog Include.



3. Select the placed symbol and open the Edit Object Properties form for it. 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

## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

---

value NetlistLib verinc verilog\_include, leaving out the punctuation. Apply the changes. After these steps, the form looks like this.

**Edit Object Properties**

OK Cancel Apply Defaults Previous Next Help

Apply To: ☒ only current ☐ instance

Show: ☐ system ☒ user ☐ CDF

Browse Reset Instance Labels Display

Property	Value	Display
Library Name	NetlistLib	off <input type="checkbox"/>
Cell Name	verinc	off <input type="checkbox"/>
View Name	symbol	off <input type="checkbox"/>
Instance Name	I1	off <input type="checkbox"/>

Add Delete Modify

User Property	Master Value	Local Value	Display
VERILOG_INC..		NetlistLib verinc	both <input type="checkbox"/>

4. 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) )
          (error errmsg))))))
```



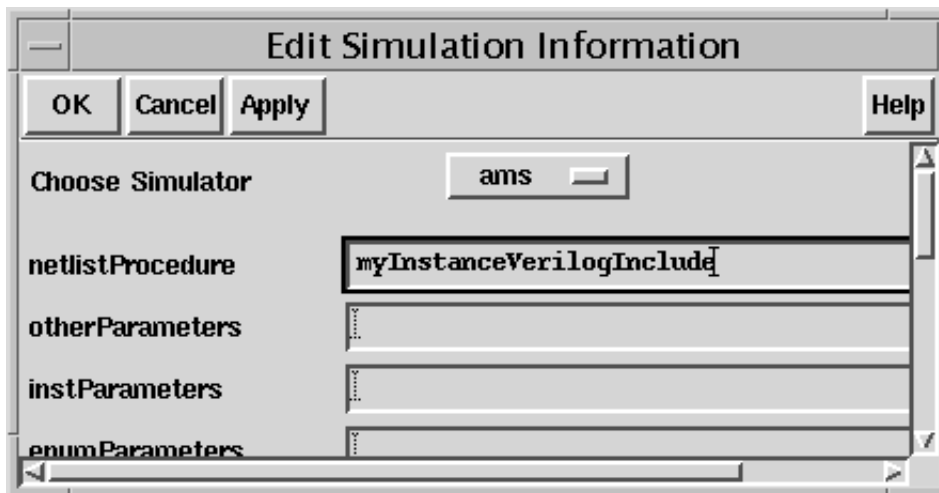
## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

---

```
(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
```

5. Open the Edit Component CDF form for the `netlistLib.verinc` cell, then from that form open the Edit Simulation Information form. Choose the *ams* simulator and add the name of the overriding netlisting procedure in the *netlistProcedure* field. After these steps, the form looks like this:



6. Click OK, in the Edit Component CDF form.
7. 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))
```

## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

---

```
(* 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 ----
```

### Example: Using CDF Instance Parameters to Define Inherited Connections

This example illustrates how to establish inherited connections on programmable nodes in such a way 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, use the following steps.

1. Determine what object properties are to be used as input to the netlisting function.

## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

---

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 590, 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.

## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

---

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,

<b>Bulk node connection</b>	VPOS!
<b>Substrate connection</b>	sub!
<b>Deep NWell connection</b>	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 the 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. Specify the instance master in the *Library Name* and *Cell Name* fields.

The form expands to display the information for that master.

- c. Ensure that *CDF Type* is set to `Base`.

- d. Scroll down to the Simulation Information section and click *Edit*.

The Edit Simulation Information form appears.

- e. Ensure that `ams` appears in the *Choose Simulator* field.

## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

- f. Scroll down to the *extraTerminals* field and enter the name of the function, using the following format.

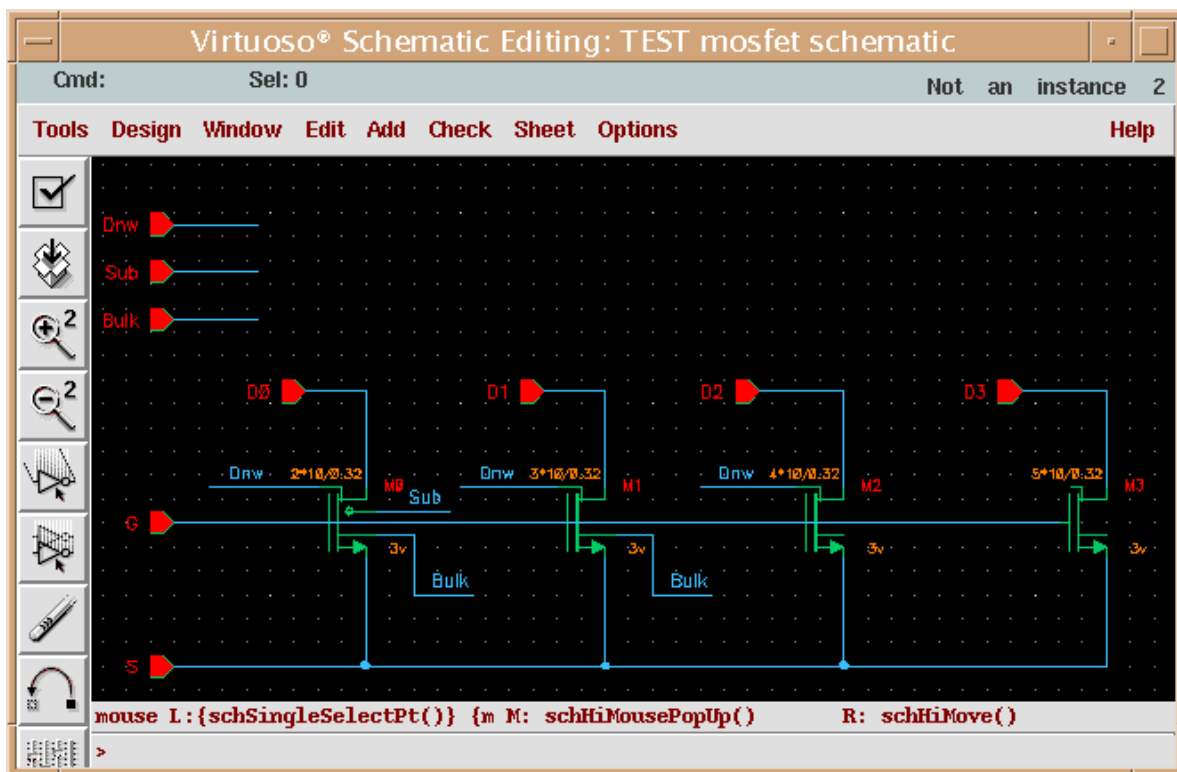
```
FUNCTION AMSnmos_dnw_inhExtraTerminals
```

- g. Click *OK* in the Edit Simulation Information form.
- h. Click *OK* in the Edit Component CDF form.

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.

## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

---

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.

Notice how 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! "; *)
  \dnw!_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! "; *)
  \bulk!_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
```

## Data Objects Supported for Netlisting

The data used for netlisting is organized into objects. The AMS netlister objects include netlister, formatter, cellview, parameter, and instance. Each of these is described in detail in the following sections.

### 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. However, the values of the fields of the netlister object can change with every netlist run as the netlister rereads the `ams.env` file.

Running a command like

```
A_netlisterId->?
```

you find that the netlister object contains the following fields. With the exception of the `vlog` and `vhdl` fields, each field corresponds to an `ams.env` variable described in [Appendix A, “Variables for `ams.env` Files.”](#)

Field	Values	Corresponding <code>ams.env</code> Variable
<code>lsbMsb</code>	boolean t/nil	<u><code>amsLSB_MSB</code></u>
<code>scalarizeInstances</code>	boolean t/nil	<u><code>amsScalarInstances</code></u>
<code>includeInstCDFParams</code>	boolean t/nil	<u><code>includeInstCdfParams</code></u>
<code>excludeParams</code>	<code>l_params</code>	<u><code>amsExcludeParams</code></u>
<code>expScalingFactor</code>	<code>'no   'sci   'dec</code>	<u><code>amsExpScalingFactor</code></u>
<code>modifyParamScope</code>	<code>'no   'warn   'yes</code>	<u><code>modifyParamScope</code></u>
<code>vlog</code>	The ID of the Verilog-AMS formatter object.	
<code>vhdl</code>	Always <code>nil</code> . No VHDL formatter is supported.	

## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

---

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_netlistId->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 netlist 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. However, the values of the fields of the formatter object can change with every netlist run as the netlister rereads the `ams.env` file.

Running the command

```
A_formatterId->?
```

you find that the formatter object contains the fields listed in the next two tables. Some of the fields correspond to `ams.env` variables described in [Appendix A, “Variables for `ams.env` Files.”](#)

**Table 13-2 Fields of the Formatter Object**

Field	Values	Corresponding <code>ams.env</code> Variable
<code>comments</code>	<i>string</i>	None
<code>headers</code>	<i>string</i>	None
<code>ifdefLanguageExtensions</code>	<code>boolean t/nil</code>	<u><code>ifdefLanguageExtensions</code></u>
<code>useDefparam</code>	<code>boolean t/nil</code>	<u><code>useDefparam</code></u>
<code>includeFiles</code>	<i>l_strings</i>	<u><code>includeFiles</code></u>
<code>paramDefVals</code>	<i>l_paramValPairs</i>	<u><code>paramDefVals</code></u>
<code>paramGlobalDefVal</code>	<i>paramGlobalDefVal</i>	<u><code>paramGlobalDefVal</code></u>
<code>netlister</code>	<i>A_netlistId</i>	None.



## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

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 13-1](#) on page 301.

**Table 13-3 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)
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. Notice that 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, try to use such indirectly collected information sparingly.

Field	Values	Meaning of the Value
libName	<i>string</i>	Name of the library.
cellName	<i>string</i>	Name of the cell.
viewName	<i>string</i>	Name of the cellview.
ID	DB???	ID of the cellView
primitive	boolean t/nil	Whether the master is a Spectre primitive. (Ports must be printed by order for instances of primitives.)
compName	<i>string</i>	componentName field from the simInfo.
termOrder	<i>l_strings</i>	List of terminals for the component. (The list can change for each instance.)
ports	<i>l_ports</i>	Not yet supported. Always returns nil.
parameters	<i>l_parameters</i>	Collected indirectly. The parameters 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 or track the `ams.env` variables 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 = amsGetNetlist()
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	<i>string</i>	Name of the parameter (as used in the Verilog-AMS netlist).
cdfName	<i>string</i>	Name of the parameter (as used in the CDBA).
type	<i>string</i>	Type of the parameter (as used when declaring the parameter).

## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

Field	Value	Meaning of the Value
dbType	symbol int / float, string / ael / aelNoNum / aelNum	Type of the parameter (as found in the DB/ CDF).
value	<i>string, integer, or float</i> (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	boolean 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	boolean 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. Note that 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 = amsGetNetlistter()
```

## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

```
vlog = netId->vlog  
vlog->instanceProc = 'myPrintInstance
```

Running these statements, you find that the instance object contains the following fields. Notice that some of these fields return information that is collected indirectly, and is therefore more costly of processing time and memory. For efficient netlisting, try to use such indirectly collected information sparingly.

Field	Value	Meaning of the Value
id	DB	instanceId
name	<i>string</i>	Name of the instance.
range	<i>l_integers</i>	<code>'( x_left x_right )</code> , nil for scalars
master	<i>A_masterCellViewId</i>	Collected indirectly. The cellViewId of the instance master.
masterName	<i>t_masterName</i>	Collected indirectly. The master name determined after applying the algorithm discussed in <a href="#">“componentName”</a> on page 589.
attributes	<i>l_attributes</i>	Collected indirectly. The attributes of the instance. Always returns nil.
parameters	<i>l_parameters</i>	Collected indirectly. The parameters of the instance.
ports	<i>l_ports</i>	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.

## SKILL Functions Supported for Netlisting

The SKILL functions specifically developed for use in custom netlisting procedures can be divided into those that replicate the default netlisting behavior and those that perform lower-level helping functions. These two varieties are described in [“Default Netlisting Procedures”](#) on page 334 and [“Netlisting Helper Functions”](#) on page 344.

## Default Netlisting Procedures

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

The default netlisting procedures take into account information, such as the values of `ams.env` variables, that might be required to create a netlist. For that reason, customized netlist procedures often run default netlisting procedures after setting up the appropriate data.

The default netlisting procedures are described in the following pages. They are:

- [amsPrintAttributes](#) on page 334
- [amsPrintInstance](#) on page 336
- [amsPrintInstanceMasterName](#) on page 338
- [amsPrintInstanceParameters](#) on page 339
- [amsPrintInstancePorts](#) on page 340
- [amsPrintParameters](#) on page 342

### **amsPrintAttributes**

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

#### **Description**

Default netlisting procedure for printing the attributes of the object specified by `A_objectId`. Attributes are printed in accordance with the value of the `ams.env` `ifdefLanguageExtensions` variable.

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

---

Omitting this attribute...	Can result in...
<code>library_binding</code>	Binding an instance to a library other than that intended by the schematic

---

## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

---

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

---

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

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

### Arguments

<code>A_formatterId</code>	ID of the formatter object.
<code>A_objectId</code>	ID of the object. Currently, an <code>instanceId</code> is the only value supported for <code>A_objectId</code> .

### Value Returned

<code>t</code>	Attributes of the object were printed.
<code>nil</code>	Attributes of the object were not printed.

### Example

In this example, assume that you have defined a function called `isDigitalGate`, which determines whether a gate is a digital gate. You use that function to determine whether to

## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

---

print the attributes to the netlist. If the attributes do need to be printed, you use the default function `amsPrintAttributes` to do the work.

```
;; Do not print attributes for instances of digital gates.
(defun DontPrintInstanceAttributesForDigitalGates (formatterId objectId)
  ;; Determine if it is an instance of a digital gate.
  digitalGate = isDigitalGate(objectId->masterName)
  (if (digitalGate == nil) then
    amsPrintAttributes(formatterId objectId)
  );; if
)

;; =====
;; Set up area
;; =====
netlistId = amsGetNetlistId()
formatterId = netlistId->vlog
formatterId->attributesProc = 'DontPrintInstanceAttributesForDigitalGates
```

### **amsPrintInstance**

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

### **Description**

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

If you override this function, you must:

- Develop an algorithm for obtaining the master name
- Handle the `library_binding`, `view_binding`, `elaboration_binding`, `inh_conn_prop_name`, `inh_conn_def_value`, `passed_mfactor`, `supplySensitivity`, `groundSensitivity`, and `cds_net_set` attributes
- Determine how to print instance parameters
- Take into account the `ams.env` variables `amsScalarInstances`, `iterInstExpFormat`, and `useDefparam`

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



## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

---

#### Arguments

<i>A_formatterId</i>	ID of the formatter object.
<i>A_cellviewId</i>	ID of the cellview object.
<i>A_instanceId</i>	ID of the instance object.

#### Value Returned

<i>t</i>	Instance was printed.
<i>nil</i>	Instance was not printed.

#### Example

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

```
;; =====
;; A custom netlist procedure to generate $display for listed signals.
;; =====
(defun myInstanceSignalMonitor (formatterId cellviewId instanceId)
  ;; Call the default instance procedure
  amsPrintInstance(formatterId cellviewId instanceId)
  (progn
    ;; Check for property CLOCK_MONITOR on the instance
    ;;
    (when instanceId->id->CLOCK_MONITOR
      (setq signal_list (parseString instanceId->id->CLOCK_MONITOR))
      monitor_signal = car(signal_list)
      print_debug_signals = 0
      (if (monitor_signal != nil) then
        amsPrint(formatterId "\n// Debug signals at every clock transition")
        amsPrint(formatterId "always @(posedge(clock) or negedge(clock))\n")
        amsPrint(formatterId "begin")
        amsPrint(formatterId "  $display($stime, 'Signal values are:');")
        print_debug_signals = 1
      ); if
      ;; Display the listed signals
      (while (monitor_signal != nil)
        display_signal = strcat("  $display(' monitor_signal ": %b', " mon
itor_signal ");")
        amsPrint(formatterId display_signal)
        signal_list = cdr(signal_list)
        monitor_signal = car(signal_list)
      )
    )
  )
)
```

## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

---

```
(if (print_debug_signals == 1) then
  amsPrint(formatterId "\nend \n")
); if
) ; when
) ; progn
) ; defun
;; =====
;; Set up area
;; =====
netlistId = amsGetNetlistId()
formatterId = netlistId->vlog
;; Override the printing of instance netlist procedure
formatterId->instanceProc = 'myInstanceSignalMonitor
```

### amsPrintInstanceMasterName

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

### Description

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

### Arguments

<i>A_formatterId</i>	ID of the formatter object.
<i>A_cellviewId</i>	ID of the attribute object.
<i>A_instanceId</i>	ID of the instance object.

### Value Returned

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

### Example

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

## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

---

```
(setq MYNetlister (amsGetNetlister))
;; Override the function that prints the master name.
MYNetlister->vlog->instanceMasterNameProc = 'MYInstMasterNameProc
(defun MYInstMasterNameProc ( formatterId cellViewId instanceId )
  ;; Check the name of the instance "fingers" property. Use the value
  ;; to generate the name of the master.
  (setq numFingers instanceId->id->fingers)
  (if (numFingers != nil) then
    ;; print the custom instance master name.
    (amsPrint formatterId (sprintf nil "\n%s_%d"
      instanceId->masterName numFingers) )
    else
    ;; print using the default instance master name netlist procedure.
    amsPrintInstanceMasterName(formatterId cellViewId instanceId)
  ); if
t
) ; defun
```

The MYInstMasterNameProc procedure generates and prints master names like nmos\_1 or nmos\_3 if the instance of nmos has 1 or 3 fingers and a master name like capacitance if the instance of capacitance has no property called fingers.

### amsPrintInstanceParameters

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

#### Description

Default netlisting procedure for printing instance parameters. This function uses the `ams.env useDefparam` variable to determine how to netlist the instance parameters.

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

If you do override the `amsPrintInstanceParameters` function, be aware that you must take into account the `ams.env useDefparam` variable, and, when the instance is an iterated instance, the `amsScalarInstances` variable.

#### Arguments

*A\_formatterId* ID of the formatter object.

*A\_cellviewId* ID of the cellview object.

## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

---

*A\_instanceId* ID of the instance object.

#### **Value Returned**

*t* Instance parameters were printed.

*nil* Instance parameters were not printed.

#### **Example**

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

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

#### **amsPrintInstancePorts**

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

#### **Description**

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

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

## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

---

#### Arguments

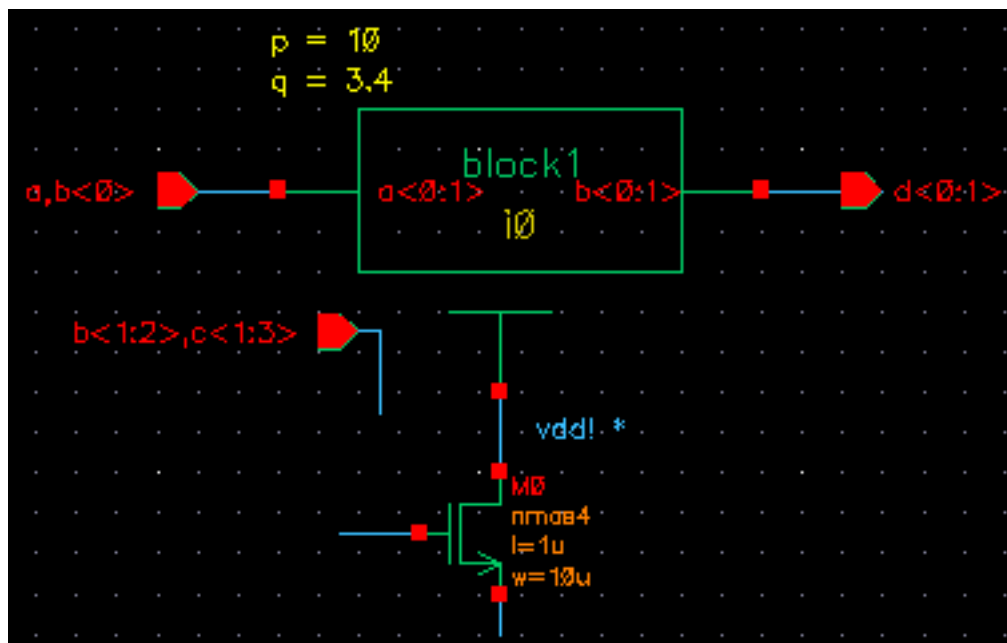
<i>A_formatterId</i>	ID of the formatter object.
<i>A_instanceId</i>	ID of the instance object.
<i>x_iteration</i>	Iteration number specifying a particular iterated instance. A value of -1 has a special meaning, indicating that the port list of an iterated instance is not to be split among the iterations of an instance. The default value for this argument is -1.

#### Value Returned

<i>t</i>	Port list was printed.
<i>nil</i>	Port list was not printed.

#### Example

You have the following schematic to be netlisted.



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

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

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

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

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

## **amsPrintParameters**

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

### **Description**

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

### **Arguments**

`A_formatterId`                      ID of the formatter object.

`A_cellViewId`                      ID of the cellview object.

### **Value Returned**

`t`                                      Parameter declarations were printed.

`nil`                                    Parameter declarations were not printed.

### **Example**

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

```
;; print the default CBN parameters  
amsPrintParameters(formatterId cvId)  
(let (newParam printInfo)
```

## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

---

```
;; Check for property CUSTOM_PARAMS on the CBN
;;
(when cvId->id->CUSTOM_PARAMS
  (setq paramList (parseString cvId->id->CUSTOM_PARAMS))
  newParam = car(paramList)
  printInfo = 0
  ;; Send informative messages to the log file
  (if (newParam != nil) then
    (amsInfo formatterId "Adding custom CBN parameters to the netlist...\n"
)
    printInfo = 1
  ); if

  ;; Display the listed parameters
  (while (newParam != nil)
    amsPrint(formatterId "\nparameter ")
    mappedParamName = amsMapName(formatterId cvId newParam 'parameter)
    amsPrint(formatterId mappedParamName)
    amsPrint(formatterId "= 0;")
    paramList = cdr(paramList)
    newParam = car(paramList)
  ); while
  (if (printInfo == 1) then
    (amsInfo formatterId "Done.\n")
  ); if
); when
); let
```

## Netlisting Helper Functions

The netlisting helper functions are described in the following pages. They are:

- [amsError](#) on page 344
- [amsGetInstanceName](#) on page 345
- [amsGetNetlister](#) on page 347
- [amsGetUniqueName](#) on page 348
- [amsInfo](#) on page 349
- [amsMapName](#) on page 350
- [amsPrint](#) on page 351
- [amsPrintInstanceParameter](#) on page 354
- [amsPrintParameter](#) on page 356
- [amsWarning](#) on page 357

### **amsError**

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

### **Description**

Helper function that prints *t\_msg* in the form of an error message and increments the error count. The message is added to the log file. Calling this function causes netlisting to fail, although processing to detect further netlisting problems continues (unless the incremented error count exceeds the value specified by the `ams.env amsMaxErrors` variable).

### **Arguments**

*A\_formatterId*                      ID of the formatter object.

*t\_msg*                                Message to be printed. If you want newline characters to appear in the message, you must include them in the message.



## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

---

#### **Value Returned**

<code>t</code>	String was printed.
<code>nil</code>	String was not printed.

#### **Example**

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

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

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

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

#### **amsGetInstanceName**

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

#### **Description**

Helper function that returns the print name of the instance specified by `A_instanceId` or the print name of the specified iteration of the instance. This function returns the name in the format specified by the `ams.env iterInstExpFormat` and `instClashFormat` variables.

## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

---

#### **Arguments**

<i>A_formatterId</i>	ID of the formatter object.
<i>A_instanceId</i>	ID of the instance object.
<i>x_iteration</i>	Iteration number specifying a particular iterated instance.

#### **Value Returned**

<i>t_instanceName</i>	Print name of the specified instance.
<i>nil</i>	Print name was not retrieved.

#### **Example**

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

```
;; Customize the parameter "r" for resistor to be 4K always.
(defun myPrintInstanceParameters (formatterId cvId instanceId)
  (if (instanceId->masterName == "resistor") then
    (amsPrint formatterId "#(")
    ;; Go through the list of parameters for resistor
    (foreach param instanceId->parameters
      (unless (equal param (car instanceId->parameters))
        (amsPrint formatterId ",")
      ); unless
      ;; Print only the parameter called "r"
      (if (param->name == "r") then
        ;; Compute my_new_value
        my_new_value = "4K"
        ;; Set the value of "r" to new value
        param->value = my_new_value
        instanceName = (amsGetInstanceName formatterId instanceId)
        ;; Call the helper function to print the parameter
        amsPrintInstanceParameter(formatterId instanceName param)
      )
    );foreach
    (amsPrint formatterId ")")
  ); if
  ;; For any instance whose masterName is NOT "resistor", print its
  ;; parameters in the default way using the default print function.
  ;;
  (if (instanceId->masterName != "resistor") then
```

## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

---

```
        amsPrintInstanceParameters(formatterId cvId instanceId)
    )
); defun
```

#### **amsGetNetlister**

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

#### **Description**

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

For information about the netlister object, see “[Netlister Object](#)” on page 327.

#### **Arguments**

None

#### **Value Returned**

<code>A_netlisterId</code>	The ID of the current netlister.
<code>nil</code>	The ID was not obtained.

#### **Example 1**

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

```
netlisterId = amsGetNetlister()
;; Get the Verilog-AMS formatterId.
vlogFormId = netlisterId->vlog
;; Override the default comment printing function.
vlogFormId->commentsProc = 'MyCommentsProc
```

#### **Example 2**

You enter the following into the CIW.

## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

---

```
netlisterID=amsGetNetlister()
```

AMS Designer returns the ID, in a format similar to

```
ams:28164120
```

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

```
netlisterID->??
```

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

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

### **amsGetUniqueName**

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

### **Description**

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

### **Arguments**

<i>A_formatterId</i>	ID of the formatter object.
<i>s_objectType</i>	Kind of object to be named. The <i>s_objectType</i> value can be 'net, 'instance, or 'alias.

### **Value Returned**

<i>t_objectName</i>	Generated unique name.
<i>nil</i>	Unique name was not generated.

### ***Example***

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

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

### **amsInfo**

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

### ***Description***

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

### ***Arguments***

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

### ***Value Returned***

<i>t</i>	Message was printed.
<i>nil</i>	Message was not printed.

### ***Example***

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

```
netlistId = amsGetNetlistId()
;; Get the Verilog-AMS formatterId.
vlogFormId = netlistId->vlog
```

## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

---

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

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

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

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

### amsMapName

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

### Description

Helper function that checks whether the specified name is a legal Verilog-AMS identifier. If the name is legal, the function simply returns the name. If the name is not a valid Verilog-AMS identifier, the identifier is mapped. If the mapped name collides with a name that is already in use, the mapped name is further collision mapped according to the settings of the `ams.env` `netClashFormat` and `instClashFormat` variables.

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

### Arguments

<i>A_formatterId</i>	ID of the formatter object.
<i>A_cellViewId</i>	Cellview object to which the name belongs.
<i>t_name</i>	Name to be checked and, if necessary, mapped.
<i>s_objectType</i>	Kind of object referred to by the name to be checked. The <i>s_objectType</i> value can be 'net, 'instance, 'alias, or

## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

---

'other. The default value is 'other. If *s\_objectType* is omitted, name collision checks are not performed.

#### **Value Returned**

<i>t_mappedName</i>	The name passed in, or, if necessary, a mapped transformation of the name.
<i>nil</i>	No name was returned.

#### **Example**

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

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

#### **amsPrint**

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

#### **Description**

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

#### **Arguments**

<i>A_formatterId</i>	ID of the formatter object.
<i>t_msg</i>	String to be written to the netlist. If you want newline characters to appear in the netlist, you must include them in the string.

## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

---

*s\_sectionId* Symbol representing the buffer into which *t\_msg* is printed. The possible values for *s\_sectionId* are:  
'INCLUDES\_LIST, 'MODULE\_INTERFACE,  
'PORT\_DECLARATION, 'PARAMETER\_DECLARATION,  
'SIGNAL\_DECLARATION, 'VLOG\_INSTANCES, and  
'END\_MODULE.

If you omit the *s\_sectionId*, the default value is determined from the netlist procedure field being overwritten. For a list of the default *s\_sectionId* value for each field see [Table 13-4](#) on page 352.

#### Value Returned

*t* Message string was written to the netlist.

*nil* Message string was not written to the netlist.

#### Default s\_sectionId Values

If you omit the *s\_sectionId* option on the `amsPrint` function, the default value is determined from the *formatterId* field being overwritten. The defaults are listed in the following table. Note that because the default function for the `moduleProc` field (`amsPrintModule`) cannot be overridden, that field is not included in the table. The corresponding netlist sections listed in the third column are illustrated in [Figure 13-1](#) on page 301.

**Table 13-4 Default s\_sectionId Values**

If the <i>formatterId</i> field being overridden is...	Then the default value for <i>s_sectionId</i> is...	Corresponding to this netlist section...
<i>attributesProc</i>	'VLOG_INSTANCES when <i>A_objectId</i> is an <i>instanceId</i> .	Instances
<i>commentsProc</i>	'INCLUDES_LIST	Includes list
<i>headersProc</i>	'INCLUDES_LIST	Includes list
<i>instanceMasterNameProc</i>	'VLOG_INSTANCES	Instances
<i>instanceParametersProc</i>	'VLOG_INSTANCES	Instances
<i>instancePortsProc</i>	'VLOG_INSTANCES	Instances



**Table 13-4 Default s\_sectionId Values, *continued***

<b>If the formatterId field being overridden is...</b>	<b>Then the default value for s_sectionId is...</b>	<b>Corresponding to this netlist section...</b>
instanceProc	'VLOG_INSTANCES	Instances
parametersProc	'PARAMETER_DECLARATION	Parameter declarations

### **Example 1**

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

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

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

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

### **Example 2**

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

```
netlistId = amsGetNetlistId()
;; Get the Verilog-AMS formatterId.
vlogFormId = netlistId->vlog
; The default s_sectionId of myPrintInstance is set to 'VLOG_INSTANCES'
; because myPrintInstance is set on the instanceProc field.
vlog->instanceProc = 'myPrintInstance
;; My function to print instances
(defun myPrintInstance (formatterId cellViewId instanceId)
(amsPrint formatterId "\n//Comment 1 in 'VLOG_INSTANCES\n")
(amsPrint formatterId "\n//Comment 2 in 'INCLUDES_LIST\n" 'INCLUDES_LIST)
```

## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

---

```
(amsPrintInstance formatterId cellViewId instanceId)
);defun
```

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

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

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

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

//Comment 2 in 'INCLUDES_LIST
//Comment 2 in 'INCLUDES_LIST
module comparator ( inp,inn,out );
input    inp;
input    inn;
output   out;

parameter chivalue=5.0;

//Comment 1 in 'VLOG_INSTANCES
pmos4 #(.l(3u), .region("triode"), .w(40u))
(* integer library_binding = "amslib";
integer cds_net_set[0:0]= {"bulk_n"};
integer bulk_n = "cds_globals.\vdd! "; *)
M11 ( net92, cds_globals.,nd! , net79, cds_globals.\vdd! );

//Comment 1 in 'VLOG_INSTANCES
isource #(.type("dc"), .dc(cds_globals.idc)) (*
integer library_binding = "analogLib"; *) I3 ( vref1,
cds_globals.,nd! );

endmodule
```

### amsPrintInstanceParameter

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

## **Description**

Helper function that prints the instance parameter specified by *A\_parameterId*, using the `ams.env useDefparam` variable to determine whether to print a `defparam` or an instance parameter override.

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

## **Arguments**

<i>A_formatterId</i>	ID of the formatter object.
<i>t_instanceName</i>	Name of the instance that has the parameter to be printed.
<i>A_parameterId</i>	ID of the parameter to be printed.

## **Value Returned**

<i>t</i>	Instance parameter was printed.
<i>nil</i>	Instance parameter was not printed.

## **Example**

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

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

## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

---

```
;; Set the value of "r" to new value
param->value = my_new_value
instanceName = (amsGetInstanceName formatterId instanceId)
;; Call the helper function to print the parameter
amsPrintInstanceParameter(formatterId instanceName param)
)
);foreach
(amsPrint formatterId ")")
); if
;; For any instance whose masterName is NOT "resistor", print its
;; parameters in the default way using the default print function.
;;
(if (instanceId->masterName != "resistor") then
  amsPrintInstanceParameters(formatterId cvId instanceId)
)
); defun
```

### amsPrintParameter

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

### Description

Helper function that prints the parameter specified by *A\_parameterId*.

### Arguments

<i>A_formatterId</i>	ID of the formatter object.
<i>A_parameterId</i>	ID of the parameter to be printed.

### Value Returned

t	Parameter was printed.
nil	Parameter was not printed.

### Example

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

## Virtuoso AMS Environment User Guide

### Producing Customized Netlists

---

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

### **amsWarning**

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

### **Description**

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

### **Arguments**

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

### **Value Returned**

<i>t</i>	Warning message was printed.
<i>nil</i>	Warning message was not printed.

### **Example**

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

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

## **Virtuoso AMS Environment User Guide**

### **Producing Customized Netlists**

---

```
parentheses and continuing.\n")  
);defun
```

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

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

---

## Variables for ams.env Files

---

The variables and values in `ams.env` files specify the basic behavior of the AMS netlister and AMS Design Prep. This appendix describes the mechanism used to load `ams.env` files and what these variables do.

This appendix contains the following sections:

- [How AMS Designer Determines the Set of Variables](#) on page 360
- [Why AMS Designer Uses ams.env Files, Not .cdsenv Files](#) on page 360
- [List of ams.env Variables](#) on page 362
- [Detailed Descriptions of ams.env Variables](#) on page 366

## How AMS Designer Determines the Set of Variables

The environment variables used by the AMS netlister and AMS Design Prep are stored in `ams.env` files. A file containing the default values is shipped with the software in the `share/cdssetup/amsDirect` directory. You can customize these default values by placing `ams.env` files with altered default values in any of the locations listed in the `setup.loc` file shipped in the `share/cdssetup` directory of the Cadence installation hierarchy. Any `ams.env` files found in the locations listed in the `setup.loc` file are progressively loaded, starting from the bottom of the list and working upward, with values for variables in subsequent files overwriting values for the same variables in previous files. This merging process produces the *base set* of values for `ams.env` variables. Note that any `ams.env` file found in the working directory (“.”), is *not* included in establishing the base set.

After the base set is established, the `ams.env` file in the working directory is read. After the run directory is chosen, any `ams.env` file found there is read. The variables in the working directory and run directory `ams.env` files together constitute the *local set* of values.

Immediately after the `ams.env` files are read, the current set of values consists of the values from the base set of variables overlaid by the values from the local set of variables. If the GUI in an AMS Designer tool is then used to modify the current set of values, AMS Designer writes the changes into the local (working directory and run directory) `ams.env` files. When writing the changes, AMS Designer compares the current value of each variable with its corresponding value in the base set. If the current and base values are different, the variable is written to one of the local `ams.env` files. If the current value is the same as that found in the base set, the variable is not written.

Variables are written to the local `ams.env` files as follows:

Location of <code>ams.env</code> file	Included variables
working directory	<code>amsDirect</code> , <code>amsDirect.vhdl</code> , <code>amsDirect.vlog</code>
run directory	<code>amsDirect.prep</code> , <code>amsDirect.simcntl</code>

## Why AMS Designer Uses `ams.env` Files, Not `.cdsenv` Files

AMS Designer uses the values of the variables found in `ams.env` files to control the behavior of the AMS Designer tools. Be aware that the similarly-named variables found in `.cdsenv` files have no effect at all on AMS Designer. AMS Designer does not read the variables in `.cdsenv` files because the `ams.env` file mechanism provides the following advantages.

- The `ams.env` files are read every time an AMS Designer tool performs an operation, not just once as the `.cdsenv` files are read. The repeated reads ensure that each tool uses the current values, even if those values are changed by a different tool. This approach



## Virtuoso AMS Environment User Guide

### Variables for ams.env Files

---

ensures, for example, that if you change an AMS option from the CIW, the new value is used even if you then run AMS Designer from the hierarchy editor.

- Each time AMS Designer writes the `ams.env` variables, it splits them into two files: an `ams.env` file in the working directory and an `ams.env` file in the run directory. This arrangement allows the variables used for netlisting and compiling to be in a location that is common to both AMS Design Prep and the AMS netlister (running from either the CIW or the hierarchy editor). This arrangement also allows the variables used for `cds_globals` module processing, elaboration, and simulation to be customized in each run directory.

## 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 tool, 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 366.

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	hdlVarFile	string	" "
amsDirect	includeInstCdfParams	boolean	nil
amsDirect	logFileName	string	"ams_direct.log"
amsDirect	modifyParamScope	cyclic	"no"
amsDirect	simRunDirLoc	string	" "
amsDirect	initFile	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	"lns/lns"
amsDirect.prep	use5xForVHDL	boolean	t
amsDirect.prep	useNcelabNowarn	boolean	t
amsDirect.prep	useNcelabSdfCmdFile	boolean	t
amsDirect.prep	useNcsimNowarn	boolean	t
amsDirect.prep	useSimVisScriptFile	boolean	t
amsDirect.prep	vlogGroundSigs	string	"gnd!"
amsDirect.prep	vlogSupply0Sigs	string	" "
amsDirect.prep	vlogSupply1Sigs	string	" "
amsDirect.prep	wfDefaultDatabase	boolean	nil
amsDirect.prep	wfDefInstCSaveLvl	int	1

## Virtuoso AMS Environment User Guide

### Variables for ams.env Files

---

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	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	ncelabLoadplil	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	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	ncelabOmichecklvl	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

## Virtuoso AMS Environment User Guide

### Variables for ams.env Files

---

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	boolean	nil
amsDirect.prep	ncelabUseExtendtcheckreferencelimit	boolean	nil	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	scaddlglblopts	string	" "		
amsDirect.simcntl	scaddltranopts	string	" "		
amsDirect.simcntl	scanotate	cyclic	"sweep"		
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"		

## Virtuoso AMS Environment User Guide

### Variables for ams.env Files

---

amsDirect.simcntl	scIteratio	string	" "
amsDirect.simcntl	scmacromod	boolean	nil
amsDirect.simcntl	scmaxiters	int	5
amsDirect.simcntl	scmaxnotes	int	5
amsDirect.simcntl	scmaxrsd	string	" "
amsDirect.simcntl	scmaxstep	string	" "
amsDirect.simcntl	scmaxwarn	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	"0.50"
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	useScaddlgloblopts	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

## Virtuoso AMS Environment User Guide

### Variables for ams.env Files

---

amsDirect.simcntl	useScwritefinal	boolean	t
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	ieee1364	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	nopragsawarn	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

The next sections discuss each of the ams . env file variables in detail. The sections are arranged alphabetically by variable.

## aliasInstFormat

Specifies the format to be used to create instances of the `cds_alias` module.

### Syntax

```
amsDirect.vlog aliasInstFormat string "format"
```

### Values

*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 <i>No – Print Errors</i> 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 <i>Yes – Print Warnings</i> 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 <i>Yes – Silently</i> value used in the GUI.

### Example

An example of conflicting bus ranges in CDBA data is shown below:

```
a<0:7>  
a<7:6>  
a<5:0>  
a<2:4>
```

The same example in Verilog-AMS is shown below:

```
a[0:7]  
{a[7],a[6]}  
{a[5],a[4],a[3],a[2],a[1],a[0]}  
a{2:4}
```



## Virtuoso AMS Environment User Guide

### Variables for ams.env Files

---

#### 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<sup>®</sup>-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 <i>No – Print Errors</i> 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 <i>Yes – Print Warnings</i> 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 <i>Yes – Silently</i> 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

## Virtuoso AMS Environment User Guide

### Variables for ams.env Files

---

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 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 <i>No – Print Errors</i> 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 <i>Yes – Print Warnings</i> 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 <i>Yes – Silently</i> value used in the GUI.

### Description

To comply with AMS 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

## Virtuoso AMS Environment User Guide

### Variables for ams.env Files

---

#### How Verilog-AMS Handles Non-Unique Identifiers, *continued*

---

Objects sharing a name	AMS netlister action
net, parameter	Net identifier maps to <i>netName_netclash</i>
net, module terminal	Net identifier maps to <i>netName_netclash</i> . (However, no mapping occurs when the net and module terminal are connected to each other.)
net, cell	Net identifier maps to <i>netName_netclash</i>
instance, net	Instance identifier maps to <i>instName_instclash</i>
instance, parameter	Instance identifier maps to <i>instName_instclash</i>
instance, module terminal	Instance identifier maps to <i>instName_instclash</i>
instance, cell	Instance identifier maps to <i>instName_instclash</i>

---

## 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 <i>No – Print Errors</i> 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 <i>Yes – Print Warnings</i> value used in the GUI.
yes	Overdeclares any sparse buses. The AMS netlister does not issue a warning. This value corresponds to the <i>Yes – Silently</i> value used in the GUI.

### Description

Sparse buses do not comply with AMS 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

An example of a sparse bus in CDBA data is

```
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:

## Virtuoso AMS Environment User Guide

### Variables for ams.env Files

---

```
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 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 environment.
nil	Specifies that certain properties used in legacy VHDL modules are <i>not</i> to be supported by the AMS environment.

### Description

The following legacy properties are supported by the AMS 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"
```

## Values

*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 CDBA namespace. Any included views that are created or imported by the CIW must be accompanied by a shadow CDBA. The default value is an empty string.

## Virtuoso AMS Environment User Guide

### Variables for ams.env Files

---

#### Example

```
amsDirect.vlog amsDefinitionViews string "symbol verilog"
```

## **amsEligibleViewTypes**

Specifies the cellview types that trigger netlisting.

### **Syntax**

```
amsDirect.vlog amsEligibleViewTypes string "list"
```

### **Values**

*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"
```

### **Values**

*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 [“Specifying Parameters to be Excluded from Netlisting”](#) on page 137.

## 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.

For example, Verilog-XL does not support scaling factor suffixes. If you intend to use Verilog-XL, you can use this variable so the AMS netlister expands suffixes when it generates 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 <sup>x</sup> )	AEL	Verilog-AMS	Spectre	SKILL	cdsSpice
Y	Yotta	10 <sup>24</sup>	See note below.				
Z	Zetta	10 <sup>21</sup>	See note below.				
T	Tera	10 <sup>12</sup>	yes	yes	yes	yes	yes
G	Giga	10 <sup>9</sup>	yes	yes	yes	yes	yes

## Virtuoso AMS Environment User Guide

### Variables for ams.env Files

#### Scaling Factor Suffixes and Target Simulators, *continued*

Suffix		Scaling Factor ( $e^x$ )	AEL	Verilog-AMS	Spectre	SKILL	cdsSpice
M	Mega	$10^6$	yes	yes	yes	yes	yes
ME	Mega	$10^6$	yes			yes	yes
K	Kilo	$10^3$	yes	yes	yes	yes	yes
k	kilo	$10^3$	yes		yes	yes	yes
%	percent	$10^{-2}$	yes		yes	yes	
c	percent	$10^{-2}$			yes		
m	milli	$10^{-3}$	yes	yes	yes	yes	yes
u	micro	$10^{-6}$	yes	yes	yes	yes	yes
n	nano	$10^{-9}$	yes	yes	yes	yes	yes
p	pico	$10^{-12}$	yes	yes	yes	yes	yes
f	femto	$10^{-15}$	yes	yes	yes	yes	yes
a	atto	$10^{-18}$	yes	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**

<code>t</code>	Orders the bits as [LSB : MSB] when constructing buses.
<code>nil</code>	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
```

### **Values**

<i>maxErrors</i>	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 <i>Print informational messages</i> field on the <i>Netlister</i> 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"
```

### **Values**

*file*                      The analog control file to be used. If *file* is specified with an absolute path, the analog 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"
```

## 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 Cadence 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 <code>library_binding = "basic"</code> attribute to automatically inserted instances of the <code>cds_alias</code> module. This is the default.
nil	Does not add the <code>library_binding</code> attribute to instances of the <code>cds_alias</code> 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 Cadence 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 <code>view_binding = "functional"</code> attribute to automatically inserted instances of the <code>cds_alias</code> module. This is the default.
nil	Does not add the <code>view_binding</code> attribute to instances of the <code>cds_alias</code> 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 Design Prep.

### **Syntax**

```
amsDirect.prep cdsGlobalsLib string "lib_name"
```

### **Values**

<i>lib_name</i>	The library to hold the <code>cds_globals</code> module. The default is an empty string.
-----------------	------------------------------------------------------------------------------------------

### **Description**

AMS Design Prep 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 Design Prep to store the `cds_globals` module in the `myglobelib` library.

## **cdsGlobalsView**

Specifies the view for the `cds_globals` module created by AMS Design Prep.

### **Syntax**

```
amsDirect.prep cdsGlobalsView string "view_name"
```

### **Values**

<i>view_name</i>	The view to be used for the <code>cds_globals</code> module. The default is an empty string.
------------------	----------------------------------------------------------------------------------------------

### **Description**

AMS Design Prep 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 Design Prep 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 [“cgsGlobalsLib”](#) on page 391.)



## **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 Design Prep 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 <code>-ams</code> option. This is the default.
nil	Verilog files with the extensions <code>.vams</code> or <code>.va</code> are compiled with the <code>-ams</code> option. Verilog files with the extension <code>.v</code> are compiled without the <code>-ams</code> option. If a Verilog file is actually a link, the decision to use or omit the <code>-ams</code> 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 Design Prep 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 Design Prep runs in compile all mode.

### **Syntax**

```
amsDirect.prep compileExcludeLibs string "list_of_libraries"
```

### **Values**

*list\_of\_libraries*    A list of library names separated by white space. Libraries with these names are not considered when AMS Design Prep 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 Design Prep 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 Design Prep not to attempt to compile any cells that belong to either the `compiledLib` or the `readOnlyLib` library.

## compileMode

Specifies the conditions under which AMS Design Prep (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 <code>ncvlog</code> or <code>ncvhdl</code> compiler is to compile the netlist specified by the <code>master.tag</code> file for the view. If the master netlist is a Verilog, Verilog-A, or Verilog-AMS file, the <code>ncvhdl</code> compiler also compiles the first netlist found in files named <code>vhdl.vhms</code> or <code>vhdl.vhd</code> (in that order). If the master netlist is a VHDL or VHDL-AMS file, the <code>ncvlog</code> compiler also compiles the first netlist found in files named <code>verilog.vams</code> , <code>verilog.va</code> , <code>verilog.v</code> , or <code>veriloga.va</code> (in that order). These compilations occur whether or not the cell is netlisted in this run.

Note that AMS Design Prep 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 Design Prep 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).

## Virtuoso AMS Environment User Guide

### Variables for ams.env Files

---

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 Design Prep to compile only newly created netlists.

## **connectRulesCell**

Specifies the cell that contains the `connectrules` module.

### **Syntax**

```
amsDirect.prep connectRulesCell string "cell"
```

### **Values**

<i>cell</i>	The cell that contains the <code>connectrules</code> module. The default is <code>mixedsignal</code> .
-------------	--------------------------------------------------------------------------------------------------------

### **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"
```

### **Values**

<i>cell</i>	The cell that contains the <code>connectrules</code> module. The default is <code>ConnRules_5V_full</code> .
-------------	--------------------------------------------------------------------------------------------------------------

### **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"
```

### **Values**

<i>lib</i>	The library that contains the <code>connectrules</code> 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"
```

### **Values**

<i>view</i>	The cellview that contains the <code>connectrules</code> 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
```

## **detailedDisciplineRes**

Specifies the kind of discipline resolution to be used.

### **Syntax**

```
amsDirect.prep detailedDisciplineRes boolean t | nil
```

### **Values**

t	AMS Design Prep uses the detailed method of discipline resolution.
nil	AMS Design Prep 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"
```

### **Values**

<i>discipline</i>	The discipline to be used for discrete nets of otherwise unknown discipline. The default is <code>logic</code> .
-------------------	------------------------------------------------------------------------------------------------------------------

### **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.

## **excludeViewNames**

Specifies the names of cellviews that are not to be netlisted.

### **Syntax**

```
amsDirect.vlog excludeViewNames string "list_of_view_names"
```

### **Values**

*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"
```

### Values

<i>file</i>	An <code>hdl.var</code> file to be used with the <code>-hdlvar</code> option of the <code>ncvlog</code> , <code>ncelab</code> , and <code>ncsim</code> commands. If <i>file</i> is not specified, the <code>-hdlvar</code> 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 tool looks for an `hdl.var` file in the directory where that tool started. If there is no `hdl.var` file in that location, the tool 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 tools are likely to use different `hdl.var` files if they are not specified explicitly.

If you use a relative path, be aware that paths are relative to the directory where the tool starts. The `ncvlog` tool starts in the current working directory so the path is relative to that directory. However, the `ncelab` and `ncsim` tools start in the run directory so the path for them is relative to the run directory. As a consequence, the different tools are likely to use different `hdl.var` files.

To be sure that all the tools 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 Design Prep.

### 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 <code>templateFile</code> variable. For more information, see <a href="#">“templateFile”</a> on page 551.
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 <code>templateScript</code> variable. For more information, see <a href="#">“templateScript”</a> on page 552.

### 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 <i>IEEE-1364 Verilog Hardware Description Language Reference Manual</i> .
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 Design Prep 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 <code>`ifdef INCA</code> clauses in the netlist for attribute statements.
nil	Does not generate <code>`ifdef INCA</code> 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 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.

```
vsourcesrc #(.dc(3), .type("dc")) (*
integer library_binding = "analogLib"; *) V0 ( cds_globals.\vdd! ,
cds_globals.,nd! );
vsourcesrc #(.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.

```
vsourcesrc #(.dc(3), .type("dc"))
`ifdef INCA (* integer library_binding = "analogLib"; *) `endif
V0 ( cds_globals.\vdd! , cds_globals.,nd! );
vsourcesrc #(.dc(-3), .type("dc"))
`ifdef INCA (* integer library_binding = "analogLib"; *) `endif
V1 ( cds_globals.\vss! , cds_globals.,nd! );
```

## **incdir**

Specifies directories to be searched for files specified by the ``include` compiler directive.

### **Syntax**

```
amsDirect.vlog incdir string "dirs_to_search"
```

### **Values**

<i>dirs_to_search</i>	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 "l1-LevelOneDirl1-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) }"
```

### Values

*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 585.

## 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"
```

### Values

<i>path</i>	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"
```

### Values

<i>format</i>	All characters, except those listed below, are printed exactly as included in <i>format</i> . The following characters have the indicated special meanings.
%b	Original name of the instance
%%	Prints the % character
	The default value of <i>format</i> 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"
```

### Values

*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"
```

### **Values**

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 off 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
```

## Virtuoso AMS Environment User Guide

### Variables for ams.env Files

---

Tells AMS Design Prep 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"
```

### Values

<i>logFileName</i>	Specifies the name of the log file. The default is <code>ams_direct.log</code> .
--------------------	----------------------------------------------------------------------------------

### 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 70.

### 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.

## Virtuoso AMS Environment User Guide

### Variables for ams.env Files

---

- 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"
```

### **Values**

*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 <code>`celldefine</code> and <code>`endcelldefine</code> compiler directives to tag module instances as cell instances.
nil	Does not insert <code>`celldefine</code> and <code>`endcelldefine</code> 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
```

### **Values**

<i>maxErrors</i>	A positive integer. Halts compilation after this number of errors occur. The default is 50.
------------------	---------------------------------------------------------------------------------------------

### **Example**

```
amsDirect.vlog maxErrors int 50
```

Tells AMS Design Prep 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 Design Prep 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.

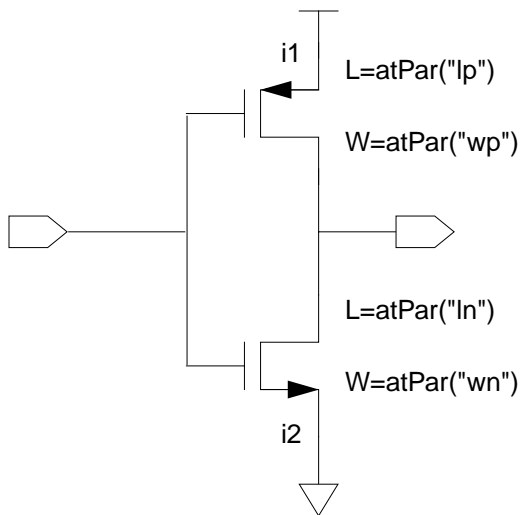
### **Description**

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  $l_n=3u$  and  $w_n=20u$  and that the pmos has defaults of  $l_p=3u$  and  $w_p=40u$ .



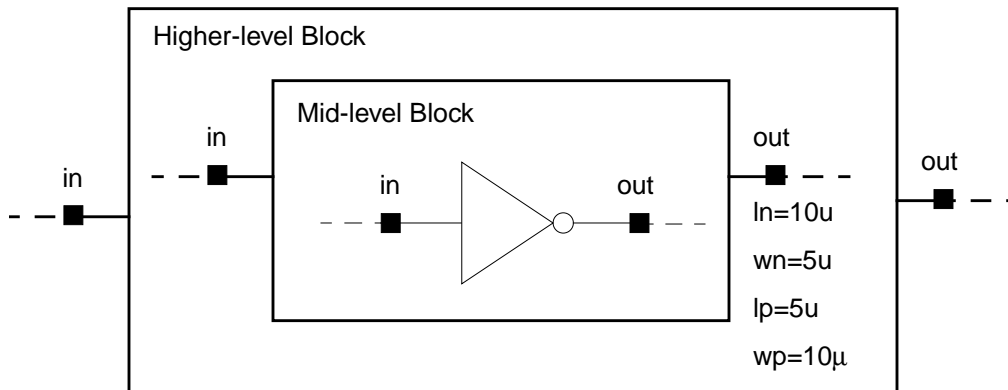
When this inverter is netlisted by the AMS netlister, it has an instance of an nmos and an instance of a pmos, each with parameters to be passed in:

```
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  $10\mu$  for the simulation
- $wn$  is set to  $5\mu$  for the simulation
- $lp$  is set to  $5\mu$  for the simulation
- $wp$  is set to  $10\mu$  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)) il ( 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 il ( 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  $3\mu$  for the simulation
- `wn` is set to  $20\mu$  for the simulation
- `lp` is set to  $3\mu$  for the simulation
- `wp` is set to  $40\mu$  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, you must 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 <code>-access -r-w-c</code> . This is the default.
Read	Appends the option <code>-access +r-w-c</code> .
Read/Write	Appends the option <code>-access +r+w-c</code> .
Connectivity	Appends the option <code>-access +r-w+c</code> .
All	Appends the option <code>-access +r+w+c</code> .

### **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"
```

### **Values**

*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"
```

### **Values**

<i>arguments</i>	One or more arguments to be passed to the <code>ncelab</code> 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 <code>Unit</code> 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 <code>specify</code> blocks. When a module contains a <code>specify</code> block with one or more module path delays, all structural and continuous assignment delays within that module (with the exception of <code>triereg</code> 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 <code>specify</code> blocks.

### Example

```
amsDirect.prep ncelabDelayMode cyclic "Unit"
```

Generates an `ncelab` command like the following.

```
ncelab amslib.top:config -delay_mode Unit
```

## **Virtuoso AMS Environment User Guide**

### Variables for ams.env Files

---

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**

The `ams.env` `ncelab` variables correspond to options used on the `ncelab` command. For information about an `ncelab` variable, see the information about the corresponding `ncelab` command option or options in chapter 8, of the *NC-Verilog Simulator Help*.

**Table A-1 Mapping ams.env ncelab Variables to ncelab Command Options**

<b>ams.env Variable</b>	<b>ncelab Command Option</b>
<code>ncelabDelayType</code>	<code>-MAxdelays, -MIndelays, -TYpdelays</code>
<code>ncelabDisableenht</code>	<code>-DISAble_enht</code>
<code>ncelabEpulseFiltering</code>	<code>-EPULSE_ONDetect, -EPULSE_ONEvent</code>
<code>ncelabEpulseNeg</code>	<code>-EPULSE_Neg</code>
<code>ncelabExpand</code>	<code>-EXPand</code>
<code>ncelabExtendtcheckdatalimit</code>	<code>-EXTEND_TCHECK_Data_limit</code>
<code>ncelabExtendtcheckreferencelimit</code>	<code>-EXTEND_TCHECK_Reference_limit</code>
<code>ncelabGenafile</code>	<code>-GENAfile</code>
<code>ncelabIeeel1634</code>	<code>-IEEel1364</code>
<code>ncelabInterconnmultisrc</code>	<code>-CAint</code>
<code>ncelabLibverbose</code>	<code>-LIBVerbose</code>
<code>ncelabLoadpli1</code>	<code>-LOADPli1</code>
<code>ncelabLoadvpi</code>	<code>-LOADVpi</code>
<code>ncelabLogFileAction</code>	<code>-LOGfile, -NOLog, -APpend_log</code>
<code>ncelabMaxErrors</code>	<code>-ERrormax</code>
<code>ncelabMessages</code>	<code>-MEssages</code>

## **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 <i>Allow mixed-case, escaped identifiers in VHDL</i> field, on the <i>VHDL</i> pane of the AMS Options window. As a result, the <code>-mixesc</code> option is passed on the <code>ncelab</code> command.
nil	Removes the checkmark, indicating that the <code>-mixesc</code> option is not to be passed on the <code>ncelab</code> 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

This variable, which is not intended for hand editing, contains the information needed to populate the Analog Model Files table.

### Syntax

```
amsDirect.prep  ncelabModelFilePaths    string  "model_files"
```

### Values

*model\_files*                      The status, paths, names, and sections of analog model files containing models used in the design. The default value is an empty string.

### Example

The variable

```
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"
```

populates the Analog Model Files table like this:

Enable	Model File	Section
<input checked="" type="checkbox"/>	/usr1/cds11752/alpha6/vh...	
<input type="checkbox"/>	\$PROJ3/SAR_A2D/spectrepr...	typical
<input type="checkbox"/>	\$PROJ3/SAR_A2D/spectrepr...	



## ncelabNeverwarn through ncelabVipdelay

The `ams.env` `ncelab` variables correspond to options used on the `ncelab` command. For information about an `ncelab` variable, see the information about the corresponding `ncelab` command option or options in chapter 8, of the *NC-Verilog Simulator Help*.

**Table A-2 Mapping ams.env ncelab Variables to ncelab Command Options**

<b>ams.env Variable</b>	<b>ncelab Command Option</b>
<code>ncelabNeverwarn</code>	<code>-NEVerwarn</code>
<code>ncelabNoautosdf</code>	<code>-NOAutosdf</code>
<code>ncelabNocopyright</code>	<code>-NOCopyright</code>
<code>ncelabNoipd</code>	<code>-NOIpd</code>
<code>ncelabNonegtchk</code>	<code>-NONEg_tchk</code>
<code>ncelabNonotifier</code>	<code>-NONotifier</code>
<code>ncelabNosource</code>	<code>-NOSource</code>
<code>ncelabNostdout</code>	<code>-NOSTdout</code>
<code>ncelabNoTchkMsg</code>	<code>-NO_TCHK_Msg</code>
<code>ncelabNoTchkXgen</code>	<code>-NO_TCHK_Xgen</code>
<code>ncelabNotimingchecks</code>	<code>-NOTimingchecks</code>
<code>ncelabNovitalaccl</code>	<code>-NOVitalaccl</code>
<code>ncelabNoVpdmsg</code>	<code>-NO_VPD_Msg</code>
<code>ncelabNoVpdXgen</code>	<code>-NO_VPD_Xgen</code>
<code>ncelabNowarn</code>	<code>-NOWarn</code>
<code>ncelabNtcWarn</code>	<code>-NTC_Warn</code>
<code>ncelabOmichecklvl</code>	<code>-OMicheckinglevel</code>
<code>ncelabPathpulse</code>	<code>-PAthpulse</code>
<code>ncelabPlinooptwarn</code>	<code>-PLINOOptwarn</code>
<code>ncelabPlinowarn</code>	<code>-PLINOWarn</code>
<code>ncelabPresrvResFn</code>	<code>-Preserve</code>
<code>ncelabPulseE</code>	<code>-PULSE_E</code>
<code>ncelabPulseIntE</code>	<code>-PULSE_INT_E</code>

## Virtuoso AMS Environment User Guide

### Variables for ams.env Files

**Table A-2 Mapping ams.env ncelab Variables to ncelab Command Options, *continued***

<b>ams.env Variable</b>	<b>ncelab Command Option</b>
ncelabPulseIntR	-PULSE_INT_R
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
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

## Virtuoso AMS Environment User Guide

### Variables for ams.env Files

**Table A-2 Mapping ams.env ncelab Variables to ncelab Command Options, *continued***

<b>ams.env Variable</b>	<b>ncelab Command Option</b>
ncelabUsePulseIntR	-PULSE_INT_R
ncelabUsePulseR	-PULSE_R
ncelabUseSdfprecision	-SDF_Precision
ncelabV93	-V93
ncelabVipdelay	-VIPDMAx, -VPIDMin

## **ncsimArguments**

Specifies additional arguments to be passed to the `ncsim` simulator.

### **Syntax**

```
amsDirect.prep ncsimArguments string "arguments"
```

### **Values**

*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 Design Prep 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**

The `ams.env` `ncsim` variables correspond to options used on the `ncsim` command. For information about an `ncsim` variable, see the information about the corresponding `ncsim` command option or options in chapter 9, of the *NC-Verilog Simulator Help*.

**Table A-3 Mapping ams.env ncsim Variables to ncsim Command Options**

<b>ams.env Variable</b>	<b>ncsim Command Option</b>
<code>ncsimEpulseNoMsg</code>	<code>-EPulse_no_msg</code>
<code>ncsimExtassertmsg</code>	<code>-EXTassertmsg</code>

## **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 <code>ncsimTcl</code> 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**

The `ams.env` `ncsim` variables correspond to options used on the `ncsim` command. For information about an `ncsim` variable, see the information about the corresponding `ncsim` command option or options in chapter 9, of the *NC-Verilog Simulator Help*.

**Table A-4 Mapping `ams.env` `ncsim` Variables to `ncsim` Command Options**

<b>ams.env Variable</b>	<b>ncsim Command Option</b>
<code>ncsimLoadvpi</code>	<code>-LOADVPi</code>
<code>ncsimLogFileAction</code>	<code>-LOGfile, -NOLOg, -APPEND_Log</code>
<code>ncsimMaxErrors</code>	<code>-ERrormax</code>
<code>ncsimMessages</code>	<code>-Messages</code>
<code>ncsimNeverwarn</code>	<code>-NEverwarn</code>
<code>ncsimNocifcheck</code>	<code>-NOCIIfcheck</code>
<code>ncsimNosource</code>	<code>-NOSource</code>
<code>ncsimNostdout</code>	<code>-NOSTdout</code>
<code>ncsimNowarn</code>	<code>-NOWarn</code>
<code>ncsimOmichecklvl</code>	<code>-Omicheckinglevel</code>
<code>ncsimPlinooptwarn</code>	<code>-PLINOOptwarn</code>
<code>ncsimPlinowarn</code>	<code>-PLINOWarn</code>
<code>ncsimProfile</code>	<code>-PROFile</code>
<code>ncsimProfthread</code>	<code>-PROFThread</code>
<code>ncsimRedmem</code>	<code>-REdmem</code>
<code>ncsimStatus</code>	<code>-STATUs</code>

## **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**

<code>t</code>	Opens the Tcl command window when the simulator runs.
<code>nil</code>	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**

The `ams.env` `ncsim` variables correspond to options used on the `ncsim` command. For information about an `ncsim` variable, see the information about the corresponding `ncsim` command option or options in chapter 9, of the *NC-Verilog Simulator Help*.

**Table A-5 Mapping `ams.env` `ncsim` Variables to `ncsim` Command Options**

<b>ams.env Variable</b>	<b>ncsim Command Option</b>
<code>ncsimUnbuffered</code>	<code>-UNbuffered</code>
<code>ncsimUpdate</code>	<code>-UPdate</code>
<code>ncsimUseAddArgs</code>	None. Determines whether additional specified arguments are used on the <code>ncsim</code> command.

## **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"
```

### **Values**

<i>arguments</i>	One or more arguments to be passed to the <code>ncvlog</code> 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**

<code>t</code>	The additional compiler arguments specified by the <code>ncvlogArguments</code> variable are used.
<code>nil</code>	The additional compiler arguments specified by the <code>ncvlogArguments</code> variable are <i>not</i> 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"
```

### Values

*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 CDBA cellviews in the hierarchy only if their HDL data is not synchronized with their CDBA data. This is the default.
all	Netlists all CDBA cellviews in the hierarchy, regardless of whether their HDL data is synchronized with their CDBA data.

### **Example**

```
amsDirect.prep netlistMode cyclic "all"
```

Tells AMS Design Prep (working through the AMS netlister) to netlist all the cellviews that can be netlisted.

## 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
```

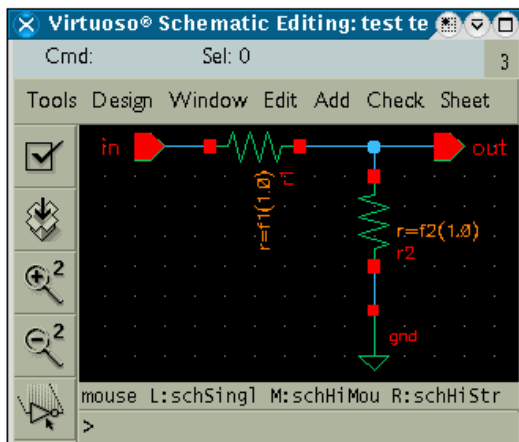
### Values

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.



The netlister uses equivalent macro references in the netlist.

```
resistor #(.r(\f1(1.0))) (* ... *) R1 ( ... );  
resistor #(.r(\f2(1.0))) (* ... *) R1 ( ... );
```



## **Virtuoso AMS Environment User Guide**

### **Variables for ams.env Files**

---

The referenced macros must be defined in an accessible location, as described in “Preparing to Netlist User-Defined Functions” on page 140.

## **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 <i>not</i> 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**

<code>t</code>	Prepares design units for access by the PLI routine <code>tf_nodeinfo</code> .
<code>nil</code>	Does not prepare design units for access by the PLI routine <code>tf_nodeinfo</code> . This is the default.

### **Example**

```
amsDirect.vlog nomempack boolean t
```

Tells AMS Design Prep to compile Verilog files with the `-nomempack` option. As a result, the generated command might look like this.

```
ncvlog -nomempack
```

## **nopragswarn**

Suppresses warning messages related to pragmas.

### **Syntax**

```
amsDirect.vlog nopragswarn 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"
```

### **Values**

*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

<i>type</i>	The type of <i>parameter_name</i> : integer or real.
<i>parameter_name</i>	A Verilog-AMS module parameter.
<i>value</i>	The default associated with <i>parameter_name</i> .

### 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 CDBA 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"
```

### **Values**

<i>value</i>	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 Design Prep to compile Verilog files with the `-pragma` option. As a result, the generated command might look like this.

```
ncvlog -pragma
```

## **processViewNames**

Specifies the names of cellviews that are to be netlisted.

### **Syntax**

```
amsDirect.vlog processViewNames string "list_of_view_names"
```

### **Values**

*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.

## **scaddlgloblopts**

Specifies options to be appended to the end of the `options` card in the generated simulation control file.

### **Syntax**

```
amsDirect.simcntl scaddlgloblopts string "options"
```

### **Values**

*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 scaddlgloblopts 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"
```

### **Values**

*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"
```

### **Values**

*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"
```

### **Values**

*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**

Unsupported by AMS Designer.

Specifies the degree of annotation for the transient analysis. For more information, see “Annotation Parameters,” in the “Transient Analysis (tran)” section, in chapter 5, of the *Virtuoso Spectre Circuit Simulator Reference*.

## **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
```

## **sccheckstmt**

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” section, in chapter 7, of the *Virtuoso Spectre Circuit Simulator User Guide*.

## **sccmin**

Specifies the minimum capacitance from each node to ground.

### **Syntax**

```
amsDirect.simcntl sccmin string "capacitance"
```

### **Values**

*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" |  
    "cdsspice" | "hspice" | "spiceplus"
```

### **Values**

spectre

spice2

spice3

cdsspice

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 <i>not</i> 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 <i>not</i> 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
```

### **Values**

<i>digits</i>	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**

<code>t</code>	Requests a method of evaluating VHDL-AMS <code>break</code> statements that is often faster than the default method. Under some circumstances, the method chosen by setting <code>scfastbreak</code> to <code>t</code> does not comply with the VHDL-AMS standard. Possible non-compliance with the standard arises when the <code>break</code> statement is associated with a discontinuity that causes a zero-delay <code>Q' ABOVE</code> event. The <code>Q' ABOVE</code> 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.
<code>nil</code>	Requests the <code>break</code> 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.

## **scgmin**

Specifies the minimum conductance across each nonlinear device.

### **Syntax**

```
amsDirect.simcntl scgmin string "conductance"
```

### **Values**

<i>conductance</i>	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"
```

### **Values**

<i>tolerance</i>	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 <code>ic</code> statements and <code>ic</code> parameters, and <code>ic</code> parameters override <code>ic</code> statements.
dc	Ignores any initial condition specifiers, and uses the DC solution.
node	Uses <code>ic</code> statements, and ignores <code>ic</code> parameters on capacitors and inductors.
dev	Uses <code>ic</code> parameters on capacitors and inductors, and ignores <code>ic</code> 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:l=1.0u"
```

In response, the generated simulation control file contains

```
ic 7=0 out=1 OpAmp1.comp=5 L1:l=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**

<i>ratio</i>	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 <i>not</i> 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**

<i>maxnotes</i>	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
```

## **scmaxrsd**

Specifies the threshold below which parasitic node reduction occurs.

### **Syntax**

```
amsDirect.simcntl scmaxrsd string "threshold"
```

### **Values**

<i>threshold</i>	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**

<i>maxwarn</i>	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
```

## **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 <code>mos_method = s</code> 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 543.

### Example

You set the variables

```
amsDirect.simcntl      scusemodeleval boolean t
amsDirect.simcntl      scmodelevaltype 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 0.50.

### **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
```

## **snotation**

Specifies the notation to be used when displaying real numbers.

### **Syntax**

```
amsDirect.simcntl snotation 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 snotation 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 <i>not</i> 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**

<i>threshold</i>	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**

<i>threshold</i>	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**

<i>tolerance</i>	The maximum relative tolerance for values computed in the last two iterations of a solution.
------------------	----------------------------------------------------------------------------------------------

### **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**

<i>resistance</i>	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**

<i>factor</i>	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**

<i>factor</i>	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**

<i>sim_con_file</i>	The path and name of the simulation control file to be included.
---------------------	------------------------------------------------------------------

### **Description**

AMS Design Prep

### **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 Design Prep

### **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**

<i>skipcount</i>	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**

<i>time</i>	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 <code>options speed</code> 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 <code>options speed = 1</code> to the analog simulation control file.
2	Writes <code>options speed = 2</code> to the analog simulation control file.
3	Writes <code>options speed = 3</code> to the analog simulation control file.
4	Writes <code>options speed = 4</code> to the analog simulation control file.
5	Writes <code>options speed = 5</code> to the analog simulation control file.
6	Writes <code>options speed = 6</code> 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
<i>Tran Analysis</i>	<i>Error preset</i>	<a href="#"><u>"scerrpreset"</u></a> on page 485
<i>Performance</i>	<i>Node reduction threshold</i>	<a href="#"><u>"scmaxrsd"</u></a> on page 501
<i>Convergence/Accuracy</i>	<i>Reltol</i>	<a href="#"><u>"screltol"</u></a> on page 518
<i>Convergence/Accuracy</i>	<i>Vabstol</i>	<a href="#"><u>"scvabstol"</u></a> on page 544

## Virtuoso AMS Environment User Guide

### Variables for ams.env Files

Pane	Field	For more information, see
Convergence/Accuracy	<i>labstol</i>	<a href="#">“sciabstol”</a> on page 490
Tran Convergence/Accuracy	<i>Lteratio</i>	<a href="#">“sclteratio”</a> on page 497
Tran Convergence/Accuracy	<i>Relref</i>	<a href="#">“screlref”</a> on page 517
Tran Convergence/Accuracy	<i>Integration method</i>	<a href="#">“scmethod”</a> on page 504
Tran Convergence/Accuracy	<i>Maxstep</i>	<a href="#">“scmaxstep”</a> on page 502

### 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
<i>Error preset</i>	moderate
<i>Node reduction threshold</i>	<Value defaulted>
<i>Reltol</i>	<Value defaulted>
<i>Vabstol</i>	1e-6
<i>labstol</i>	1e-12
<i>Lteratio</i>	<Value defaulted>
<i>Relref</i>	<Default value>
<i>Integration method</i>	<Default value>
<i>Maxstep</i>	<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 <i>not</i> 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**

<i>stop_time</i>	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**

<i>temperature</i>	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 <code>tc</code> , such as <code>tc1</code> , and <code>tc2</code> ) 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 `options 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
```

## **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
```

## **sctransave**

```
amsDirect.prep ncelabDelayType cyclic "Minimum"
```

## 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 <code>mos_method</code> option to the <code>options</code> statement. For guidance on setting the value of the <code>mos_method</code> option, see <a href="#">“scmodelevaltype”</a> on page 505.
nil	The simulator does <i>not</i> add the <code>mos_method</code> option to the <code>options</code> 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**

<i>tolerance</i>	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"
```

## **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 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**

<i>script_file</i>	The script file to be run at the beginning of simulation. If <i>script_file</i> uses a relative path, the <code>ncsim</code> tool 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

<i>text_file</i>	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 <code>headerText</code> variable has the value "file". For more information, see <a href="#">"headerText"</a> 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

<i>script_file</i>	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 <code>headerText</code> variable has the value "script". For more information, see <a href="#">"headerText"</a> 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 modules.

### **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 “-USe5x4vhdI Option” section of chapter 7, in the *Virtuoso AMS Simulator User Guide*.

## useDefparam

Controls the netlisting of parameters passed onto instantiated modules.

### Syntax

```
amsDirect.vlog useDefparam boolean t | nil
```

### Values

t	Generates one <code>defparam</code> 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 that Verilog-XL and other 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
```

## Virtuoso AMS Environment User Guide

### Variables for ams.env Files

---

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
```

## **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 <i>Suppress specific warnings</i> 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 <i>Use SDF command file</i> 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 <i>Use SDF command file</i> 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 <i>Suppress specific warnings</i> 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 <i>Suppress specific warnings</i> 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.

## **useScaddlgblopts**

Controls whether the list of additional options on the *Analog Solver* pane of the AMS Options window is used.

### **Syntax**

```
amsDirect.simcntl useScaddlgblopts boolean t | nil
```

### **Values**

t	Places a checkmark next to the <i>Additional options</i> 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 useScaddlgblopts 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 <i>Additional options</i> 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 <i>Set initial conditions</i> 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 <i>Read IC from file</i> 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 <i>Read nodesets from file</i> 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 <i>Include simulation control file</i> 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 <i>Write initial solution to file</i> 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 <i>Write final solution to file</i> 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 <i>Tcl input script</i> 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 <i>Tcl input script</i> 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.

## **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 <code>processViewNames</code> list.
nil	The AMS netlister determines which cellviews to process by consulting, in combination, the <code>amsEligibleViewTypes</code> list and the <code>excludeViewNames</code> 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.

## 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 <i>Print verbose messages during update</i> field on the <i>Verilog-AMS</i> 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**

<i>signal_list</i>	A list of signals to be declared, by default, as ground. The default is <code>gnd!</code> .
--------------------	---------------------------------------------------------------------------------------------

### **Description**

AMS Design Prep 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 Design Prep 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 vloglinedebug boolean t
```

Tells AMS Design Prep 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**

<i>signal_list</i>	A list of signals to be declared, by default, as supply0 wires. The default is an empty string.
--------------------	-------------------------------------------------------------------------------------------------

### **Description**

AMS Design Prep 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 Design Prep 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**

<i>signal_list</i>	A list of signals to be declared, by default, as supply1 wires. The default is an empty string.
--------------------	-------------------------------------------------------------------------------------------------

### **Description**

AMS Design Prep 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 Design Prep 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**

<i>database</i>	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 <code>all</code> appearing in the <i>Depth</i> 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 <code>amsDirect.prep wfDefInstCSaveLvl</code> 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**

<i>level</i>	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 <i>Currents at terminals or ports</i> label in the <i>Waveforms</i> 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 <i>Currents at terminals or ports</i> label in the <i>Waveforms</i> 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 <i>Voltages/Signals</i> label in the <i>Waveforms</i> 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 <i>Voltages/Signals</i> field in the <i>Waveforms</i> 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 <code>all</code> appearing in the <i>Depth</i> 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 <code>amsDirect.prep wfDefInstVSaveLvl</code> 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**

<i>level</i>	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.



---

## Updating Legacy SimInfo for Analog Primitives

---

Cadence<sup>®</sup> 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, with each set representing the formatting instructions for that device for a given simulator. For example, an nmos device to be used in netlists targeted for the Spectre simulator has information in the *spectre* section of the simInfo. If that nmos device is also used in netlists targeted for the cdsSpice simulator, it has cdsSpice-specific information in the *cdsSpice* section of the simInfo. Although the information for each simulator is unique, its basic purpose is the same: to enable a netlister to accurately present an instance of the given primitive device to the specified simulator.

Virtuoso AMS Designer uses the AMS netlister, which translates CDBA data into Verilog<sup>®</sup>-AMS netlists targeted for the AMS simulator. To support the AMS simulator, the simInfo for analog primitives now contains a section called *ams*. This appendix describes the information contained in the *ams* section and describes how the information affects the formatting of primitive devices in Verilog-AMS netlists produced by the AMS netlister. The AMS netlister does not consider whether the view being netlisted is a stop view or not before obeying the formatting instructions in the *ams* simInfo fields.

### The ams Fields

For traditional Cadence netlisters, which are usually written in Cadence SKILL language, a SKILL procedure that formats instances can be named in the *netlistProcedure* field for a device. The AMS netlister is not written in SKILL and cannot use a customized procedure for devices. Specifying a procedure does not affect the formatting of instances because the AMS netlister does not recognize the procedure. New fields provided in the *ams* section provide the 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`. If you are targeting the *cdsSpice* simulator, you specify this value with a parameter called `td`. In each case, these names (`delay` and `td`) are 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.

- [arrayParameters](#) on page 587
- [componentName](#) on page 589
- [enumParameters](#) on page 589
- [excludeParameters](#) on page 590
- [extraTerminals](#) on page 590
- [instParameters](#) on page 591
- [isPrimitive](#) on page 592
- [otherParameters](#) on page 594
- [propMapping](#) on page 594
- [referenceParameters](#) on page 595
- [stringParameters](#) on page 595
- [termOrder](#) on page 596

## 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* is the name of the arrayed parameter (must be a symbol or a string)

*start* is 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* is 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* is 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* is a string to append to the element counter

## Virtuoso AMS Environment User Guide

### Updating Legacy SimInfo for Analog Primitives

---

*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* is 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* is the value that must be matched by the *propertyName* value 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 *tcounter* and *vcounter*, 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( { p1s1, p2s2, ..., pm1sm,
        p1s2, p2s2, ..., 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 594 for details.

### **componentName**

This field, which contains the type of the component being instantiated, overrides the device master cell name. The *model* and *modelName* parameters also override the device master cell name. The precedence for determining the Verilog-AMS module name for the instance is

1. Value of *model* parameter, if available
2. Value of *modelName* parameter, if available
3. Value of *componentName* field in the *ams* section of the *simInfo*, if available
4. 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.

### **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.

### **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` `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 [“Specifying Parameters to be Excluded from Netlisting”](#) on page 137.

### **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

## Virtuoso AMS Environment User Guide

### Updating Legacy SimInfo for Analog Primitives

---

```
extraTerminals ::=  
  { ( nil name termName  
      direction directionType  
      netExpr netExpression ) }
```

#### Where

<i>termName</i>	is the name of the terminal (must be a string)
<i>directionType</i>	is the direction type of the terminal as specified in cv~>terminals~>direction  <i>directionType</i> can be "input", "output", or "inputOutput"
<i>netExpression</i>	is 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 596 for details.

#### 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, 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.

### 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 597.

The *AMS simInfo from Spectre* conversion tool does not fill in the *isPrimitive* field of the AMS simulation information.

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 Simulation Information**

OK Cancel Apply Help

Choose Simulator **ams**

**otherParameters** SimInfo

**instParameters** SimInfo

**enumParameters** SimInfo

**referenceParameters** SimInfo

**stringParameters** SimInfo

**arrayParameters** SimInfo

**componentName** SimInfo

**termOrder** SimInfo

**propMapping** SimInfo

**extraTerminals** SimInfo

**isPrimitive** SimInfo

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.

## **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 587 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

```
typairs 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 *typairs*, which are used to determine the upper range of the array (see “*arrayParameters*” on page 587) and the list of array elements that are possible in the array.

## **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 <code>acm</code> is found...	Use <code>mag</code> as the parameter name.
When the parameter <code>acp</code> is found...	Use <code>phase</code> as the parameter name.
When the parameter <code>td</code> is found...	Use <code>delay</code> as the parameter name.

---

The `cdsSpice` `simInfo` *propMapping* entry for this same device has no value. Thus, when generating a netlist for the `cdsSpice` simulator, the CDF names (`vdc`, `acm`, `acp`, and `td`) are used for these parameters.

### 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.

### **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())
```

The `cdsSpice` and `spectreS` (`spectreSocket`) simulators support programmable nodes, which are specified in the *termOrder* entry as the argument to `Progn` (i.e., `Progn (nodeName)`). `Progn` is not supported by the AMS netlister. Programmable nodes must be represented as *extraTerminals* in the *ams simInfo*. If `Progn` is encountered by the [AMS simInfo from Spectre](#) conversion tool, it issues a warning that manual editing of the *ams simInfo* is necessary.

## 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 [“Passing Model Names as Parameters”](#) on page 134.

## Converting an Existing Analog Primitive Library

Cadence provides a conversion tool that you can use to add *ams* simInfo information to an existing analog primitive library. This new information is derived from existing *spectre* simInfo data.

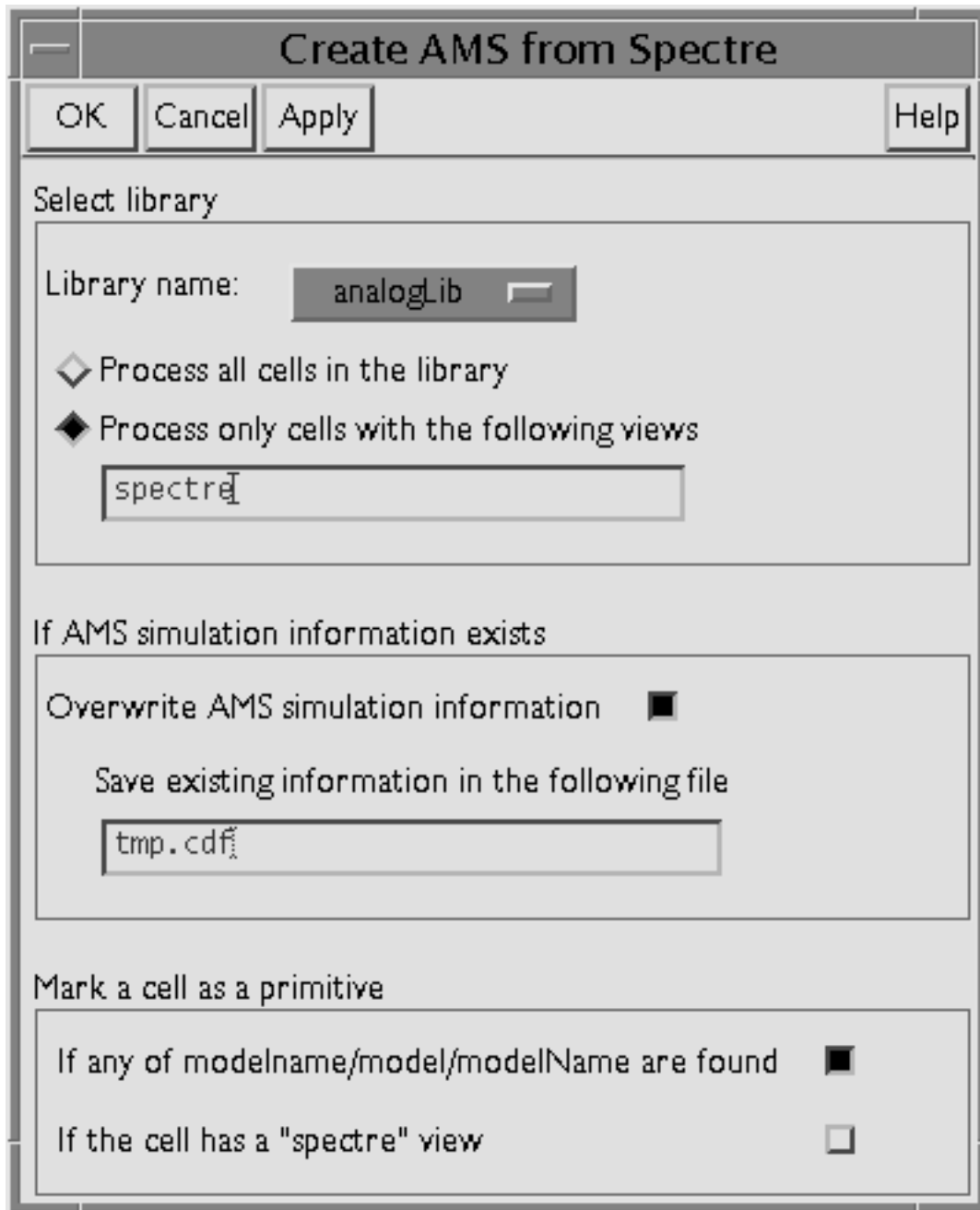
To use the conversion tool

1. From the CIW, choose *Tools – Conversion Tool Box*.

The Conversion Tool Box window appears.

2. Click *AMS simInfo from Spectre*.

The Create AMS from Spectre form appears.



**Create AMS from Spectre**

OK Cancel Apply Help

Select library

Library name: analogLib

☒ Process all cells in the library

☐ Process only cells with the following views

spectre

If AMS simulation information exists

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 either *Process all cells in the library* or *Process only cells with the following views*.

If you choose the latter, type in one or more views. Only cells that have at least one of the specified views are processed. Cells that do not have any of the specified views are unchanged and any existing *ams* simInfo is left untouched.

5. Enable *Overwrite AMS simulation information* if the library already has *ams* simInfo data.
6. In the *Save existing information in the following file* field, type the name of a backup file to be created by the conversion tool.

If necessary, you can use the backup file to restore the original *ams* simInfo.

7. Specify conditions for the conversion tool to use when determining whether a cell is a primitive.

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 manually set *isPrimitive* to the opposite value.

- a. Select *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. Select *If the cell has a "spectre" view* to have the netlister use this condition.

8. Click *OK*.

The *AMS simInfo from Spectre* conversion tool traverses the library, reporting which cells it creates *ams* simInfo data for and which ones it skips. When a cell is skipped, the *AMS simInfo from Spectre* conversion tool reports the reason.

The conversion tool 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 tool recognizes and converts the following analogLib procedures:

- *spectreCCPrim*



- spectreFsrcPrim
- spectreMindPrim
- spectreNportPrim
- spectrePolyCntrlPrim
- spectrePortPrim
- spectrePortSrcPrim
- spectrePwlsrcPrim
- spectreSCCPrim
- spectreSVCPPrim
- spectreSrcPrim
- spectreVandlSourcePrim
- spectreWindingPrim

The conversion tool 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 tool 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 tool does not fill in the *isPrimitive* field of the AMS simulation information. If the library you are converting contains components that are primitives, you need to set the value of the *isPrimitive* field for those components to *t*. For more information, see “[isPrimitive](#)” on page 592.

Finally, if the cell has a *spectre* view, the conversion tool 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 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.

If *spectre* simInfo data existed when the conversion tool 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.

---

## Designing for Virtuoso AMS Compliance

---

This appendix describes guidelines for creating schematic designs that the Virtuoso® AMS 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](#) on page 604
- [Terminals](#) on page 606
- [Buses](#) on page 607
- [Component Description Format](#) on page 608
- [Parameters](#) on page 608
- [Parameterized Cells](#) on page 609
- [VHDL-AMS Component Declarations](#) on page 610
- [Properties](#) on page 610

## Identifiers

So that your design works efficiently with the Virtuoso AMS 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 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 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

You should also 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 <i>netName_netclash</i>
net, terminal	Net identifier maps to <i>netName_netclash</i> . (However, no mapping occurs when the net and terminal are connected to each other.)
net, cell	Net identifier maps to <i>netName_netclash</i>
instance, net	Instance identifier maps to <i>instName_instclash</i>

## **VHDL-AMS: Handling of Non-Unique Identifiers, *continued***

---

<b>When these objects share a name</b>	<b>Then</b>
instance, parameter	Instance identifier maps to <i>instName_instclash</i>
instance, terminal	Instance identifier maps to <i>instName_instclash</i>
instance, cell	Instance identifier maps to <i>instName_instclash</i>

---

## **Verilog-AMS: Handling of Non-Unique Identifiers**

---

<b>When these objects share a name</b>	<b>Then</b>
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 <i>netName_netclash</i>
net, terminal	Net identifier maps to <i>netName_netclash</i> . (However, no mapping occurs when the net and terminal are connected to each other.)
net, cell	Net identifier maps to <i>netName_netclash</i>
instance, net	Instance identifier maps to <i>instName_instclash</i>
instance, parameter	Instance identifier maps to <i>instName_instclash</i>
instance, terminal	Instance identifier maps to <i>instName_instclash</i>
instance, cell	Instance identifier maps to <i>instName_instclash</i>

---

## **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 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 environment, library CDF information has no effect on netlisting.

(You can, however, use cell CDF. For information, see [“Using Cell Parameters”](#) on page 609.)

## Parameters

This section describes the guidelines for using inherited parameters, cell parameters, and parameter formats in the AMS 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 environment. Parameters can be passed from one level of the design hierarchy to a lower level by using `pPar` expressions.



## Using Cell Parameters

Be sure to follow these guidelines for cell parameters as you develop your design.

- 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<sup>®</sup> cellview outside of the AMS 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 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 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 environment to reanalyze and renetlist cells that use inconsistent methodologies.

## Properties

Your designs function most efficiently in the AMS 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 environment and should not be used. If you use these properties, the AMS environment omits them from the netlist. This process increases the run time.

---

<b>Property to avoid</b>	<b>Other Cadence netlisters that use the property</b>
<code>hnlVerilogCDFdefparamList</code>	Verilog
<code>hnlVerilogHandleRCdata</code>	Verilog
<code>verilogFormatProc</code>	Verilog
<code>verilogView</code>	Verilog
<code>vhdlArchitectureName</code>	VHDL
<code>vhdlNetlistType</code>	VHDL
<code>vhdlPortType</code>	VHDL
<code>vhdlSignalKind</code>	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.

## Virtuoso AMS Environment User Guide

### Designing for Virtuoso AMS Compliance

For information about how AMS Designer uses the `portOrder` property, see [“amsDefinitionViews”](#) on page 378.

Property to avoid unless required	Other Cadence netlisters that use the property
<code>portOrder</code>	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
<code>vhdlAttributeDefList</code>	VHDL
<code>vhdlComponentDecl</code>	VHDL
<code>vhdlFormalPortFuncName</code>	VHDL
<code>vhdlPackageComponents</code>	VHDL
<code>vhdlPackageNames</code>	VHDL

### Properties That Have No Special Meaning in the AMS Environment

The properties in this section have meaning for some netlisters, but have no special meaning in the AMS environment.

Property	Other Cadence netlisters that use the property
<code>algorithm</code>	Verilog
<code>c</code>	Verilog
<code>chargeDecay</code>	Verilog

## Virtuoso AMS Environment User Guide

### Designing for Virtuoso AMS Compliance

---

---

Property	Other Cadence netlisters that use the property
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 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

## Virtuoso AMS Environment User Guide

### Designing for Virtuoso AMS Compliance

---

---

<b>Property</b>	<b>Other Cadence netlisters that use the property</b>
<code>vhdlGenericDefList</code>	VHDL
<code>vhdlInitialValue</code>	VHDL
<code>vhdlResolveFunction</code>	VHDL
<code>vhdlScalarType</code>	VHDL
<code>vhdlVectorType</code>	VHDL

---

# **Virtuoso AMS Environment User Guide**

## Designing for Virtuoso AMS Compliance

---

---

## SKILL Functions and Customization Variables

---

This appendix describes the Cadence SKILL languages functions and customization variables associated with the AMS environment. The included sections are:

- [SKILL Functions](#) on page 616
- [Customization Variables](#) on page 635

## SKILL Functions

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

<b>SKILL Function</b>	<b>For information, see</b>
<code>amsCheckCV</code>	<a href="#"><u>amsCheckCV</u></a> on page 617
<code>amsIsPresent</code>	<a href="#"><u>amsIsPresent</u></a> on page 618
<code>amsNetlist</code>	<a href="#"><u>amsNetlist</u></a> on page 619
<code>amsProcessCellViews</code>	<a href="#"><u>amsProcessCellViews</u></a> on page 622
<code>amsUIOptionsForm</code>	<a href="#"><u>amsUIOptionsForm</u></a> on page 625
<code>amsUIRunNetlisterForm</code>	<a href="#"><u>amsUIRunNetlisterForm</u></a> on page 626
<code>ddsCvtAMSTranslateCell</code>	<a href="#"><u>ddsCvtAMSTranslateCell</u></a> on page 627
<code>ddsCvtAMSTranslateLib</code>	<a href="#"><u>ddsCvtAMSTranslateLib</u></a> on page 630
<code>ddsCvtToolBoxAMS</code>	<a href="#"><u>ddsCvtToolBoxAMS</u></a> on page 632
<code>vmsUpdateCellViews</code>	<a href="#"><u>vmsUpdateCellViews</u></a> on page 633



## amsCheckCV

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

### Description

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

### Arguments

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

### Value Returned

<i>l_numCount</i>	A list of two integers: the number of errors, and the number of warnings encountered while running AMS checks.
-------------------	----------------------------------------------------------------------------------------------------------------

### Example

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

```
amsCheckCV( cv t )
```

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

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

## **amsIsPresent**

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

### **Description**

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

### **Arguments**

None.

### **Value Returned**

t	AMS netlisting capability is included in the executable.
nil	AMS netlisting capability is <i>not</i> included in the executable.

### **Example**

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

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

## **amsNetlist**

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

### **Description**

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

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

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

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

### **Arguments**

<i>t_libName</i>	A string, which is the name of the library to process.
<i>t_cellName</i>	A string, which is the name of the cell to process. If <i>t_cellName</i> is left blank (with just " "), all the cells in the library are processed.
<i>t_viewName</i>	A string, which is the name of the view to process. If <i>t_viewName</i> is left blank (with just " "), all the views are processed.
<i>g_checkOnly</i>	The value <code>t</code> or <code>nil</code> . If <code>t</code> is specified, the checks run. If <code>nil</code> is specified, the checks do not run. If no value is specified, the value defaults to that of the <code>amsDirect.vlog_checkOnly</code>

environment variable. For additional information, see [“checkOnly”](#) on page 394.

*g\_netlist*                      The value `t` or `nil`. If `t` is specified, a Verilog-AMS netlist is generated. If *g\_netlist* is `nil`, no netlist is generated. If no value is specified, the value defaults to that of the `amsDirect.vlog checkAndNetlist` environment variable. For additional information, see [“checkAndNetlist”](#) on page 393.

*s\_netlistMode*                A symbol with the value ``incr` or ``all`. If ``incr` is specified, only new or revised cellviews are netlisted. For example, changing a symbol or the CDF for a device in a schematic and then requesting netlisting triggers netlisting for only affected cells

When ``all` is specified and netlisting is requested, every cell is netlisted. This is the default value.

*g\_compile*                    The value `t` or `nil`. If `t` is specified, the generated Verilog-AMS netlist is compiled. If *g\_compile* is `nil`, the netlist is not compiled.

If no value is specified, the default value depends on the value of the `amsDirect.vlog prohibitCompile` environment variable. When the value of the `prohibitCompile` variable is `t`, the default value for *g\_compile* is `nil`. When the value of the `prohibitCompile` variable is `nil`, the default value for *g\_compile* is `t`.

## Value Returned

`t`                                The function was successful.

`nil`                             The function failed.

## Example

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

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

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

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

## **Virtuoso AMS Environment User Guide**

### **SKILL Functions and Customization Variables**

---

To compile all the cellviews in mylib:

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

## amsProcessCellViews

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

### Description

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

### Arguments

<i>t_libName</i>	A string, which is the name of the library to process.
<i>t_cellName</i>	A string, which is the name of the cell to process. If <i>t_cellName</i> is left blank (with just " "), all the cells in the library are processed.
<i>t_viewName</i>	A string, which is the name of the view to process. If <i>t_viewName</i> is left blank (with just " "), all the views are processed.
<i>g_checkOnly</i>	The value <i>t</i> or <i>nil</i> . If <i>t</i> is specified, the checks run. If <i>nil</i> is specified, the checks do not run. If no value is specified, the value defaults to that of the <code>amsDirect.vlog checkOnly</code> environment variable. For additional information, see <a href="#">“checkOnly”</a> on page 394.
<i>g_netlist</i>	The value <i>t</i> or <i>nil</i> . If <i>t</i> is specified, a Verilog-AMS netlist is generated. If <i>nil</i> is specified, no netlist is generated. If no value is specified, the value defaults to that of the <code>amsDirect.vlog checkAndNetlist</code> environment variable. For additional information, see <a href="#">“checkAndNetlist”</a> on page 393.
<i>g_compile</i>	The value <i>t</i> or <i>nil</i> . If <i>t</i> is specified, the generated Verilog-AMS netlist is compiled. If <i>nil</i> is specified, the netlist is not compiled.

If no value is specified, the default value depends on the values of the `amsDirect.vlog prohibitCompile` and the `amsDirect.vhdl prohibitCompile` variables, as shown in the following table. When the value of both `amsDirect.vlog prohibitCompile` and `amsDirect.vhdl prohibitCompile` are set to `t`, the default value for `g_compile` is `nil`. When the value of one or both of `amsDirect.vlog prohibitCompile` and `amsDirect.vhdl prohibitCompile` are set to `nil`, the default value for `g_compile` is `t`.

*s\_netlistMode*

A symbol with the value `'incr` or `'all`. If `'incr` is specified, only new or revised cellviews are netlisted. For example, changing a symbol or the CDF for a device in a schematic and then requesting netlisting triggers netlisting for only affected cells.

When `'all` is specified and netlisting is requested, every cell is netlisted. This is the default value.

*s\_compileMode*

A symbol with the value `'whenNetlist` or `'all`. The `'whenNetlist` value specifies that only cellviews that are netlisted are compiled. The `'whenNetlist` value is the default.

The `'all` value specifies that all cellviews are compiled, whether newly netlisted or not.

## Value Returned

`t` The function was successful.

`nil` The function failed.

## Examples

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

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

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

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

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

## **Virtuoso AMS Environment User Guide**

### **SKILL Functions and Customization Variables**

---

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



## **amsUIOptionsForm**

`amsUIOptionsForm()`

### **Description**

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

### **Arguments**

None.

### **Value Returned**

None.

### **Example**

`amsUIOptionsForm()`

## **amsUIRunNetlisterForm**

`amsUIRunNetlisterForm( )`

### **Description**

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

### **Arguments**

None.

## ddsCvtAMSTranslateCell

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

### Description

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

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

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

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

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

### Arguments

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

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

## Notes

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

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

See [Appendix B, “Updating Legacy SimInfo for Analog Primitives”](#) for more details about AMS simulation information.

## Examples

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

## **Virtuoso AMS Environment User Guide**

### **SKILL Functions and Customization Variables**

---

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

## ddsCvtAMSTranslateLib

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

### Description

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

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

- *stringParameters*
- *referenceParameters*
- *enumParameters*
- *arrayParameters*
- *extraTerminals*

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

### Arguments

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

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

## Examples

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

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

## **ddsCvtToolBoxAMS**

`ddsCvtToolBoxAMS ( )`

### **Description**

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

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

### **Arguments**

None.

### **Examples**

`ddsCvtToolBoxAMS ( )`



## **vmsUpdateCellViews**

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

### **Description**

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

### **Arguments**

<i>lt_lib</i>	A string, which is the name of a library or a list of library names to look in for cellviews to update. If this argument is not specified (with just " ") or is specified as <code>nil</code> , all libraries defined in the <code>cds.lib</code> file are searched.
<i>lt_cell</i>	A string, which is the name of a cell or a list of cell names to be searched for update in the libraries. If this argument is not specified (with just " ") or is specified as <code>nil</code> , all cells are searched.
<i>lt_view</i>	A string, which is the name of a cellview or a list of cellview names to be searched for update. If this argument is not specified (with just " ") or is specified as <code>nil</code> , all views are searched.

### **Value Returned**

<code>t</code>	The function ran successfully.
<code>nil</code>	The function failed.

## Examples

This example updates the specified text cellview.

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

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

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

## Customization Variables

You can use variables to customize the operation of the AMS environment. For example, you can put these variables in a `.cdsinit` file or set their values in the CIW. The variables apply when you:

- 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 editing tools
- Create another cellview from a Verilog-AMS cellview
- Call the function `vmsUpdateCellViews`

For details, see the cross-references below.

---

Variable	For information, see
<code>schHdlNotCreatedDB</code>	<a href="#"><u>schHdlNotCreatedDB</u></a> on page 636
<code>schHdlParseUsingNcvhdl</code>	The “ <a href="#"><u>schHdlParseUsingNcvhdl</u></a> ” section, in Appendix A, of the “Virtuoso Schematic Composer VHDL Interface User Guide”
<code>schHdlUseVamsForVerilog</code>	<a href="#"><u>schHdlUseVamsForVerilog</u></a> on page 637
<code>vmsAnalysisType</code>	<a href="#"><u>vmsAnalysisType</u></a> on page 638
<code>vmsCreateMissingMasters</code>	<a href="#"><u>vmsCreateMissingMasters</u></a> on page 639
<code>vmsNcvlogExecutable</code>	<a href="#"><u>vmsNcvlogExecutable</u></a> on page 640
<code>vmsPortProcessing</code>	<a href="#"><u>vmsPortProcessing</u></a> on page 641
<code>vmsRunningInUI</code>	<a href="#"><u>vmsRunningInUI</u></a> on page 642
<code>vmsTemplateScript</code>	<a href="#"><u>vmsTemplateScript</u></a> on page 643
<code>vmsVerboseMsgLevel</code>	<a href="#"><u>vmsVerboseMsgLevel</u></a> on page 644

---

## **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:

<i>view_type</i>	The environment does not create database data for these HDL view types. In this release, the only values supported for <i>view_type</i> are VerilogAMSText and, when schHdlUseVamsForVerilog is set to t, VerilogText.
<i>nil</i>	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 hierarchy editor.

### **Example 1**

You use the following command in the CIW to identify the existing view types.

```
ddMapGetDataTypeInfo()
```

The returned information looks similar to

```
( "AHDNetlist" "AHDNetlist" "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.

## **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.

## **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 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 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:

<i>template_script</i>	The specified script is used to generate headers.
<i>nil</i>	A default header is used. It has the form <i>//Verilog-AMS HDL for libname, cellname viewname</i>

### **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
```

## **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:

<i>message_level</i>	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.
```

---

## Compiling Cadence-Provided Libraries

---

This appendix explains the purpose and use of the `amsLibCompile` tool.

- [Purpose of the `amsLibCompile` Tool](#) on page 646
- [Running the `amsLibCompile` Tool Manually](#) on page 646

## Purpose of the amsLibCompile Tool

The purpose of the amsLibCompile tool is to compile the Verilog modules in the following Cadence provided libraries:

- ahdLib
- basic
- bmslib
- rfLib
- sample

The amsLibCompile tool runs automatically during the post-load phase of the amsEnv kit installation process. Before the tool 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, nclog, is in the executable search path (\$PATH)

If libraries fail to compile during the installation process, or if the libraries need to be re-compiled for use with a different version of the NC tools, you can run amsLibCompile manually.

## Running the amsLibCompile Tool Manually

If the libraries listed in the previous section do not compile successfully during the installation process, or if you need to recompile the libraries for use with a different version of the LDV tools, you can run the amsLibCompile tool manually.

To run the amsLibCompile tool manually,

- You must have write permission to the libraries
- You must start the amsLibCompile tool from the root of the DFII installation hierarchy. In other words, you must start the tool in the directory that contains the `tools` directory

## Example

Assume there are two LDV installations available:

- `/ams/ldv50/s1`

## Virtuoso AMS Environment User Guide

### Compiling Cadence-Provided Libraries

---

■ /usr2/ldv51

Also assume that, of these two, your \$PATH includes only /ams/ldv50/s1.

1. Run the `amsLibCompile` command.

```
amsLibCompile
```

The tool 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 tool 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 tool adds this path to the list of LDV hierarchies.

4. When asked to do so, press *Enter* to continue:

The tool 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 tool 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 that they can later be used with the different versions of AMS Designer.

# **Virtuoso AMS Environment User Guide**

## Compiling Cadence-Provided Libraries

---