cadence®

OCEAN Reference

Product Version 6.1.6
November 2014

Contents

14 File Commands and Functions

15 OCEAN Commands in XL Mode

Preface

Open Command Environment for Analysis (OCEAN) lets you set up, simulate, and analyze circuit data without starting Virtuoso Analog Design Environment L, XL or GXL.

This manual describes OCEAN and the commands required to set up, simulate, and analyze circuit data using OCEAN. This manual assumes that you are familiar with analog design and simulation using the Virtuoso Analog Design Environment. You should also be proficient in Cadence® SKILL language programming.

The preface discusses the following:

Scope of this Manual

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

Important

Only the functions and arguments described in this manual are available for public use. Any undocumented functions or arguments are likely to be private and could be subject to change without notice. It is recommended that you check with your Cadence representative before using them.

Licensing in OCEAN

You need to have the `Analog_Design_Environment_L` licence to use OCEAN. For information on licensing, see *Virtuoso Software Licensing and Configuration Guide*.

Related Documents for OCEAN

OCEAN is based on the Virtuoso® SKILL programming language. The following manuals give you more information about the SKILL language and other related products.

Installation, Environment, and Infrastructure

- For information on installing Cadence products, see the *Cadence Installation Guide*.
- For information on the Virtuoso design environment, see the *Virtuoso Design Environment User Guide*.
- The *Cadence SKILL Language User Guide* describes how to use the SKILL language functions, the SKILL++ functions, and the SKILL++ object system (for object-oriented programming).
- The *Cadence SKILL Language Reference* provides descriptions, syntax, and examples for the SKILL and SKILL++ functions.
- The *Cadence SKILL++ Object System Reference* provides descriptions, syntax, and examples for the object system functions.

- The *Virtuoso Design Environment SKILL Reference* describes database SKILL functions, including data access functions.
- The *Virtuoso Design Environment SKILL Reference* describes database SKILL functions, including data access functions.
- The *Virtuoso Analog Design Environment L SKILL Language Reference* provides descriptions, syntax, and examples for the SKILL commands supported by Virtuoso Analog Design Environment L.
- The *Virtuoso Analog Design Environment XL SKILL Language Reference* provides descriptions, syntax, and examples for the SKILL commands supported by Virtuoso Analog Design Environment XL and Virtuoso Analog Design Environment XL.

Virtuoso Tools

- The *Virtuoso Analog Design Environment L User Guide* explains how to design and simulate analog circuits using Virtuoso Analog Design Environment L.
- The *Virtuoso Analog Design Environment XL User Guide* explains how to design and simulate analog circuits using Virtuoso Analog Design Environment XL.
- The *Virtuoso Analog Design Environment GXL User Guide* explains how to design and simulate analog circuits using Virtuoso Analog Design Environment GXL.
- The *Virtuoso Analog Distributed Processing Option User Guide* explains how to set up and run distributed processing for OCEAN and other Virtuoso Analog Design Environment applications.

Typographic and Syntax Conventions

This list describes the syntax conventions used for the Virtuoso® Analog Design Environment SKILL functions.

`literal` — Nonitalic words indicate keywords that you must type literally. These keywords represent command (function, routine) or option names.

`argument` (*`z_argument`*)
Words in italics indicate user-defined arguments for which you must substitute a name or a value. (The characters before the underscore (_) in the word indicate the data types that this argument can take. Names are case sensitive. Do not type the

underscore (*z_*) before your arguments.) For a listing of data types, see "Data Types Used in OCEAN" on page 32.

`|`	Vertical bars (OR-bars) separate possible choices for a single argument. They take precedence over any other character.
`[ ]`	Brackets denote optional arguments. When used with OR-bars, they enclose a list of choices. You can choose one argument from the list.
`{ }`	Braces are used with OR-bars and enclose a list of choices. You must choose one argument from the list.
...	Three dots (...) indicate that you can repeat the previous argument. If you use them with brackets, you can specify zero or more arguments. If they are used without brackets, you must specify at least one argument, but you can specify more. *argument*... Specify at least one, but more are possible. *[argument]*... Specify zero or more.
`,...`	A comma and three dots together indicate that if you specify more than one argument, you must separate those arguments by commas.
`=>`	A right arrow precedes the possible values that a SKILL function can return. This character is represented by an equal sign and a greater than sign.
`/`	A slash separates the possible values that can be returned by a SKILL function.
<*yourSimulator*>	Angle brackets indicate places where you need to insert the name of your simulator. Do not include the angle brackets when you insert the simulator name.

Important

The characters included in the list above are the only characters that are not typed literally. All other characters in the SKILL language are required and must be typed literally.

SKILL Syntax Examples

The following examples show typical syntax characters used in the SKILL language. For information on the SKILL language, see the *Cadence SKILL Language User Guide*.

Example 1

```
list( g_arg1 [g_arg2] …
    )
    => l_result
```

Example 1 illustrates the following syntax characters.

`list`	Plain type indicates words that you must type literally.
g_arg1	Words in italics indicate arguments for which you must substitute a name or a value.
`( )`	Parentheses separate names of functions from their arguments.
_	An underscore separates an argument type (left) from an argument name (right).
`[ ]`	Brackets indicate that the enclosed argument is optional.
`=>`	A right arrow points to the return values of the function. Also used in code examples in SKILL manuals.
…	Three dots indicate that the preceding item can appear any number of times.

Example 2

```
needNCells(
    s_cellType | st_userType
    x_cellCount
    )
    => t / nil
```

Example 2 illustrates two additional syntax characters.

|	Vertical bars separate a choice of required options.
`/`	Slashes separate possible return values.

Identifiers Used to Denote Data Types

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

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

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

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

Additional Learning Resources

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

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

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

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

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

1

Introduction to OCEAN

This chapter provides an introduction to Open Command Environment for Analysis (OCEAN). In this chapter, you can find information about

- Types of OCEAN Commands on page 28
- OCEAN Online Help on page 28
- OCEAN Syntax Overview on page 29
- Parametric Analysis on page 35
- Distributed Processing on page 36

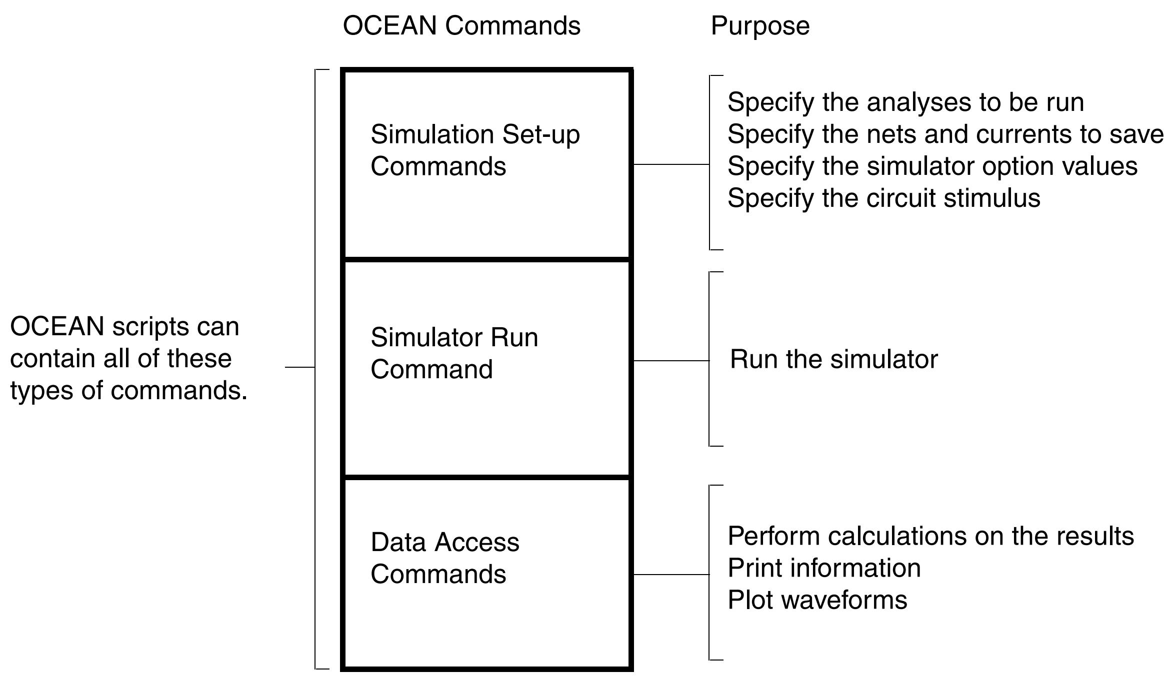
OCEAN lets you set up, simulate, and analyze circuit data. OCEAN is a text-based process that you can run from a UNIX shell or from the Command Interpreter Window (CIW). You can type OCEAN commands in an interactive session, or you can create scripts containing your commands, then load those scripts into OCEAN. OCEAN can be used with any simulator integrated into the Virtuoso® Analog Design Environment.

Typically, you use the Virtuoso® Analog Design Environment when creating your circuit (in Composer) and when interactively debugging the circuit. After the circuit has the performance you want, you can use OCEAN to run your scripts and test the circuit under a variety of conditions. After making changes to your circuit, you can easily rerun your scripts. OCEAN lets you

- Create scripts that you can run repeatedly to verify circuit performance
- Run longer analyses such as parametric analyses and statistical analyses more effectively
- Run long simulations in OCEAN without starting the Virtuoso® Analog Design Environment graphical user interface
- Run simulations from a nongraphic, remote terminal

Types of OCEAN Commands

You can create OCEAN scripts to accomplish the full suite of simulation and data access tasks that you can perform in the Virtuoso® Analog Design Environment. An OCEAN script can contain three types of commands, as shown in the following figure.



All the parameter storage format (PSF) information created by the simulator is accessible through the OCEAN data access commands. (The data access commands include all of the Virtuoso® Analog Design Environment calculator functions.)

You can use the history command to view the command history from the current session and the most recently terminated session.

OCEAN Online Help

Online help is available for all the OCEAN commands when you are in an OCEAN session. To get help for a specific OCEAN command, type the following:

```
ocnHelp( 'commandName )
```

This command returns an explanation of the command and examples of how the command can be used.

To get a listing of all the different types of commands in OCEAN, type the following:

```
ocnHelp()
```

For more information, see "ocnHelp" on page 169.

OCEAN Syntax Overview

OCEAN is based on the Virtuoso® SKILL programming language and uses SKILL syntax. All the SKILL language commands can be used in OCEAN. This includes `if` statements, `case` statements, `for` loops, `while` loops, `read` commands, `print` commands, and so on.

The most commonly used SKILL commands are documented in this manual. However, you are not limited to these commands. You can use any SKILL routine from any SKILL manual.

Common SKILL Syntax Characters Used in OCEAN

This section provides an overview of some basic SKILL syntax concepts that you need to understand to use OCEAN. For more information about SKILL syntax, see Chapter 3, "Introduction to SKILL."

Parentheses

Parentheses surround the arguments to the command. The command name is followed immediately by the left parenthesis, with no intervening space.

Examples

The following example shows parentheses correctly enclosing two arguments to the `path` command.

```
path( "~/simulation1/schematic/psf" "~/simulation2/schematic/psf" )
```

In the next example, the space after the command name causes a syntax error.

Syntax error.

```
path ( "~/simulation1/schematic/psf" "~/simulation2/schematic/psf" )
```

Quotation Marks

Quotation Marks are used to surround string values. A string value is a sequence of characters, such as `"abc"`.

In the following example, the directory names provided to the `path` command are strings, which must be surrounded by quotation marks.

```
path( "~/simulation1/schematic/psf" "~/simulation2/schematic/psf" )
```

Convention

In this manual, a SKILL convention is used to let you know when an argument must be a string. When you see the prefix *t_*, you must substitute a string value (surrounded by quotation marks) for the argument. Consider the following syntax statement:

desVar(*t_desVar1 g_value1 t_desVar2 g_value2*)

In this case, there are two string values that must be supplied: *t_desVar1* and *t_desVar2*. (The *g_* prefix indicates a different type of argument. For more information about prefixes, see Chapter 4, "Working with SKILL.")

Recovering from an Omitted Quotation Mark

Accidentally omitting a closing quotation mark from an OCEAN command can cause great confusion. For example, typing the incorrect command

```
strcat( "rain" "bow )
```

appears to hang OCEAN. In an attempt to recover, you type a `Control-c`. That gives you a prompt but it does not fix the problem, as you discover when you then type the correct command.

```
strcat( "rain" "bow" )
```

Again, you have to type a `Control-c` and OCEAN responds with another message.

```
^C*Error* parser: interrupted while reading input
```

If you find yourself in this situation, do not press a `Control-c`. Instead, recover by entering a quotation mark followed by a right square bracket (`]`). This procedure reestablishes a normal OCEAN environment and you can then reenter the correct command.

```
ocean> strcat( "rain" "bow )
"]
"rainbow ) "
ocean> strcat( "rain" "bow" )
"rainbow"
ocean>
```

Single Quotation Marks

The single quotation mark indicates that an item is a symbol. Symbols in SKILL correspond to constant enums in C. In the context of OCEAN, there are predefined symbols. The simulator that you use also has predefined symbols. When using symbols in OCEAN, you must use these predefined symbols.

Examples

In the following example, `tran` is a symbol and must be preceded by a single quotation mark. The symbol `tran` is predefined. You can determine what the valid symbols for a command are by checking the valid values for the command's arguments. For example, if you refer to "analysis" on page 89, you see that the valid values for the first argument include `'tran`.

```
analysis( 'tran … )
```

The list of items you can save with the `save` command is also predefined. You must choose from this predefined list. See "save" on page 143 and refer to the valid values for the *`s_saveType`* argument. The `'v` symbol indicates that the item to be saved is the voltage on a net.

```
save( 'v "net1" )
```

Convention

In this manual, a SKILL convention is used to let you know when an argument must be a symbol. When you see the prefix *`s_`*, you must substitute a symbol (preceded by a single quotation mark) for the argument. Consider the following syntax statement:

```
selectResults( s_resultsName ) => t / nil
```

In this case, there is one symbol that must be supplied: *`s_resultsName`*. For the `selectResults` command, there is a different mechanism that lets you know the list of predefined symbols. If you type the following command, with no arguments, the list of predefined symbols is returned: `results() => ( dc tran ac )`

Note: Depending on which results are selected, the values returned by the `results` command vary.

Question Mark

The question mark indicates an optional keyword argument, which is the first part of a keyword parameter. A keyword parameter has two components:

- The first component is the keyword, which has a question mark in front of it.

- The second component is the value being passed, which immediately follows the keyword.

Keyword parameters, composed of these keyword/value pairs, are always optional.

Examples

In the following example, all the arguments to the `analysis` command except `'tran` are keyword/value pairs and are optional.

Keyword Value passed

```
analysis( 'tran ?start 0 ?stop 1u ?step 1n )
```

For example, you can use `?center` and `?span` instead of `?start` and `?stop`. You also can omit `?start` altogether because it is an optional argument.

Convention

In this manual, a SKILL convention is used to let you know when arguments are optional. Optional arguments are surrounded by square brackets [ ]. In the following example, all of the keyword/value pairs are surrounded by square brackets, indicating that they are optional.

```
report([?output t_filename | p_port] [?type t_type] [?name t_name] [?param
     t_param] [?format s_reportStyle] ) => t / nil
```

Data Types Used in OCEAN

The following table shows the internal names and prefixes for the SKILL data types that are used in OCEAN commands.

Data Type	Internal Name	Prefix
floating-point number	flonum	f
any data type	general	g
linked list	list	l
integer, floating-point number, or complex number		n
user-defined type		o

Data Type	Internal Name	Prefix
I/O port	port	p
symbol	symbol	s
symbol or character string		S
character string (text)	string	t
window type		w
integer number	fixnum	x

For more information about SKILL datatypes, see Chapter 4, “Working with SKILL.”

OCEAN Return Values

You get return values from most OCEAN commands and can use these values in other OCEAN commands.

The following table shows some examples in which the return value from a command is assigned to a variable.

Assigning a Return Value to a Variable	Resulting Value for the Variable
`a=desVar("r1" 1k)`	a=1k
`a=desVar("r1" 1k "r2" 2k)`	a=2k
`a=desVar("r1")`	a=1k, assuming `r1` was set in a `desVar` command
`a=desVar("r2")`	a=2k, assuming `r2` was set in a `desVar` command

Design Variables in OCEAN

Design variables in OCEAN function as they do in the Virtuoso® Analog Design Environment. Design variables are not assigned in the order specified. Rather, they are reordered and then assigned. Consider the following example:

```
desVar( "a" "b+1" )
desVar( "b" 1 )
```

You might expect an error because `a` is assigned the value `b+1` before `b` is assigned a value. However, OCEAN reorders the statements and sends them as follows:

```
desVar( "b" 1 )
desVar( "a" "b+1" )
```

After the reordering, there is no error. (`b` is equal to `1` and `a` is equal to `2`.)

Suppose you run a simulation, then specify the following:

```
desVar("b" 2)
```

You might expect `a` to be equal to `2`, which was the last value specified. Instead, `a` is reevaluated to `b+1` or `3`.

This approach is similar to how the design variables are used in simulation. For example, consider a circuit that has the following resistor:

```
R1 1 0 resistor r=b
```

If you change the value of `b`, you expect the value of `R1` to change. You do not expect to have to netlist again or retype the `R1` instantiation.

This approach is used in the Virtuoso® Analog Design Environment. Users are not expected to enter design variables in a particular order. Rather, the design variables are gathered during the design variable search then reordered before they are used.

Note: Do not use simulator reserved words as design variable names. For more information, see the Reserved Words section in the *Virtuoso Analog Design Environment User Guide*.

outputs() in OCEAN

Throughout this manual are examples of nets and instances preceded by a “`/`” as well as examples without the “`/`”. There is a significant difference between the two.

If you create a design in the Virtuoso® Analog Design Environment and save the OCEAN file, all net and instance names will be preceded with a “`/`”, indicating they are schematic names. The `netlist/amap` directory must be available to map these schematic names to names the simulator understands. (If your design command points to the raw netlist in the `netlist` directory, the `amap` directory is there.)

If you create a design or an OCEAN script by hand, or move the raw netlist from the `netlist` directory, the net and instance names might not be preceded with “`/`”. This indicates that simulator names are used, and mapping is not necessary.

If you are unsure whether schematic names or simulator names are used, after `selectResult( S_resultsName )`, type `outputs()` to see the list of net and instance names.

Note: Although you can move the raw netlist file from the `netlist` directory, it is not advised. There are other files in the `netlist` directory that are now required to run OCEAN.

Parametric Analysis

There are two ways you can run parametric analyses in OCEAN:

- You can use the `paramAnalysis` command (recommended approach).
- You can use a SKILL `for` loop.

Using the `paramAnalysis` command is an easier approach. With this command, you can set up any number of nested parametric analyses in an OCEAN script. The `paramRun` command runs all the parametric analyses. When the analysis is complete, the data can be plotted as a family of curves. The following example shows how you might use nested parametric analyses:

```
paramAnalysis( "rl" ?start 200 ?stop 600 ?step 200
    paramAnalysis( 'rs ?start 300 ?stop 700 ?step 200
    )
)
paramRun ()
```

In this example, the outer loop cycles through `r1`, and the inner loop cycles through `rs` as follows:

Loop through `r1` from 200 to 600 by 200.

Loop through `rs` from 300 to 700 by 200.

Run.

End the first loop.

End the second loop.

So, for `r1=200`, `rs` equals 300, 500 and then 700. Then, for `r1=400`, `rs` equals 300, 500, and then 700. Finally, for `r1=600`, `rs` equals 300, 500, and then 700

Use a SKILL `for` loop only if the `paramAnalysis` command is not adequate. You can use the SKILL `for` loop to set up any number of variable-switching runs. After all the simulations are complete, you have to work with the results directories individually. The following example shows how you might use SKILL loops for parametric analyses.

```
Cload = list( 2p 4p 6p 8p )
foreach( val  Cload
    desVar( "Cload" val )
    a=resultsDir( sprintf( nil "./demo/Cload=%g" val ) )
    printf( "%L", a )
    run( )
)
foreach( val  Cload
    openResults( sprintf( nil "./<dir>/Cload=%g" val ) )
    selectResults( 'ac )
    plot( vdb( "vout" ) )
)
```

Data Access Without Running a Simulation

You can retrieve and use data from previous simulations at any time by opening the data with the `openResults` command. After opening the data, you can use any data access commands on this data. For more information, see Chapter 7, "Data Access Commands."

You can use query commands such as `results`, `outputs`, and `dataTypes` to see what data is available to be opened.

Distributed Processing

You can use OCEAN distributed processing commands to run simulations across a collection of computer systems. The distributed processing commands allow you to specify where and when jobs are run and allow you to monitor and control jobs in a variety of ways. Using distributed commands, you can

- Submit one or more jobs to a distributed processing queue
- Specify a host or group of hosts on which to distribute jobs
- View the status of jobs
- Specify when a job will run or in what sequence a group of jobs will run
- Suspend and resume jobs
- Cancel jobs

For you to be able to use the distributed processing commands, your site administrator needs to set up the lists of machines to which jobs are submitted. Each list of machines is a group of hosts and is called a queue. Consult the *Virtuoso Analog Distributed Processing*

Option User Guide for more information on how to configure systems for distributed processing. For information on the distributed processing commands for OCEAN, see Chapter 12, “OCEAN Distributed Processing Commands.”

Blocking and Nonblocking Modes

You can configure jobs to run in blocking or nonblocking mode. In blocking mode, execution of subsequent OCEAN commands is halted until the job completes. In nonblocking mode, the system does not wait for the first job to finish before starting subsequent jobs.

Blocking Mode

You must run jobs in blocking mode to be able to use the data resulting from a job in a subsequent command in an OCEAN script or batch run.

For example, if you want to run a simulation, select the `tran` results from that simulation, and then plot them, you

1. Configure the simulation with setup commands
2. Run the simulation with the `run()` command
3. Select the desired results with the `selectResults( 'tran)` command
4. Plot the results with the `plot()` command

A job like the one in the example above must run in blocking mode so that the commands are processed sequentially. If the jobs in the example above are run in nonblocking mode, the `selectResult` command starts before the `run` command can return any data, and the `selectResult` command and the `plot` command cannot complete successfully.

Nonblocking Mode

If you are submitting several jobs that have no interdependencies, you can run them concurrently when `hostmode` is set to `distributed`.

For example, if you want to run two separate simulations as jobs, but do not want to wait until the first is complete before starting the second, you

1. Configure the first simulation with setup commands
2. Configure a second simulation with setup commands

In the example above, the script starts the first job and then starts the second job without waiting for the first job to finish.

If you are running several commands, and some of them are data access commands, you can use the wait command to block a single job. The `wait` command is needed between the simulation and the data access commands to ensure the desired simulation is complete before the data is accessed.

For example, if you want to run two separate simulations as jobs (`sim1` and `sim2`), and want to select and plot the results of the second simulation run, you

1. Configure the first simulation with setup commands
2. Run the simulation with a `run(?jobPrefix "sim1")` command
3. Configure a second simulation with setup commands
4. Run the second simulation with the `run( ?jobPrefix "sim2)` command
5. Cause the script to wait until the second simulation finishes before starting the `selectResults` command with the `wait(sim2)` command
6. Select the desired results with the `selectResults('tran)` command
7. Plot the results with the `plot( )` command

In the example above, the script starts the first job and then starts the second job without waiting for the first job to finish. When the script reaches the `wait` command, it pauses until the second simulation runs and then selects the results to plot.

Plotting Simulation Results

The simulation results can be plotted in Virtuoso Visualization and Analysis XL, which is supported in the OCEAN environment.

2

Using OCEAN

This chapter explains the different ways you can use OCEAN to perform simulation tasks. In this chapter, you can find information about

OCEAN Use Models

There are two ways you can use OCEAN:

- You can use OCEAN interactively to perform simple tasks.
- You can use OCEAN in batch mode and provide the name of an existing (or parameterized) script as a command line argument. OCEAN scripts can be created
  - From the Virtuoso® Analog Design Environment window with the command *Session – Save Script*
  - By hand (by you or someone else in your organization) with a text editor

  For information about creating scripts, see “Creating OCEAN Scripts” on page 45.

All the OCEAN commands are described in this manual, and online help is available for all these commands. For information about using the OCEAN online help, see “OCEAN Online Help” on page 28.

Note: The current version of OCEAN has some specific issues that are addressed in Appendix 16, “OCEAN 4.4.6 Issues.” Please refer to this appendix before using OCEAN.

Using OCEAN Interactively

You can run OCEAN from a UNIX prompt or from the Virtuoso® design framework II (DFII) Command Interpreter Window (CIW).

Note: The primary use model is to use OCEAN in a UNIX shell. Unless otherwise indicated, the rest of this chapter assumes that you are working from OCEAN in a UNIX shell.

Using OCEAN from a UNIX Shell

To start OCEAN from a UNIX prompt, type the following command:

```
ocean
```

This command loads and reads the `.oceanrc` file. You can place OCEAN commands in your `.oceanrc` file, which is similar to the `.cdsinit` file. This file can contain any valid OCEAN command, function or SKILL initialization routine (excluding graphical dfII references, such as bindkeys and so on, which are not applicable to OCEAN). If you do not want to specify any startup initialization options for OCEAN, you do not need to create or add an `.oceanrc` file.

The OCEAN prompt appears indicating that you have started OCEAN:

```
ocean>
```

If you do not see this prompt after starting OCEAN, press `Return`. If you still do not see this prompt, you may have redefined the prompt with the `setPrompt` command. (This does not affect OCEAN; the prompt just will not indicate OCEAN is running.)

Now you can start typing OCEAN commands interactively. For an example of interactive use, see "Interactive Session Demonstrating the OCEAN Use Model" on page 43.

To quit the OCEAN executable from UNIX, type the following command:

```
exit
```

OCEAN in Non-Graphical Mode

OCEAN is an executable shell script that calls the AWD workbench and passes all its command-line options to it using the following shell command:

```
#! /bin/sh -

exec awd -ocean "$@"
```

This makes OCEAN highly dependent on the UNIX shell environment.

You can run OCEAN in a non-graphical mode by using the `-nograph` option with the `ocean` command. This disables the graphical options of the software. This option is useful if OCEAN is started on a machine that does not have X-Windows running.

Note: You can use the `-nograph` option to run the OCEAN job through a cron. Ensure that `DISPLAY` is set to "`:0`". If the screen will be locked when the OCEAN cron job runs, use the `allowaccess` option with the `xlock` command on the UNIX prompt. For more information on the usage of `xlock`, type `man xlock` in a terminal window.

The `-nograph` option must only be used to replay logfiles that have been created interactively. For example, while using OCEAN with the `-nograph` option, your `oceanScript.ocn` file must have an `exit()` statement at the end followed by a newline. Otherwise, OCEAN hangs. The reason for this is that when the workbench is started in the non-graphical mode, it does not redirect standard I/O as it normally does; instead, it lets the SKILL human interface (HI) handle the standard I/O. HI expects an explicit `exit()` statement at the end of the OCEAN script and OCEAN exits only when it detects an `exit()` at EOF. The command is used as follows:

```
ocean -nograph < oceanScript.ocn > oceanScript.log
```

Alternatively, you can execute the OCEAN script using the `-replay` option. The command is used as follows:

```
ocean -nograph -replay oceanScript.ocn -log oceanScript.log
```

`.oceanrc` is automatically loaded while using ocean `-nograph -replay` command. If you use the `virtuoso` command, `.oceanrc` is not loaded automatically.

While using the `-nograph` option with `ocean`, if you find that simulation run messages are not being stored in the log file, check for the following environment variable in the `.cdsenv` file:

```
(envGetVal "spectre.envOpts" "firstRun" )
```

It must be set to `nil` as shown below for simulation run messages to be stored in it:

```
(envSetVal "spectre.envOpts" "firstRun" 'boolean nil)
```

For more information about this variable, see Appendix B of the *Virtuoso Analog Design Environment L User Guide*.

Using OCEAN from the CIW

You can type OCEAN commands in the CIW after you bring up the Virtuoso® Analog Design Environment. (Starting the design environment loads the required OCEAN files.)

Your `.oceanrc` file is *not* automatically read when you start the DFII software (using the `virtuoso` command). Therefore, you might want to load your `.oceanrc` file manually in the CIW if you need information that it contains.

You can also use the history command from the CIW to list and reuse the most recently used commands.

Interactive Session Demonstrating the OCEAN Use Model

The following figure shows a typical set of simulation tasks that you might perform interactively in OCEAN with the corresponding commands.

Start OCEAN and specify your simulator.	`ocean` `simulator( 'spectre )`

Specify an AC analysis.	`analysis( 'ac ?start 1 ?stop 1000 ?lin 100 )`
	`or`
	`ac( 1 1000 "linear" 100 )`

Set a design variable.	`desVar( "rs" 1k )`
Perform the first simulation run.	`run( )`
Change a design variable.	`desVar( "rs" 2k )`
Perform the second simulation run	`run( )`

Specify a transient analysis.	`analysis( 'tran ?stop 1u ) (Spectre only)`
	`or`
	`tran( 0 1u 1n )`

Perform the third simulation run.	`run( )`

On the second and third run, the AC analysis runs because it is still active. If you do not want it to run, you must disable it with the following command:

```
delete( 'analysis 'ac )
```

The simulator is not called and run until the `run()` command is entered.

The commands can be given in any order, as long as they are all defined before the `run()` command.

License Requirements

You need licenses to run the `simulator()` and `ocnSetXLMode()` OCEAN commands. For more information on these commands, see simulator on page 147 and ocnSetXLMode on page 592.

- To run the `simulator()` OCEAN command, you must have one of the following licenses. If one of these licenses are not already checked out, the first available license will be checked out in the following order:
    - 95200 Virtuoso(R) Analog Design Environment L
    - 95210 Virtuoso(R) Analog Design Environment XL
    - 95220 Virtuoso(R) Analog Design Environment GXL
- To run the `ocnSetXLMode()` OCEAN command, you must have one of the following licenses. If one of these licenses are not already checked out, the first available license will be checked out in the following order:
    - 95210 Virtuoso(R) Analog Design Environment XL
    - 95220 Virtuoso(R) Analog Design Environment GXL

  Note: If you have run the `ocnSetXLMode()` command, running the `simulator()` command subsequently will not checkout an additional license.

If you do not want OCEAN to automatically checkout a higher tiered license—for example, if you do not want OCEAN to automatically checkout the `95210` license if the `95200` license is not available—set the following environment variable in your `.cdsenv` file:

```
asimenv.misc  alwaysTryHigherTieredLicenseInOcean   'boolean nil
```

Note: If the `alwaysTryHigherTieredLicenseInOcean` environment variable is set to `nil`, errors are displayed if OCEAN is unable to checkout a license.

Note: The `95200 Virtuoso(R) Analog Design Environment L` license is checked out when you load the ocean script. Exit Virtuoso, or run the `ocnCloseSession()` command to release the license.

Creating OCEAN Scripts

You can modify the included sample script files or create script files interactively from the Virtuoso® Analog Design Environment.

Creating Scripts Using Sample Script Files

You can create your own script files with a text editor using the sample scripts as examples, or you can make copies of the sample scripts and modify them as needed using a text editor. The scripts can be found in the following directory:

your_install_dir/tools/dfII/samples/artist/OCEAN

Refer to the `README` file in this directory for information about the scripts.

Creating Scripts from the Analog Design Environment

When you perform tasks in the design environment, the associated OCEAN commands are automatically stored in the *simulatorx*.ocn file in your `netlist` directory. For example, if you start the Virtuoso software, open the Virtuoso® Analog Design Environment window, and run a simulation using the Cadence SPICE simulator, a `cdsSpice0.ocn` file is created in your `netlist` directory. You can load this `cdsSpice0.ocn` script as described in "Loading OCEAN Scripts" on page 48.

Selectively Creating Scripts

You can be selective about the information that is created in your `.ocn` script. The Virtuoso® Analog Design Environment has a feature that lets you create an OCEAN script based on the state of your current session. The following example illustrates how using this feature is different than using the automatic script generation feature.

Consider the following task flow:

1. Start the Virtuoso® Analog Design Environment.
2. Specify a DC analysis.
3. Select nets on the schematic to save.
4. Run the simulation.
5. Turn off the DC analysis.
6. Select a transient analysis.

7. Run the simulation.
8. Save the script from the Virtuoso® Analog Design Environment.

The script that is created, called `oceanScript.ocn` by default, contains only the selected nets, the transient analysis, and the run command. The script does not contain the DC analysis because it was turned off.

In contrast, the *`simulator`*`0.ocn` script, which is automatically created in the `netlist` directory, contains all of the commands, including the DC analysis and the current state of the analysis (on or off).

Creating a Script

To selectively create a script from Virtuoso Analog Design Environment L or ,

1. Start the Virtuoso software,

   `virtuoso`

   The CIW appears.
2. From the CIW, choose *Tools – ADE L – Simulation*.

   The Virtuoso Analog Design Environment window appears.
3. Perform all of the design environment tasks that you want to capture in the script.
4. Choose *Session – Save Script*.

   The Save Ocean Script to File form appears.
5. Click *OK* to accept the default file name (`./oceanScript.ocn`), or change the name for the file and click *OK*.

   A script containing the OCEAN commands for the tasks you performed is created. For information about how to load this script, see "Loading OCEAN Scripts" on page 48.

Controlling What Is Included in Scripts

You can use `.cdsenv` variables to alter the OCEAN script that is created when you choose *Session – Create Script* in the Virtuoso Analog Design Environment. One variable allows you to include default environment settings in a script, two other variables allow you to run procedures before and after a script is created.

Including Default Control Statements

To save every control statement, including default statements, in your OCEAN script, add the following line to your `.cdsenv` file.

```
asimenv.misc saveDefaultsToOCEAN boolean t
```

Setting `saveDefaultsToOCEAN` to `t` results in a complete dump of the current circuit design environment, defaults and all. Because the created OCEAN script contains all the settings, you might use this variable when you plan to archive a script, for example.

If `saveDefaultsToOCEAN` is not set to `t`, the created OCEAN script contains only those items that you explicitly set to some value other than their default.

Running Functions Before or After Creating a Script

The information in this section describes how you can specify functions to be run before or after a script is created. You can use these functions, for example, to add information at the beginning or end of a script. To use this capability follow these steps.

1. Decide when you want the functions to run.

   - ❑ Add the following line to your `.cdsenv` file to run the function *preOceanFunc* before the OCEAN script is created.

     ```
     asimenv.misc preSaveOceanScript string "preOceanFunc"
     ```

   - ❑ Add the following line to your `.cdsenv` file to run the function *postOceanFunc* after the OCEAN script is created.

     ```
     asimenv.misc postSaveOceanScript string "postOceanFunc"
     ```

2. Use the following syntax to specify the functions.

   ```
   preOceanFunc( session fp )
   postOceanFunc( session fp )
   ```

   In this syntax, *session* is the OASIS session and `fp` is the file pointer to the OCEAN script file. For guidance on determining the *session* to use, see the *Virtuoso Analog Design Environment L SKILL Language Reference*.

3. Load the functions in your `.cdsinit` file.

   For example, you might add the following lines to your `.cdsenv` file.

   ```
   asimenv.misc  preSaveOceanScript string "MYfirstProc"
   asimenv.misc  postSaveOceanScript string "MYlastProc"
   ```

   The functions `MYfirstProc` and `MYlastProc` might be defined like this.

   ```
   procedure( MYfirstProc( session fp)
     fprintf(fp "; This will be the first line in the ocean script.\n")
   )
   ```

```
procedure( MYlastProc( session fp)
  fprintf(fp "; This will be the last line in the ocean script.\n")
)
```

If these procedures are defined in a file called `myOceanProcs.il`, you can load them by adding to your `.cdsinit` file a command like the following.

```
load "myOceanProcs.il"
```

When you choose *Session – Create Script*, first the `preSaveOceanScript` procedure is called, then the OCEAN script is created, then the `postSaveOceanScript` procedure is called.

Loading OCEAN Scripts

You can load OCEAN scripts from OCEAN (in UNIX) or from the CIW.

From a UNIX Shell

To load an OCEAN script,

1. Type the following command to start OCEAN:

   `ocean`

   The OCEAN prompt appears.

2. Use the SKILL `load` command to load your script:

   `load( "`*`script_name`*`.ocn" )`

   Messages about the progress of your script appear.

From the CIW

To load an OCEAN script,

1. Start the Virtuoso software

   `virtuoso &`

   The CIW appears.

2. In the text entry field, use the SKILL `load` command to load your script:

   `load( "`*`script_name`*`.ocn" )`

   Messages about the progress of your script appear in the CIW.

Selecting Results

You may use OCEAN to run several simulations on the same design and save the results in different result directories. You can then use Analog Design Environment XL to select the results and work with features like annotation etc.

Selecting Results Run from Worst Case Scripts for Cross-Probing or Back Annotating Operating Points

Assume that you have been using Ocean to create separate data directories for worst case parameter sweeps. Also, assume that the new directories you make are accessed with the `resultsDir()` ocean command in your Ocean script and that these directories are in the standard location where `psf` data is stored in the Analog Design Environment.

In the Analog Design Environment, `psf` data is stored in:

```
<runDir>/simulation/<testSchemName>/spectre/schematic/psf
```

where,

`runDir`	is the directory where you invoke `virtuoso&`
`testSchemName`	is your test schematic

This implies that your script should store the new directories under the `schematic` directory. Therefore, if c1, c2 and c3 are the worst case directories, they are located at:

```
<runDir>/simulation/<testSchemName>/spectre/schematic/c1
<runDir>/simulation/<testSchemName>/spectre/schematic/c2
<runDir>/simulation/<testSchemName>/spectre/schematic/c3
```

1. Choose *Results – Select*
2. The *Select Results* form opens. Click *Browse*. A Unix Browser form appears.
3. Navigate to the directory that contains your Ocean generated directories c1, c2, and c3.
4. Click *OK* on the Unix Browser form. Now the *Select Results* Form should show c1, c2 and c3.
5. Double click on c1, c2 or c3. Alternatively, you can also single click on c1, c2 or c3 and then choose *Update Results* and click *OK*. At this point the data is selected though there is no confirmation in the CIW. Now, you should be able to use *Results – Direct Plot* , *Results – Annotate* etc to see the results of that particular directory.

Selecting Results Run from Spectre Standalone

After running spectre standalone, you can select results using the *Results Browser* and use calculator to plot the results. However, this does not allow you to use ADE features like *Results – Direct Plot* or *Results – Annotate*.

Consider that your data is in

`<runDir>/simulation/<testSchemName>/spectre/schematic/psf.`

where,

`runDir`	is the directory where you invoke `virtuoso&`
`testSchemName`	is your test schematic

1. Choose *Tools – Results Browser* . A pop up box appears. Enter your design path up to the spectre directory.
2. Click *OK*, and the browser comes up.
3. Click on schematic directory. The psf directory should appear.
4. Click on the directory with the data in it, psf. When you click on the 'psf' directory you should see the tree expand with different results from your spectre stand alone simulation, e.g. tran.tran etc.
5. Place the mouse pointer over the 'psf' node in the tree and press down the middle mouse key and scroll down to "create ROF". You should now see the psf directory change, and an intermediate node comes up --Run1-- betweenpsf/ and the results.
6. Place the middle mouse pointer over the Run1 node, scroll down and select "Select Results".

   Note: Even though there is a confirmation message in the CIW that the select was success, Analog Design Environment is not synced up to allow cross-probing and back annotation of operating points.
7. You may now use *Tools – Calculator* to select objects from the schematic. You can then choose 'plot' from the calculator, or different calculator operations.

   Note: You CAN use *Tools – Calculator* but you CAN NOT use *Results – Direct Plot* or *Results – Annotate* etc.

Running Multiple Simulators

There are times when you might want to run more than one simulator. You might be benchmarking simulators or comparing results. In OCEAN, you can only use one simulator per OCEAN session. If you change simulators, you must start a new OCEAN session. This is because some OCEAN command arguments are simulator specific, and therefore change when the simulator changes. For example, the arguments to the `option` command are simulator specific. (No two simulators have the exact same options.) The analyses are typically simulator specific also.

OCEAN Tips

The information in this section can help you solve problems that you encounter while using OCEAN.

- While working in OCEAN, you might get the following SKILL error message:

  ```
  *Error* eval: unbound variable - nameOfVariable
  ```

  In this case, you need to see if you have an undeclared variable or if you are missing a single quotation mark (') or a quotation mark (") for one of your arguments. For example, the following command returns an error message stating that `fromVal` is an unbound variable because the variable has not been declared:

  ```
  analysis('tran ?from fromVal)
  ```

  However, the following pair of statements work correctly because `fromVal` has a value (is bound).

  ```
  fromVal=0
  analysis('tran ?from fromVal)
  ```

- If you get an error in an OCEAN session, you are automatically put into the SKILL debugger. In this case, you see a prompt similar to this:

  ```
  ocean-Debug 2>
  ```

  You can continue working. However, if you would like to get out of the debugger, you can type

  ```
  debugQuit()
  ```

  Now you are back to the normal prompt:

  ```
  ocean>
  ```

- If it appears that OCEAN does not accept your input, or OCEAN appears to hang, then you may have forgotten to enter a closing quotation mark. Type `"]` to close all strings. For more information, and some examples, see “Recovering from an Omitted Quotation Mark” on page 30.

- In SKILL, the following formats are equivalent: `(one two)` and `one(two)`. Results might be returned in either format. For example, OCEAN might return `ac(tran)` or `(ac tran)`, but the two forms are equivalent.
- You can check your script for simple syntax errors by running SKILL lint. For example, you might use a command like

  ```
  sklint -file myScript.ocn
  ```

  From within OCEAN, you can run SKILL lint by typing the following at the OCEAN prompt:

  ```
  sklint(?file "yourOceanScript.ocn")
  ```

  Running SKILL lint helps catch basic errors, such as unmatched parentheses and strings that are not closed.

3

Introduction to SKILL

This chapter introduces you to the basic concepts that can help you get started with the Virtuoso® SKILL programming language. In this chapter, you can find information about

The Advantages of SKILL

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

SKILL is ideal for rapid prototyping. You can incrementally validate the steps of your algorithm before incorporating them in a larger program.

SKILL leverages your investment in Cadence technology because you can combine existing functionality and add new capabilities.

SKILL lets you access and control all the components of your tool environment: the User Interface Management System, the Design Database, and the commands of any integrated design tool. You can even loosely couple proprietary design tools as separate processes with SKILL's interprocess communication facilities.

Naming Conventions

The recommended naming scheme is to use uppercase and lowercase characters to separate your code from code developed by Cadence.

All code developed by Cadence Design Systems typically names global variables and functions with up to three lowercase characters, that signify the code package, and the name starting with an uppercase character. For example, `dmiPurgeVersions()` or *`hnlCellOutputs`*. All code developed outside Cadence should name global variables by starting them with an uppercase character, such as *AcmeGlobalForm*.

Arithmetic Operators

SKILL provides many arithmetic operators. Each operator corresponds to a SKILL function, as shown in the following table.

Sample SKILL Operators

Operators in Descending Precedence	Underlying Function
**	exponentiation
* /	multiply divide
+ –	plus minus
== !=	equal nequal
=	assignment

Scaling Factors

SKILL provides a set of scaling factors that you can add to the end of a decimal number (integer or floating point) to achieve the scaling you want.

- Scaling factors must appear immediately after the numbers they affect. Spaces are not allowed between a number and its scaling factor.
- Only the first nonnumeric character that appears after a number is significant. Other characters following the scaling factor are ignored. For example, for the value 2.3mvolt, the *m* is significant, and the *volt* is discarded. In this case, *volt* is only for your reference.

- If the number being scaled is an integer, SKILL tries to keep it an integer; the scaling factor must be representable as an integer (that is, the scaling factor is an integral multiplier and the result does not exceed the maximum value that can be represented as an integer). Otherwise, a floating-point number is returned.

The scaling factors are listed in the following table.

Scaling Factors

Character	Name	Multiplier	Examples
Y	Yotta	10^{24}	10Y [10e+25]
Z	Zetta	10^{21}	10Z [10e+22]
E	Exa	10^{18}	10E [10e+19]
P	Peta	10^{15}	10P [10e+16]
T	Tera	10^{12}	10T [1.0e13]
G	Giga	10^{9}	10G [10,000,000,000]
M	Mega	10^{6}	10M [10,000,000]
K	Kilo	10^{3}	10K [10,000]
%	percent	10^{-2}	5% [0.05]
m	milli	10^{-3}	5m [5.0e-3]
u	micro	10^{-6}	1.2u [1.2e-6]
n	nano	10^{-9}	1.2n [1.2e-9]
p	pico	10^{-12}	1.2p [1.2e-12]
f	femto	10^{-15}	1.2f [1.2e-15]
a	atto	10^{-18}	1.2a [1.2e-18]
z	zepto	10^{-21}	1.2z [1.2e-21]
y	yocto	10^{-24}	1.2y [1.2e-24]

Note: The characters used for scaling factors depend on your target simulator. For example, if you are using cdsSpice, the scaling factor for *M* is different than shown in the previous table, because cdsSpice is not case sensitive. In cdsSpice, *M* and *m* are both interpreted as 10^{-3}, so *ME* or *me* is used to signify 10^{6}.

Relational and Logical Operators

This section introduces you to

- Relational Operators: <, <=, >, >=, ==, !=
- Logical Operators: !, &&, ||

Relational Operators

Use the following operators to compare data values. SKILL generates an error if the data types are inappropriate. These operators all return `t` or `nil`.

Sample Relational Operators

Operator	Arguments	Function	Example	Return Value
<	numeric	lessp	`3 < 5` `3 < 2`	`t` `nil`
<=	numeric	leqp	`3 <= 4`	`t`
>	numeric	greaterp	`5 > 3`	`t`
>=	numeric	geqp	`4 >=3`	`t`
==	numeric string list	equal	`3.0 == 3` `"abc" == "ABc"`	`t` `nil`
!=	numeric string list	nequal	`"abc" != "ABc"`	`t`

Knowing the function name is helpful because error messages mention the function (`greaterp` below) instead of the operator (>).

```
1 > "abc"
Message: *Error* greaterp: can't handle (1 > "abc")
```

Logical Operators

SKILL considers `nil` as FALSE and any other value as TRUE. The and (`&&`) and or (`||`) operators only evaluate their second argument if it is required for determining the return result.

Sample Logical Operators

Operator	Arguments	Function	Example	Return Value

`&&`	general	and	`3 && 5`
			`5 && 3`
			`t && nil`
			`nil && t`
				`5`
				`3`
				`nil`
				`nil`

`||`	general	or	`3 || 5`
			`5 || 3`
			`t || nil`
			`nil || t`
				`3`
				`5`
				`t`
				`t`

The `&&` and `||` operators return the value last computed. Consequently, both `&&` and `||` operators can be used to avoid cumbersome `if` or `when` expressions.

The following example illustrates the difference between using `&&` and `||` operators and using `if` or `when` expressions.

You do not need to use

```
If (usingcolor then
currentcolor=getcolor( )
else
currentcolor=nil
)
```

Instead use

```
currentcolor=usingcolor && getcolor( )
```

Using &&

When SKILL creates a variable, it gives the variable a value of `unbound` to indicate that the variable has not been initialized yet. Use the `boundp` function to determine whether a variable is bound. The `boundp` function

- Returns `t` if the variable is bound to a value
- Returns `nil` if the variable is not bound to a value

Suppose you want to return the value of a variable `trMessages`. If `trMessages` is `unbound`, retrieving the value causes an error. Instead, use the expression

```
boundp( 'trMessages ) && trMessages
```

Using ||

Suppose you have a default name, such as `noName`, and a variable, such as `userName`. To use the default name if `userName` is `nil`, use the following expression:

```
userName || "noName"
```

SKILL Syntax

This section describes SKILL syntax, which includes the use of special characters, comments, spaces, parentheses, and other notation.

Special Characters

Certain characters are special in SKILL. These include the *infix* operators such as less than (<), colon (`:`), and assignment (=). The following table lists these special characters and their meaning in SKILL.

Note: All nonalphanumeric characters (other than _ and `?`) must be preceded (*escaped*) by a backslash (\) when you use them in the name of a symbol.

Special Characters in SKILL

Character	Name	Meaning
\	backslash	Escape for special characters
( )	parentheses	Grouping of list elements, function calls
[ ]	brackets	Array index, super right bracket
'	single quotation mark	Specifies a symbol (quoting the expression prevents its evaluation)
"	quotation mark	String delimiter
,	comma	Optional delimiter between list elements
;	semicolon	Line-style comment character

Special Characters in SKILL

Character	Name	Meaning
`+, −, *, /`	arithmetic	For arithmetic operators; the `/*` and `*/` combinations are also used as comment delimiters
`!,^,&,|`	logical	For logical operators
<,>,=	relational	For relational and assignment operators; < and > are also used in the specification of bit fields
`?`	question mark	If first character, implies keyword parameter
`%`	percent sign	Used as a scaling character for numbers

White Space

White space sometimes takes on semantic significance and a few syntactic restrictions must therefore be observed.

Write function calls so the name of a function is immediately followed by a left parenthesis; no white space is allowed between the function name and the parenthesis. For example

`f(a b c)` and `g()` are legal function calls, but `f (a b c)` and `g ()` are illegal.

The unary minus operator must immediately precede the expression it applies to. No white space is allowed between the operator and its operand. For example

`-1`, `-a`, and `-(a*b)` are legal constructs, but `- 1`, `- a`, and `- (a*b)` are illegal.

The binary minus (subtract) operator should either be surrounded by white space on both sides or be adjacent to non-white space on both sides. To avoid ambiguity, one or the other method should be used consistently. For example:

`a - b` and `a-b` are legal constructs for binary minus, but `a -b` is illegal.

Comments

SKILL permits two different styles of comments. One style is block oriented, where comments are delimited by `/*` and `*/`. For example:

```
/* This is a block of (C style) comments
comment line 2
comment line 3 etc.
*/
```

The other style is line- oriented where the semicolon (`;`) indicates that the rest of the input line is a comment. For example:

```
x = 1           ; comment following a statement
; comment line 1
; comment line 2 and so forth
```

For simplicity, line-oriented comments are recommended. Block-oriented comments cannot be nested because the first `*/` encountered terminates the whole comment.

Role of Parentheses

Parentheses `( )` delimit the names of functions from their argument lists and delimit nested expressions. In general, the innermost expression of a nested expression is evaluated first, returning a value used in turn to evaluate the expression enclosing it, and so on until the expression at the top level is evaluated. There is a subtle point about SKILL syntax that C programmers, in particular, must be very careful to note.

Parentheses in C

In C, the relational expression given to a conditional statement such as `if`, `while`, and `switch` must be enclosed by an outer set of parentheses for purely syntactical reasons, even if that expression consists of only a single Boolean variable. In C, an `if` statement might look like

```
if (done) i=0; else i=1;
```

Parentheses in SKILL

In SKILL, parentheses are used for specifying lists, calling functions, delimiting multiple expressions, and controlling the order of evaluation. You can write function calls in prefix notation

`(fn2 arg1 arg2)` **or** `(fn0)`

as well as in the more conventional algebraic form

`fn2(arg1 arg2)` **or** `fn0()`

The use of syntactically redundant parentheses causes variables, constants, or expressions to be interpreted as the names of functions that need to be further evaluated. Therefore,

- Never enclose a constant or a variable in parentheses by itself; for example, `(1)`, `(x)`.

- For arithmetic expressions involving *infix* operators, you can use as many parentheses as necessary to force a particular order of evaluation, but never put a pair of parentheses immediately outside another pair of parentheses; for example, `((a + b))`: the expression delimited by the inner pair of parentheses would be interpreted as the name of a function.

For example, because `if` evaluates its first argument as a logical expression, a variable containing the logical condition to be tested should be written without any surrounding parentheses; the variable by itself is the logical expression. This is written in SKILL as

```
if( done then i = 0 else i = 1)
```

Line Continuation

SKILL places no restrictions on how many characters can be placed on an input line, even though SKILL does impose an 8,191 character limit on the strings being entered. The parser reads as many lines as needed from the input until it has read in a complete form (that is, expression). If there are parentheses that have not yet been closed or binary *infix* operators whose right sides have not yet been given, the parser treats carriage returns (that is, newlines) just like spaces.

Because SKILL reads its input on a form-by-form basis, it is rarely necessary to “continue” an input line. There might be times, however, when you want to break up a long line for aesthetic reasons. In that case, you can tell the parser to ignore a carriage return in the input line simply by preceding it immediately with a backslash (\).

```
string = "This is \
a test."
=> "This is a test."
```

Arithmetic and Logical Expressions

Expressions are SKILL objects that also evaluate to SKILL objects. SKILL performs a computation as a sequence of function evaluations. A SKILL *program* is a sequence of expressions that perform a specified action when evaluated by the SKILL interpreter.

There are two types of primitive expressions in SKILL that pertain to OCEAN: constants and variables.

Constants

A *constant* is an expression that evaluates to itself. That is, evaluating a constant returns the constant itself. Examples of constants are `123`, `10.5`, and `"abc"`.

Variables

A *variable* stores values used during the computation. The variable returns its value when evaluated. Examples of variables are `a`, `x`, and `init_var`.

When the interpreter evaluates a variable whose value has not been initialized, it displays an error message telling you that you have an unbound variable. For example, you get an error message when you misspell a variable because the misspelling creates a new variable.

```
myVariable
```

causes an error message because it has been referenced before being assigned, whereas

```
myVariable = 5
```

works.

When SKILL creates a variable, it gives the variable an initial value of `unbound`. It is an error to evaluate a variable with this value because the meaning of `unbound` is that-value-which-represents-no-value. `unbound` is not the same as `nil`.

Using Variables

You do not need to declare variables in SKILL as you do in C. SKILL creates a variable the first time it encounters the variable in a session. Variable names can contain

- Alphanumeric characters
- Underscores (_)
- Question marks
- Digits

The first character of a variable must be an alphanumeric character or an underscore. Use the assignment operator to store a value in a variable. You enter the variable name to retrieve its value.

```
lineCount = 4              => 4
lineCount                  => 4
lineCount = "abc"          => "abc"
lineCount                  => "abc"
```

Creating Arithmetic and Logical Expressions

Constants, variables, and function calls can be combined with the *infix* operators, such as less than (<), colon (:), and greater than (>) to form arithmetic and logical expressions. For example: `1+2`, `a*b+c`, `x>y`.

You can form arbitrarily complicated expressions by combining any number of the primitive expressions described above.

4

Working with SKILL

This chapter provides information on using SKILL functions. It includes information on the types of SKILL functions, the types of data accepted as arguments, how data types are used, and how to declare and define functions. In this chapter, you can find information about

Skill Functions

There are two basic types of SKILL functions:

- *Functions* carry out statements and return data that can be redirected to other commands or functions.
- *Commands* are functions that carry out statements defined by the command and return `t` or `nil`. Some commands return the last argument entered, but the output cannot be redirected.

Data Types

SKILL supports several data types, including integer and floating-point numbers, character strings, arrays, and a highly flexible linked list structure for representing aggregates of data. The simplest SKILL expression is a single piece of data, such as an integer, a floating-point number, or a string. SKILL data is case sensitive. You can enter data in many familiar ways, including the following:

Sample SKILL Data Items

Data Type	Syntax Example
integer	5
floating point number	5.3
text string	`"Mary had a little lamb"`

For symbolic computation, SKILL has data types for dealing with symbols and functions.

For input/output, SKILL has a data type for representing I/O ports. The table below lists the data types supported by SKILL with their internal names and prefixes.

Data Types Supported by SKILL

Data Type	Internal Name	Prefix
array	array	a
boolean		b
floating-point number	flonum	f
any data type	general	g
linked list	list	l
floating-point number or integer		n
user-defined type		o
I/O port	port	p
symbol	symbol	s
symbol or character string		S
character string (text)	string	t
window type		w
integer number	fixnum	x

Numbers

SKILL supports the following numeric data types:

- Integers
- Floating-point

Both integers and floating-point numbers may use scaling factors to scale their values. For information on scaling factors, see “Scaling Factors” on page 54.

Atoms

An *atom* is any data object that is not a grouping or collection of other data objects. Built into SKILL are several special atoms that are fundamental to the language.

`nil`	The `nil` atom represents both a false logical condition and an empty list.
`t`	The symbol `t` represents a true logical condition.

Both `nil` and `t` always evaluate to themselves and must never be used as the name of a variable.

`unbound`	To make sure you do not use the value of an uninitialized variable, SKILL sets the value of all symbols and array elements initially to `unbound` so that such an error can be detected.

Constants and Variables

Supported constants and variables are discussed in “Arithmetic and Logical Expressions” on page 3-14.

Strings

Strings are sequences of characters; for example, `"abc"` or `"123"`. A string is marked off by quotation marks, just as in the C language; the empty string is represented as `""`. The SKILL parser limits the length of input strings to a maximum of 8,191 characters. There is, however, no limit to the length of strings created during program execution. Strings of more than 8,191 characters can be created by applications and used in SKILL if they are not given as arguments to SKILL string manipulation functions.

When typing strings, you specify

- Printable characters (except a quotation mark) as part of a string without preceding them with the backslash (`\`) escape character

- Unprintable characters and the quotation mark itself by preceding them with the backslash (\) escape character, as in the C language

Arrays

An *array* represents aggregate data objects in SKILL. Unlike simple data types, you must explicitly create arrays before using them so the necessary storage can be allocated. SKILL arrays allow efficient random indexing into a data structure using familiar syntax.

- Arrays are not typed. Elements of the same array can be different data types.
- SKILL provides run-time array bounds checking. The array bounds are checked with each array access during runtime. An error occurs if the index is outside the array bounds.
- Arrays are one dimensional. You can implement higher dimensional arrays using single dimensional arrays. You can create an array of arrays.

Allocating an Array of a Given Size

Use the `declare` function to allocate an array of a given size.

```
declare( week[7] )            => array[7]:9780700
week                          => array[7]:9780700
type( week )                  => array
days = '(monday tuesday wednesday
         thursday friday saturday sunday)
for( day 0 length(week)-1
    week[day] = nth(day days))
```

- The `declare` function returns the reference to the array storage and stores it as the value of `week`.
- The `type` function returns the symbol *array*.

Concatenating Strings (Lists)

Concatenating a List of Strings with Separation Characters (buildString)

`buildString` makes a single string from the list of strings. You specify the separation character in the third argument. A null string is permitted. If this argument is omitted, `buildString` provides a separating space as the default.

```
buildString( '("test" "il") ".")        => "test.il"
buildString( '("usr" "mnt") "/")        => "usr/mnt"
```

```
buildString( '("a" "b" "c"))          => "a b c"
buildString( '("a" "b" "c") "")       => "abc"
```

Concatenating Two or More Input Strings (strcat)

`strcat` creates a new string by concatenating two or more input strings. The input strings are left unchanged.

```
strcat( "l" "ab" "ef" )     => "labef"
```

You are responsible for any separating space.

```
strcat( "a" "b" "c" "d" )     => "abcd"
strcat( "a " "b " "c " "d " ) => "a b c d "
```

Appending a Maximum Number of Characters from Two Input Strings (strncat)

`strncat` is similar to `strcat` except that the third argument indicates the maximum number of characters from *`string2`* to append to *`string1`* to create a new string. *`string1`* and *`string2`* are left unchanged.

```
strncat( "abcd" "efghi" 2)            => "abcdef"
strncat( "abcd" "efghijk" 5)          => "abcdefghi"
```

Comparing Strings

Comparing Two Strings or Symbol Names Alphabetically (alphalessp)

`alphalessp` compares two objects, which must be either a string or a symbol, and returns `t` if *`arg1`* is alphabetically less than *`arg2`*. `alphalessp` can be used with the `sort` function to sort a list of strings alphabetically. For example:

```
stringList = '( "xyz" "abc" "ghi" )
sort( stringList 'alphalessp ) => ("abc" "ghi" "xyz")
```

The next example returns a sorted list of all the files in the login directory:

```
sort( getDirFiles( "~" ) 'alphalessp )
```

Comparing Two Strings Alphabetically (strcmp)

`strcmp` compares two strings. (To simply test if two strings are equal or not, you can use the `equal` command.) The return values for `strcmp` are explained in the following table.

Return Value	Meaning
1	*string1* is alphabetically greater than *string2*.
0	*string1* is alphabetically equal to *string2*.
-1	*string1* is alphabetically less than *string2*.

```
strcmp( "abc" "abb" )=> 1
strcmp( "abc" "abc")=> 0
strcmp( "abc" "abd")=> -1
```

Comparing Two String or Symbol Names Alphanumerically or Numerically (alphaNumCmp)

`alphaNumCmp` compares two string or symbol names. If the third optional argument is not `nil` and the first two arguments are strings holding purely numeric values, a numeric comparison is performed on the numeric representation of the strings. The return values are explained in the following table.

Return Value	Meaning
1	*arg1* is alphanumerically greater than *arg2*.
0	*arg1* is alphanumerically identical to *arg2*.
-1	*arg2* is alphanumerically greater than *arg1*.

Declaring a SKILL Function

To refer to a group of statements by name, use the `procedure` declaration to associate a name with the group. The group of statements and the name make up a SKILL function.

- The name is known as the function name.
- The group of statements is the function body.

To run the group of statements, mention the function name followed immediately by `()`.

The `clearplot` command below erases the Waveform window and then plots a net.

```
procedure( clearplot( netname )
    clearAll( )
    plot( v (netName))
    )
```

Defining Function Parameters

To make your function more versatile, you can identify certain variables in the function body as formal parameters.

When you start your function, you supply a parameter value for each formal parameter.

Defining Local Variables (let)

Local variables can be used to establish temporary values in a function. This is done using the `let` statement. When local variables are defined, they are known only within the `let` statement and are not available outside the `let` statement.

When the function is defined, the `let` statement includes the local variables you want to define followed by one or more SKILL expressions. The variables are initialized to `nil`. When the function runs, it returns the last expression computed within its body. For example:

```
procedure( test ( x )
    let(( a b )
    a=1
    b=2
    x * a+b
    )
)
```

- The function name is `test`.
- The local variables are `a` and `b`.
- The local variables are initialized to `nil`.
- The return value is the value of `x * a + b`.

Skill Function Return Values

All SKILL functions compute a data value known as the return value of the function. Throughout this document, the right arrow (`=>`) denotes the return value of a function call. You can

- Assign the return value to a SKILL variable

- Pass the return value to another SKILL function

Any type of data can be a return value.

Syntax Functions for Defining Functions

SKILL supports the following syntax functions for defining functions. You should use the `procedure` function in most cases.

procedure

The `procedure` function is the most general and is easiest to use and understand.

The `procedure` function provides the standard method of defining functions. Its return value is the symbol with the name of the function. For example:

```
procedure( trAdd( x y )
    "Display a message and return the sum of x and y"
    printf( "Adding %d and %d ... %d \n" x y x+y )
    x+y
    ) => trAdd
trAdd( 6 7 ) => 13
```

Terms and Definitions

function, procedure
: In SKILL, the terms *procedure* and *function* are used interchangeably to refer to a parameterized body of code that can be executed with actual parameters bound to the formal parameters. SKILL can represent a function as both a hierarchical list and as a function object.

argument, parameter
: The terms *argument* and *parameter* are used interchangeably. The actual arguments in a function call correspond to the formal arguments in the declaration of the function.

expression
: A use of a SKILL function, often by means of an operator supplying required parameters.

function body
: The collection of SKILL expressions that define the function’s algorithm.

5

OCEAN Environment Commands

The following OCEAN environment commands let you start, control, and quit the OCEAN environment.

appendPath

```
appendPath( t_dirName1 ... [t_dirNameN])
     => t_dirNameN/nil
```

Description

Appends a new path to the end of the search path list. You can append as many paths as you want with this command.

Arguments

t_dirName1	Directory path.
t_dirNameN	Additional directory path.

Value Returned

t_dirNameN	Returns the last path specified.
`nil`	Returns `nil` and prints an error message if the paths cannot be appended.

Example

```
appendPath( "/usr/mnt/user/processA/models" )
=> "/usr/mnt/user/processA/models"
```

Adds `/usr/mnt/user/processA/models` to the end of the current search path.

```
appendPath( "/usr/mnt/user/processA/models" "/usr/mnt/user/processA/models1")
=> "/usr/mnt/user/processA/models"
```

Adds `/usr/mnt/user/processA/models` and `/usr/mnt/user/processA/models1` to the end of the current search path.

path

```
path( t_dirName1 ... [t_dirNameN])
     => l_pathList/nil
```

Description

Sets the search path for included files.

This command overrides the path set earlier using any of these commands: path, appendPath, or prependPath.

Using this command is comparable to setting the Include Path for the direct simulator, or the `modelPath` for socket simulators in the Virtuoso® Analog Design Environment user interface. You can add as many paths as you want with this command.

Arguments

t_dirName1	Directory path.
t_dirNameN	Additional directory path.

Value Returned

l_pathList	Returns the entire list of search paths specified.
`nil`	Returns `nil` and prints an error message if the paths cannot be set.

Example

```
path( "~/models" "/tmp/models" )
=> "~/models" "/tmp/models"
```

Specifies that the search path includes `/models` followed by `/tmp/models`.

```
path()
=> "~/models" "/tmp/models"
```

Returns the search path last set.

prependPath

```
prependPath( t_dirName1 ... [t_dirNameN])
     => undefined/nil
```

Description

Adds a new path to the beginning of the search path list. You can add as many paths as you want with this command.

Arguments

t_dirName1	Directory path.
t_dirNameN	Additional directory path.

Value Returned

undefined	The return value for this command/function is undefined.
nil	Returns `nil` and prints an error message if the paths cannot be added.

Example

```
prependPath( "/usr/mnt/user/processB/models" )
=> "/usr/mnt/user/processB/models"
```

Adds `/usr/mnt/user/processB/models` to the beginning of the search path list.

```
prependPath( "/usr/mnt/user/processB/models" "/usr/mnt/user/processB/models2")
=> "/usr/mnt/user/processB/models"
```

Adds `/usr/mnt/user/processB/models` and `/usr/mnt/user/processB/models2` to the beginning of the search path list.

```
prependPath()
=> "/usr/mnt/user/processB/models" "~/models" "/tmp/models"
```

Returns the search path last set.

setup

```
setup( [?numberNotation s_numberNotation] [?precision x_precision]
     [?reportStyle s_reportStyle] [?charsPerLine x_charsPerLine]
     [?messageOn g_messageOn] )
     => t / nil
```

Description

Specifies default values for parameters.

Arguments

s_numberNotation

Specifies the notation for printed information.
Valid values: `'suffix`, `'engineering`, `'scientific`, `'none`
Default value: `'suffix`

The format for each value is `'suffix`: `1m`, `1u`, `1n`, etc.; `'engineering`: `1e-3`, `1e-6`, `1e-9`, etc.; `'scientific`: `1.0e-2`, `1.768e-5`, etc.; `'none`.

The value `'none` is provided so that you can turn off formatting and therefore greatly speed up printing for large data files.

x_precision

Specifies the number of significant digits that are printed.
Valid values: `1` through `16`
Default value: `6`

s_reportStyle

Specifies the format of the output of the report command.
Valid values: `spice`, `paramValPair`
Default value: `paramValPair`

The spice format is:

	Param1	Param2	Param3
Name1	value	value	value
Name2	value	value	value
Name3	value	value	value

The paramValPair format is:

Name1
Param1=value Param2=value Param3=value

Name2
Param1=value Param2=value Param3=value

Name3
Param1=value Param2=value Param3=value

x_charsPerLine — Specifies the number of characters per line output to the display.
Default value: `80`

g_messageOn — Specifies whether error messages are turned on.
Valid values: `t`, `nil`
Default value: `t`, which specifies that messages are turned on.

Value Returned

`t` — Returns `t` if the value is assigned to the name.

`nil` — Returns `nil` if there is a problem.

Example

```
setup( ?numberNotation 'engineering )
=> t
```

Specifies that any printed information is to be in engineering mode by default.

```
setup( ?precision 5 )
=> t
```

Specifies that 5 significant digits are to be printed.

```
setup(?numberNotation 'suffix ?charsPerLine 40 ?reportStyle 'spice ?messageOn t)
```

Sets up number notation to `suffix` format, characters per line to `40`, reporting style to `Spice`, and error message to `ON`.

history

```
history( [x_number] )
     => t
```

Description

Displays the command history. By default, it prints the last 20 commands from the current session and the most recently terminated session. More commands can be printed by giving a number as an argument.

Arguments

x_number	The number of previously entered commands to be listed. Default value: `20`

Value Returned

`t`	Returns `t` to indicate that the commands from history have been listed.

Example

```
history
1 simulator('spectre)
2 design( "tests" "simple" "schematic")
3 analysis( 'tran ?start 0 ?stop 1u ?step 10n )
4 run()
=> t
```

Displays the most recently used commands. To reuse any of these commands, use the following methods at the ocean prompt:

- `ocean> !1`

  This executes the command numbered 1, which in this example is `simulator(‘spectre)`.

- `ocean> !des`

  This executes the last command whose prefix starts with `des` in the history. In this example, it is the second command listed, that is, `design( "tests" "simple" "schematic")`.

Note: To run `history` in CIW, the syntax is:

```
<space>!<commandNumber>
```

For example:

```
<space>!1
```

This executes the command numbered 1 from the CIW.

ocnSetSilentMode

```
ocnSetSilentMode( g_silentMode )
    => t
```

Description

Filters out OCEAN warning and information messages and allows only error messages to be written. This functionality is useful while running the OCEAN scripts when you might want to skip all OCEAN messages except errors.

Arguments

g_silentMode	Accepts boolean values `t` or `nil`. Set to `t` to suppress the OCEAN warning and information messages. Set to `nil` to allow all OCEAN messages to be displayed.

Value Returned

`t`	Returns `t` to indicate the successful assignment of the passed argument.

Example

```
ocnSetSilentMode(t) => t
```

Suppresses the ocean warning messages

```
ocnSetSilentMode(nil) => t
```

Displays the ocean warning messages

6

Simulation Commands

The following OCEAN simulation commands let you set up and run your simulation.

ac

```
ac( g_fromValue g_toValue g_ptsPerDec )
     => undefined/nil
```

```
ac( g_fromValue g_toValue t_incType g_points )
     => undefined/nil
```

Description

Specifies an AC analysis.

To know more about this analysis, see the simulator-specific user guide.

Arguments

g_fromValue	Starting value for the AC analysis.
g_toValue	Ending value.
g_ptsPerDec	Points per decade.
t_incType	Increment type. Valid values: For the Spectre® circuit simulator, `"Linear"`, `"Logarithmic"`, or `"Automatic"`. For other simulators, `"Linear"` or `"Logarithmic"`.
g_points	Either the linear or the logarithmic value, which depends on *t_incType*.

Value Returned

undefined	The return value for this command/function is undefined.
`nil`	Returns `nil` and prints an error message if the analysis is not specified.

Example

```
ac(1 10000 2)
```

Specifies an AC analysis from 1 to 10,000 with 2 points per decade.

```
ac(1 10000 "Linear" 100)
```

Specifies an AC analysis from 1 to 10,000 by 100.

```
ac(1 5000 "Logarithmic" 10)
```

Specifies an AC analysis from `1` to `5000` with `10` logarithmic points per decade.

analysis

```
analysis( s_analysisType [?analysisOption1 g_analysisOptionValue1]…
    [?analysisOptionN g_analysisOptionValueN])
    => undefined/nil
```

Description

Specifies the analysis to be simulated.

You can include as many analysis options as you want. Analysis options vary, depending on the simulator you are using. To include an analysis option, replace `analysisOption1` with the name of the desired analysis option and include another argument to specify the value for the option. If you have an AC analysis, the first option/value pair might be `[?from 0]`.

Note: Some simplified commands are available for basic SPICE analyses. See the `ac`, `dc`, `tran`, and `noise` commands. Use the `ocnHelp( 'analysis )` command for more information on the analysis types for the simulator you choose.

Arguments

s_analysisType

Type of the analysis. The valid values for this argument depend on the analyses that the simulator contains.
The basic SPICE2G-like choices: `'tran`, `'dc`, `'ac`, and `'noise`.

?*analysisOption1*

Analysis option. The analysis options available depend on which simulator you use. (See the documentation for your simulator.) If you are using the Spectre® circuit simulator, see the information about analysis statements in the *Virtuoso Spectre Circuit Simulator Reference* for analysis options you can use.

g_analysisOptionValue1

Value for the analysis option.

?*analysisOptionN*

Any subsequent analysis option. The analysis options that are available depend on which simulator you use. (See the documentation for your simulator.)

g_analysisOptionValueN
Value for the analysis option.

Value Returned

undefined — The return value for this command/function is undefined.

nil — Returns `nil` and prints an error message if there is a problem specifying the analysis.

Example

```
analysis( 'ac ?start 1 ?stop 10000 ?lin 100 )
```

For the Spectre® circuit simulator, specifies that an AC analysis be performed.

```
analysis( 'tran ?start 0 ?stop 1u ?step 10n )
```

Specifies that a transient analysis be performed.

```
analysis('dc ?oppoint "rawfile" ?save "allpub"
    ?param "temp" ?start -50 ?stop 100 )
```

Sweeps temperature for the Spectre® circuit simulator.

```
analysis('dc ?saveOppoint t )
```

Saves the DC operating point information for the Spectre® circuit simulator.

```
analysis('xf ?start 0 ?stop 100 ?lin 2 ?dev "v3" ?param "dc" ?freq 1 ?probe "v4")
```

Sets the Spectre transfer function analysis.

```
analysis('sens ?analyses_list list("dcOp" "dc" "ac") ?output_list list("I7:3"
"OUT")
```

Sets the Spectre sensitivity analysis.

```
analysis( 'noise ?start 1 ?stop 10e6 ?oprobe "V4" )
```

Sets the Spectre noise analysis.

```
analysis( 'dcmatch ?oprobe "/PR1" )
analysis( 'dcmatch ?param "temp" ?start "24" ?stop "26 ?lin "5" )
```

Sets the Spectre dcmatch analysis.

```
analysis('pz ?freq "2"  ?readns "./abc"  ?oppoint "rawfile" ?fmax "4500000000"
?zeroonly "no"  ?prevoppoint "no"  ?restart "no" ?annotate "no"  ?stats "no"  )
```

Sets the Spectre pz analysis.

```
analysis('stb ?start "10"  ?stop "10G"  ?dec "10" ?probe "/PR1"  ?prevoppoint "yes"
?readns "./abc"  ?save "lvl" ?nestlvl "1"  ?oppoint "logfile"  ?restart "yes"
?annotate "no" ?stats "yes"  )
```

Sets the Spectre stability analysis.

```
analysis('pss ?fund "100M"  ?harms "3"  ?errpreset "moderate"  )
```

Sets the Spectre pss RF analysis.

```
analysis('pnoise ?start "1K"  ?stop "30M"  ?log "20" ?maxsideband "3"
?oprobe "/rif"  ?iprobe "/rf" ?refsideband "0"  )
```

Sets the Spectre pnoise RF analysis.

```
analysis('pac ?sweeptype "relative"  ?relharmnum ""  ?start "700M" ?stop "800M"
?lin "5"  ?maxsideband "3")
```

Sets the Spectre pac RF analysis.

```
analysis('pxf ?start "10M"  ?stop "1.2G"  ?lin "100" ?maxsideband "3"  ?p "/Plo"
?n "/gnd!"  )
```

Sets the Spectre pxf RF analysis.

```
analysis('qpss ?funds list("flo" "frf")  ?maxharms list("0" "0")
?errpreset "moderate" ?param "prf"  ?start "-25"  ?stop "-10"  ?lin "5"  )
```

Sets the Spectre qpss RF analysis.

```
analysis('qpac ?start "920M"  ?stop ""  ?clockmaxharm "0"  )
```

Sets the Spectre qpac analysis.

```
analysis('sp  ?start "100M"  ?stop "1.2G"  ?step "100" ?donoise "yes"
?oprobe "/PORT0" ?iprobe "/RF"  )
```

Sets the Spectre sp (S - parameter) analysis.

converge

```
converge( s_convName t_netName1 f_value1 ... [t_netNameN f_valueN])
    => undefined/nil
```

Description

Sets convergence criteria on nets.

To know more about convergence, refer to the chapter *Helping a Simulation to Converge* of the *Virtuoso Analog Design Environment L User Guide*.

Arguments

s_convName	Name of the convergence type. Valid values are one of `nodeset ic and forcenode`. Note that `forcenode` is not supported for the spectre simulator.
t_netName1	Name of the net to which you want to set convergence criteria.
f_value1	Voltage value for the net
t_netNameN	Name of the additional net
f_value	Voltage value for the additional net

Value Returned

undefined	The return value for this command/function is undefined.
`nil`	Returns `nil` and prints an error message if the function fails

Example

```
converge( 'ic "/I0/net1" 5 )
```

Sets the convergence name for the initial condition `net1` to `5` volts.

```
converge( 'nodeset "/I0/net1" 5 )
```

Sets the convergence name for nodeset of `net1` to `5` volts.

connectRules

```
connectRules( t_ruleName [?lib t_libName] [?view t_viewName]
     [baseRule t_baseRule] [?moduleInfo l_moduleInfo]
     [?resolutionInfo l_resolutionInfo] [?commonParam l_commonParam]
     [?userDefined s_userDefined]
     )
     => t / nil
```

```
connectRules ( t_ruleName )
     => t / nil
```

```
connectRules( ?none s_tag )
     => t / nil
```

The following arguments are composed of other arguments as described below:

```
l_moduleInfo:       ((s_moduleName1 [?mode s_mode] [?paramInfo l_paramInfo]
                    [?direction1 s_direction1][?discipline1 s_discipline1]
                    [?direction2 s_direction2] [?discipline2 s_discipline2])
                    [(s_moduleName2 - ) -]

l_paramInfo:        ((s_paramName1 s_value1) [(s_paramName2 s_value2) -])

l_resolutionInfo:   ((s_resolvedDiscipline1 s_equivalentDisciplines1)
                    [(s_resolvedDiscipline2 s_equivalentDisciplines2) -])

l_commonParam:      ((s_paramName1 s_value1 [(s_moduleName1 s_moduleName2 - )])
                    [(s_paramName2 s_value2 ) -]
```

Description

Sets connect rules for a given AMS OCEAN session required by the elaborator. To specify multiple connect rules, use this command multiple times. To add a connect rule to an OCEAN session, you can either choose a built-in rule from the `connectLib` library (by specifying `t_ruleName`, `t_libName` and `t_viewName`) or one of your own compiled built-in connect rules (by specifying `t_ruleName`, `t_libName` and `t_viewName`). To add a user defined connect rule to an OCEAN session specify `s_userDefined`. To modify an existing built-in rule, you need to specify `t_baseRule` (the name of the built-in rule that needs be modified), specify a new name (by specifying `t_ruleName`, `t_libName` and `t_viewName`) and also specify one or more of the optional arguments.

You can use the `delete('connectRules)` command to delete one or more specified connect rules. See the examples provided with the delete command.

You can use `ocnDisplay('connectRules)` to view the currently active connect rules in an OCEAN session. You may use `ocnDisplay('connectRules 'all)` to display all information about all active connect rules in an OCEAN session.

Note: This command is applicable only when `ams` is the selected simulator.

Arguments

t_ruleName — Name of the connect rule that you want to use in the current session.

t_libName — Name of the library that contains a list of user-compiled connect rules. If you do not specify this, the connect rules are assumed to be in the default location.

t_viewName — Name of the view of the selected cell.

t_baseRule — Name of the connect rule that you want to modify.

l_moduleInfo — Arguments that need to be updated for a specified connect rule. The arguments may include `s_mode`, `s_direction1`, `s_direction2`, `s_discipline1` and `s_discipline2`.

Valid values for `s_mode` are: null, `split`, `merged.`

`s_direction1` and `s_direction2` work as a pair. Valid combinations are: both null, `input/output`, `output/input`, `inout/inout`.

`s_discipline1` and `s_discipline2` also work as a pair. Either they should both be null or they should both have values.

t_resolutionInfo — Names of disciplines that need to be resolved to another discipline. The value specified overwrites the `l_resolutionInfo` in the base rule or in the existing connect rule.

t_commonParam — One or more parameters that you want to modify for all modules or a set of modules. Although the same result can be achieved by using the `l_moduleInfo` argument, `l_commonParam` facilitates updating parameters for all modules in one go.

s_userDefined — Name of the user defined connect rule that you want to use in the current session. Specify `3step` as the value of `s_userDefined` and specify `t_ruleName`, `t_libName` and `t_viewName` to add a user defined connect rule for the Cellview-based netlister flow. Specify `irun` as the value of `s_userDefined` and specify `t_ruleName`, `s_fileName` or both to add a user defined connect rule for the OSS-based

	netlister with irun flow. Any other argument specified when adding a user defined connect rule will be ignored.
s_tag	The option used to indicate that no connect rules are to be used for the current session.

Value Returned

`t`	Returns `t` if the specifed connect rules are set.
`nil`	Returns `nil` and prints an error message otherwise.

Example

```
connectRules("ConnRules_5V_full")
```

Sets `ConnRules_5V_full` as the current connect rule from the default `connectLib` located in your hierarchy.

```
connectRules("CustomRules_9V_high" ?lib "myConnectLib" ?view "myViewName")
```

Sets `CustomRules_9V_high` from `myConnectLib`, where `myConnectLib` contains a list of user-compiled connect rules and `myViewName` is the specified view name.

```
connectRules("connRule3" ?lib "lib2" ?view "view2" ?baseRule "ConnRules_18V_full"
?description "updated directions" ?moduleInfo ((?name "E2L" ?direction1 "input"
?direction2 "output")))
```

Checks if `connRule3` exists in the session. If it does, it updates `direction1` to `input` and `direction2` to `output` for `E2L` and `description` for this rule. If this rule does not exist, then it takes the base values as values from `ConnRules_18V_full` and updates `direction1`, `direction2`, and `description` and names the new rule as `connRule3`.

```
connectRules("connRule3" ?lib "lib2" ?view "view2" ?moduleInfo ((?name "E2L" ?mode
"split")))
```

Checks if `connRule3` exists. If it does not exist, as no base rule is specified, a relevant error message appears. If the rule exists, it would update `mode` to `split` for the existing connect rule `connRule3` for the module `E2L`.

```
connectRules("connRule3" ?lib "lib2" ?view "view2" ?description "desc123"
?moduleInfo ((?name "E2L" ?mode "split" ?direction1 "input" ?direction2 "output"))
?resolutionInfo nil)
```

If `connRule3` does not exist and the base rule is not specified but `description`, `moduleInfo` and `resolutionInfo` are specified, the connect rule `connRule3` is added with the values specified for `moduleInfo`, `resolutionInfo` and `description`. Note that in this case no checks are done (that is, module names and parameter names are not checked against base information as no base rule information is available). This command is applicable while using the `connectRules` command as saved in ocean.

```
connectRules("connRule3" ?lib "lib2" ?view "view2" ?moduleInfo ((?name "L2E"
?paramInfo (("vsup" "1.7")("vtlo" "3.2")))
```

Updates the parameters `vsup` and `vtlo` for the existing rule `connRule3` in the `L2E` module.

```
connectRules("connRule3" ?lib "lib2" ?view "view2" ?resolutionInfo (("r1" "e1
e2")("r2" "e4 e5")) ?commonParam (("vsup" "1.2") ("vtlo" "3.4" ("L2E" "Bidir"))
```

Updates `resolutionInfo` for the existing connect rule `connRule3`. The old `resolutionInfo` value for this rule is replaced with the new information. It also updates the `vsup` parameter to `1.2` for all `connRule3` modules and updates `vtlo` to `3.4` for the modules `L2E` and `Bidir`.

```
connectRules("connRule3" ?lib "lib2" ?view "view2" ?userDefined 3step)
```

Sets `connRule3` from `view2` of `lib2` as a user defined connect rule for the Cellview-based netlister flow.

```
connectRules("connRule3" ?userDefined irun)
```

Sets `connRule3` from the `connectLib` library as a user defined connect rule for the OSS-based netlister with irun flow.

```
connectRules("connRule3" ?userDefined irun ?file "file1")
```

Sets `connRule3` from `file1` as a user defined connect rule for the OSS-based netlister with irun flow.

```
connectRules(?userDefined irun ?file "file1")
```

No user-defined connect rule name is specified for the OSS-based netlister with irun flow. Hence, the first rule found in `file1` will be used for AMS simulation.

```
connectRules(?none t)
=> t
```

Sets the current connect rule to `None` so that no connect rule is provided to ncelab during elaboration.

```
delete('connectRules list("mylib" "myrule" "myview") list("mylib1" "myrule1"
"myview1"))
```

Deletes the connect rule `myrule` in the library `mylib` with the view `myview`. It also deletes the connect rule `myrule1` in the library `mylib1` with the view `myview1`.

```
delete('connectRules list("" "rule1" ""))
```

Deletes the specified connect rule `rule1` from the default `connectLib` library.

createFinalNetlist

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

Description

Creates the final netlist for viewing purposes. The netlist also can be saved but is not required to run the simulator.

Note: This command works only for socket simulators. For direct simulators, such as spectre, use `createNetlist` instead.

Arguments

None.

Value Returned

`t`	Returns `t` if the final netlist is created.
`nil`	Returns `nil` and prints an error message otherwise.

Example

```
createFinalNetlist()
```

Creates the final netlist for the current simulation run.

createNetlist

```
createNetlist( [?recreateAll g_recreateAll] [?display g_display] )
     => t_filename/nil
```

Description

Creates the simulator input file.

If the design is specified as cellview, this command netlists the design, if required, and creates the simulator input file. When the `g_recreateAll` argument is set to `t` and the design is specified as cellview, all the cells in the design hierarchy are renetlisted, before creating the simulator input file. If the design is specified as netlist file, that netlist is included in the simulator input file. Also see the design function.

When the `g_display` option is set to `t` (or `nil`) the netlist file is displayed (or undisplayed) to the user. By default, `g_display` it set to `'t` (true).

Note: This command does not work with socket simulators.

Arguments

g_recreateAll	Specifies if the netlist needs to be recreated or not.
g_display	Specifies if the netlist is to be displayed or not.

Value Returned

t_fileName	Returns the name of the simulator input file.
nil	Returns `nil` otherwise

Example

```
createNetlist()
=> "/usr/foo/netlist/input.scs"
```

Creates simulator input file for the current simulation run.

```
design( ?lib "test" ?cell "mytest" ?view "spectre")
```

```
createNetlist( ?recreateAll t )
=>"/usr/foo/netlist/input.scs"
```

Netlists and creates simulator input file for the current simulation run.

```
design( "test" "mytest1" "spectre")
```

```
createNetlist( ?recreateAll t ?display nil )
=>"/usr/foo/netlist/input.scs"
```

Netlists and creates simulator input file for the given simulation run but does not display the `input.scs` file in a new window which may be annoying to the user. By default `?display` option is set to `'t` meaning netist file would be displayed. This can be turned ON/OFF via `?display` set to `t/nil`

Note: If you regenerate the netlist after changing the design in a different Virtuoso session, the netlist is not updated with the design changes. To update the netlist with the current cellview, run the `ddsRefresh` command before running the `createNetlist` command as shown below:

```
ddsRefresh( ?cellview t )
=> t
```

```
createNetlist( ?recreateAll t )
=> "/usr/foo/netlist/input.scs"
```

dc

```
dc( t_compName [ t_compParam ] g_fromValue g_toValue g_byValue )
     => undefined/nil
```

Description

Specifies a DC sweep analysis with limited options. If other analysis options are needed, use the `analysis` command.

To know more about this analysis, see the simulator-specific user guide.

Note: *t_compParam* is valid only for the spectre simulator.

Arguments

t_compName	Name of the source (or component, for the Spectre® circuit simulator) to sweep.
t_compParam	For the Spectre® circuit simulator, the component parameter to be swept.
g_fromValue	Starting value for the DC analysis.
g_toValue	Ending value.
g_byValue	The increment at which to step through the analysis.

Value Returned

undefined	The return value for this command/function is undefined.
`nil`	Returns `nil` and prints an error message if the analysis is not specified.

Example

```
dc("v1" "dc" 0 5 1)
dc("r1" "r" 0 5 1)
```

Specifies two DC sweep analyses for the Spectre® circuit simulator.

```
dc("v1" 0 5 1)
```

Specifies one DC sweep analysis for a simulator other than the Spectre® circuit simulator.

definitionFile

```
definitionFile( t_fileName [t_fileName2 ... t_fileNameN ])
     => l_fileNames/nil
```

Description

Specifies definitions files to be included in the simulator input file.

Definitions files define functions and global variables that are not design variables. Examples of such variables are model parameters or internal simulator parameters. To know more about definitions files, see the section *Using a Definitions File* in *Chapter 3* of the *Virtuoso Analog Design Environment L User Guide*.

Note: This command does not work with socket simulators.

Arguments

t_fileName	The name of the definition file that would typically contain functions or parameter statements.

Value Returned

l_fileNames	A list of the file names specified; returned on success.
`nil`	Otherwise `nil` is returned.

Example

```
definitionFile( "functions.def" "constants.def" )
=> ("functions.def" "constants.def")
```

Includes `functions.def` and `constants.def` files in the simulator input file.

```
definitionFile( )
=> ("functions.def" "constants.def")
```

Returns the definition files set earlier.

delete

```
delete( s_command [g_commandArg1] [g_commandArg2] … )
     => t / nil
```

Description

Deletes all the information specified.

The *s_command* argument specifies the command whose information you want to delete. If you include only this argument, all the information for the command is deleted. If you supply subsequent arguments, only information specified by these arguments is deleted, and not all the information for the command.

Arguments

s_command	Command that was initially used to add the items that are now being deleted. Valid values: `analysis`, `connectRules`, `discipline`, `globalSignal`, `desVar`, `path`, `save`, `ic`, `forcenode`, `nodeset`
g_commandArg1	Argument corresponding to the specified command.
g_commandArg2	Additional argument corresponding to the specified command.

Value Returned

`t`	Returns `t` if the information is deleted.
`nil`	Returns `nil` if there is an error.

Example

```
delete( 'save )
=> t
```

Deletes all the saves.

```
delete( 'save 'v )
=> t
```

Deletes *only* the nets. The rest of the information can be saved in subsequent simulations.

```
delete( 'save "net23" )
=> t
```

Deletes only `net23`. The rest of the information can be saved in subsequent simulations.

```
delete( 'monteCarlo )
=> t
```

Turns off the monteCarlo command and sets everything back to the defaults.

design

```
design( t_cktFile | t_lib t_cell t_view [t_mode] )
    => t_cktFile/nil | (t_lib t_cell t_view)/nil
```

Description

Specifies the directory path to the netlist of a design or the name of a design to be simulated.

For the *lib*, *cell*, *view* version of the `design` command, you can specify the mode (`r`, `w` or `a`, representing `read`, `write` or `append`) in which the design should be opened.

Arguments

t_cktFile — Directory path to the netlist followed by the name of the netlist file. Name of the netlist file must be `netlist`. Note that the `netlistHeader` and `netlistFooter` files are also needed in the same directory.

Otherwise, *cktFile* is a pre-existing netlist file from another source. In this case, you might need to remove the `.cards` from the netlist because the OCEAN commands are converted to `.cards` and appended to the final netlist. The simulator might give an error or warning if the `.cards` are read twice.

t_lib — Name of the library that contains the design.

t_cell — Name of the design.

t_view — View of the design (typically `schematic`).

t_mode — The mode in which the design should be opened. The value can be `r`, `w` or `a`, representing `read`, `write` and `append`, respectively. The default mode is `append`. Read-only designs can be netlisted only by direct netlisters, and not socket. The `w` mode should not be used as it overwrites the design.

Value Returned

t_cktFile — Returns the name of the design if successful.

l_(lib cell view)
Returns the name of the view for an Virtuoso® Analog Design Environment design if successful.

`nil` Returns `nil` and prints an error message if there is a problem using the specified design.

Example

Example 1

```
design( "./opampNetlist/netlist" )
=> netlist
```

specifies that `netlist`, a netlist file, be used in the simulation.

Example 2

```
design( "tests" "simple" "schematic" )
=> (tests simple schematic)
```

Specifies that the `schematic` view of the `simple` design from your `tests` library be used in the simulation.

Example 3

```
design("mylib" "ampTest" "schematic" "a")
=> (mylib ampTest schematic)
```

Specifies that the `schematic` view of the `ampTest` design from your `mylib` library be appended to the simulation.

Example 4

```
design()
=> (mylib ampTest schematic)
```

Returns the lib-cell-view being used in the current session. If a design has not been specified, it returns `nil`.

desVar

```
desVar( t_desVar1 f_value1 … [t_desVarN f_valueN])
     => undefined/nil
```

Description

Sets the values of design variables used in your design. You can set the values for as many design variables as you want.

To know more about design variables, refer to the Chapter 3, *Design Variables and Simulation Files for Direct Simulation* of the *Virtuoso Analog Design Environment L User Guide*.

Arguments

t_desVar1	Name of the design variable.
f_value1	Value for the design variable.
t_desVarN	Name of an additional design variable.
f_valueN	Value for the additional design variable.

Value Returned

undefined	The return value for this command/function is undefined.
nil	Returns nil and prints an error message if the assignments fail.

Example

```
desVar( )
```

Returns the design variables set last, if any. Otherwise, it returns nil.

```
desVar( "rs" 1k )
```

Sets the rs design variable to 1k.

```
desVar( "r1" "rs" "r2" "rs*2" )
```

Sets the r1 design variable to rs, or 1k, and sets the r2 design variable to rs*2, or 2k.

```
a = evalstring( desVar( "rs")) / 2
```

Sets a to `1k/2` or `500`.

Note: `evalstring` is necessary because `desVar` returns a string.

discipline

```
discipline( g_discipline1 [g_discipline2 ...] )
     => t / nil
```

Description

Adds discrete disciplines to the existing set of disciplines for a given 'ams' OCEAN session. You can use `delete('discipline)` to delete one or more specified disciplines. You can use `ocnDisplay('discipline)` to view the currently active disciplines in an OCEAN session.

Note: This command is applicable only when `ams` is the simulator.

Arguments

g_discipline1	Name of a discrete discipline to be added.
g_discipline2	Names of additional discrete disciplines to be added.

Value Returned

`t`	Returns `t` if the discipline is added.
`nil`	Returns `nil` or prints an error message otherwise.

Example

```
discipline( "logic1" "logic2" `("logic3") )
```

Disciplines to be added can be either strings or lists containing the discipline name. If no disciplines have been added so far, this sample command adds the three discrete disciplines `logic1`, `logic2` and `logic3` to the session; otherwise, it adds these three disciplines to the existing set of disciplines.

```
discipline("LL")
```

Adds discipline `LL` to the existing set of disciplines. If `logic1`, `logic2` and `logic3` are already added, `LL` is added as the fourth discipline.

```
delete('discipline "logic2" "LL")
```

Deletes disciplines `logic2` and `LL` from the session.

```
delete('discipline)
```

Deletes all the specified disciplines in the session.

displayNetlist

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

Description

Displays the concatenated AMS complete design info file used in a given AMS OCEAN session. The concatenated file displays the cell-based netlisting of the cellviews used in the configuration along with the analog control file and the TCL file generated by AMS-ADE. This command is applicable for both solvers – Spectre and UltraSim.

Note: This command is applicable only when `ams` is the simulator.

Arguments

None.

Value Returned

`t`	Returns `t` if the concatenated design information file.
`nil`	Returns `nil` or prints an error message otherwise.

Example

```
displayNetlist()
=> t
```

Displays the concatenated design information file.

envOption

```
envOption( s_envOption1 g_value1 … [ s_envOptionN g_valueN ] )
     => undefined/nil
```

Description

Sets environment options.

To get the list of environment options that can be set for a simulator, first set the simulator and then run the OCEAN online help command `ocnHelp('envOption)`. For example,

```
simulator('spectre)
ocnHelp('envOptions)
```

The above command displays a list of environment options that can be set for spectre.

Important

To specify an include file, use the `includeFile` command, not the `envOption` command. To set a model path, use the `path` command, not the `envOption` command.

To know more about environment options, see the section *Environment Options* in *Chapter 2* of the *Virtuoso Analog Design Environment L User Guide*.

Arguments

s_envOption1	Name of the first environment option to set.
g_value1	Value for the option.
s_envOptionN	Name of an additional environment option to set.
g_valueN	Value for the option.

Value Returned

undefined	The return value for this command/function is undefined.
`nil`	Returns `nil` if there are problems setting the option.

Example

```
envOption( 'paramRangeCheckFile "./myDir/range.check" )
```

Sets the `paramRangeCheckFile` environment option.

```
envOption( 'initFile "./myDotSFiles/init" )
```

Sets the `initFile` environment option.

```
envOption( 'updateFile "./myDotSFiles/update" )
```

Sets the `updateFile` environment option.

evcdFile

```
evcdFile( t_evcdFileName )
     => t_evcdFileName/nil
```

Description

Sets an EVCD file for a given UltraSim OCEAN session. You also need to specify an EVCD info file while using this command. You can specify only one EVCD file for a session. You may use `ocnDisplay('evcdFile)` to view the currently active EVCD file.

Note: This command is applicable for the UltraSim simulators.

Arguments

t_evcdFileName	The name of the EVCD file to be used for session.

Value Returned

t_evcdFileName	The EVCD file name is the output if the command is successful.
`nil`	Otherwise, `nil` is returned.

Example

```
evcdFile("/tmp/evcdFile.dat")
=> "/tmp/evcdFile.dat"
```

Specifies `/tmp/evcdFile.dat` as the EVCD file to be used for current UltraSim OCEAN session.

evcdInfoFile

```
evcdInfoFile( t_evcdInfoFileName )
     => t_evcdInfoFileName/nil
```

Description

Sets a EVCD info file for a given UltraSim OCEAN session. You also need to specify an EVCD file while using this command. You can specify only one EVCD info file for a session. You may use `ocnDisplay('evcdInfoFile)` to view the currently active EVCD info file.

Note: This command is applicable only for the UltraSim simulator.

Arguments

t_evcdInfoFileName

The name of the EVCD info file to be included.

Value Returned

t_evcdInfoFileName

The EVCD info file name is the output if the command is successful.

`nil` Otherwise, `nil` is returned.

Example

```
evcdInfoFile("/tmp/evcdInfoFile.dat")
=> "/tmp/vcdInfoFile.dat"
```

Specifies `/tmp/evcdInfoFile.dat` as the EVCD file to be used for current UltraSim OCEAN session.

forcenode

```
forcenode( t_netName1 f_value1 … [t_netNameN f_valueN] )
     => undefined/nil
```

Description

Holds a node at a specified value.

To know more about convergence, refer to the chapter *Helping a Simulation to Converge* of the *Virtuoso Analog Design Environment L User Guide*.

Note: This is not available for the spectre simulator. Refer to the documentation for your simulator to see if this feature is available for your simulator.

Arguments

t_netName1	Name of the net.
f_value1	Voltage value for the net.
t_netNameN	Name of an additional net.
f_valueN	Voltage value for the net.

Value Returned

undefined	The return value for this command/function is undefined.
`nil`	Returns `nil` and prints an error message.

Example

```
forcenode( "net1" 5 "net34" 2 )
```

Sets the force nodes of `"net1"` to `5` and `"net34"` to `2`.

globalSigAlias

```
globalSigAlias( g_signalList1 [g_signalList2 ... ] )
     => t / nil
```

Description

Removes all the previous signal aliases and creates the specified aliases. The signal names in each of the signal lists are marked as aliases of each other. Each of the signal lists is a set of signal names that are to be aliased. The signal names should match the names that were specified using the globalSignal command. To unalias all signal, pecify `nil` instead of signal lists.

Note: This command is applicable only when AMS is the simulator.

Arguments

g_signalList(n)	A list of signals that are to be marked as aliases of each other.

Value Returned

`t`	Returns `t` when previous signal aliases have been removed successfully and new aliases are created according to the signal lists provided.
`nil`	Returns `nil` and prints an error message if the function was unsuccessful.

Example

```
globalSigAlias('("sig1" "sig2") '("sig4" 'sig5" 'sig8"))
```

Removes the previous signal aliases and marks `sig1` and `sig2` as aliases of each other and `sig4`, `sig5` and `sig8` as aliases of each other. The signal names in each of the signal lists are marked as aliases of each other.

```
globalSigAlias("signal2" "signal6" "signal3")
```

If there is just one list of signals to be aliased, it can be given without the list. In this case, `signal2`, `signal6` and `signal3` are marked as aliases of each other.

globalSignal

```
globalSignal( ?name t_signalName [?lang t_langName] [?wireType t_wireType]
     [?discipline t_discipline] [?ground t_ground]) )
     => t / nil
```

Description

Adds or modifies a global signal for a given AMS OCEAN session needed by the elaborator. If the global signal already exists in the session, the values are updated. If it does not exist, a global signal with the specified name is added. In case of a vector signal, the range information can be appended with the name of the signal.

Note: This command is applicable only when AMS is the simulator.

Arguments

t_signalName	The name of the global signal.
t_langName	The namespace within which the signal is entered. It is used to map the signal name to Verilog-AMS. Valid Values: `CDBA`, `Spectre`, `Spice`, `Verilog-AMS` Default Value: `CDBA`
t_wireType	Indicates the Verilog type of the signal declaration. Valid Values: `wire`, `supply0`, `supply1`, `tri`, `tri0`, `tri1`, `triand`, `trior`, `trireg`, `wand`, `wor`, `wreal` Default Value: `wire`
t_discipline	A string value to indicate the discipline of the signal.
t_ground	Indicates if the signal is a ground signal or not. Valid Values: `YES`, `NO` Default Value: `NO`

Value Returned

`t`	Returns `t` when a global signal has been successfully added or modified.
`nil`	Returns `nil` and prints an error message if the function was unsuccessful.

Example

```
globalSignal("signal1" ?wireType "tri")
```

Adds the global signal `signal1` with wire type as `tri`, default language as `CDBA`, and ground as `NO` to the list of global signals if it has not already been added. If it already exists, then it updates the wire type for signal1.

```
globalSignal("signal2" ?lang "Spectre" ?discipline "electrical")
```

Adds `signal2` with language as Spectre, discipline as `electrical`, and ground as `NO` to the list of global signals if it is not already added. If it already exists, then it updates language to `Spectre` and discipline to `electrical`.

```
delete('globalSignal "sig1" "sig2")
```

Deletes `sig1` and `sig2` after unaliasing them if they are in aliased sets.

```
delete('globalSignal)
```

Deletes all user-specified global signals.

ic

```
ic( t_netName1 f_value1 … [t_netNameN f_valueN] )
    => undefined/nil
```

Description

Sets initial conditions on nets in a transient analysis.

To know more about convergence, refer to the chapter *Helping a Simulation to Converge* of the *Virtuoso Analog Design Environment L User Guide*.

Arguments

t_netName1	Name of the net.
f_value1	Voltage value for the net.
t_netNameN	Name of an additional net.
f_valueN	Voltage value for the net.

Value Returned

undefined	The return value for this command/function is undefined.
`nil`	Returns `nil` and prints an error message.

Example

```
ic( "/net1" 5 "/net34" 2 )
```

Holds the nodes of `"/net1"` at `5` and `"/net34"` at `2`.

includeFile

```
includeFile( t_fileName )
     => t_fileName/nil
```

Description

Includes the specified file in the final netlist of the simulator for the current session.

Notes:

1. This command is not available for the direct simulator. Use the `modelFile` or `stimulusFile` command instead.
2. Using this command is comparable to using the Environment Options form of the Virtuoso® Analog Design Environment to name an include file and specify that the syntax for the file be that of the target simulator. If you want the include file to be in Cadence-SPICE circuit simulator syntax, you must edit the raw netlist file (which has a `.c` or `.C` suffix), and manually add the include file.

Arguments

t_fileName	Name of the file to include in the final netlist.

Value Returned

t_fileName	Returns the name of the file if successful.
`nil`	Returns `nil` and prints an error message otherwise.

Example

```
includeFile( "~/projects/nmos" )
=> "~/projects/nmos"
```

Includes the `nmos` file in the final netlist of the simulator for the current session.

```
includeFile()
=>"~/projects/nmos"
```

Returns the includeFile, if one was set earlier. Otherwise, it returns `nil`.

modelFile

```
modelFile( [g_modelFile1 [g_modelFile2 …]] )
     => l_modelFile
```

Description

Specifies model files to be included in the simulator input file.

This command returns the model files used. When model files are specified through the arguments, the model files are set accordingly. Use of full paths for the model file is recommended.

Arguments

g_modelFile1	This argument can be a string to specify the name of the model file.
g_modelfile2	This argument can be a list of two strings to specify the name of the model file and the name of the section.

Value Returned

l_modelfile	A list of all the model file/section pairs.
`nil`	Returned when no file section pairs have been specified with the current call or a previous call of this command. The `nil` value is also returned when an error has been encountered.

Example

```
modelFile( "bjt.scs" "nmos.scs" )
=>( ("bjt.scs" "") ("nmos.scs" "") )
```

```
modelFile( "bjt.scs" '("nmos.scs" "typ") 'my_models )
=> ( ("bjt.scs" "") ("nmos.scs" "typ") ("my_models" "") )
```

```
modelFile()
=> ( ("bjt.scs" "") ("nmos.scs" "") )
```

Returns the modelFile, if one was set earlier. Otherwise, it returns `nil`.

nodeset

```
nodeset( t_netName1 f_value1 … [t_netNameN f_valueN])
     => undefined/nil
```

Description

Sets the initial estimate for nets in a DC analysis, or sets the initial condition calculation for a transient analysis.

To know more about convergence, refer to the chapter *Helping a Simulation to Converge* of the *Virtuoso Analog Design Environment L User Guide*.

Arguments

t_netName1	Name of the net.
f_value1	Voltage value for the net.
t_netNameN	Name of an additional net.
f_valueN	Voltage value for the net.

Value Returned

undefined	The return value for this command/function is undefined.
`nil`	Returns `nil` and prints an error message otherwise.

Example

```
nodeset( "net1" 5 "net34" 2 )
```

Sets the initial estimates of `"net1"` to `5` and `"net34"` to `2`.

noise

```
noise( t_output t_source )
     => undefined/nil
```

Description

Specifies a noise analysis.

Note: This command cannot be used with the spectre simulator.

Arguments

t_output	Output node.
t_source	Input source.

Value Returned

undefined	The return value for this command/function is undefined.
nil	Returns `nil` and prints an error message If there is a problem specifying the analysis.

Example

```
noise( "n1" "v1" )
```

Specifies a noise analysis.

ocnCloseSession

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

Description

Closes the current OCEAN session without saving any settings made during the session. The command has no effect if no session is currently active.

Value Returned

`t`	Returns t when the current session is successfully closed.
`nil`	Returns `nil` if there is a problem closing the active session.

Example

```
ocnCloseSession()
=> t
```

Closes the current OCEAN session.

ocnDisplay

```
ocnDisplay([?output t_filename | p_port] s_command [g_commandArg1]
     [g_commandArg2] … )
     => t / nil
```

Description

Displays all the information specified.

The *s_command* argument specifies the command whose information you want to display. If you include only this argument, all the information for the command displays. If you supply subsequent arguments, only those particular pieces of information display as opposed to displaying all the information for that command. If you provide a filename as the `?output` argument, the `ocnDisplay` command opens the file and writes the information to it. If you provide a port (the return value of the SKILL `outfile` command), the `ocnDisplay` command appends the information to the file that is represented by the port.

Arguments

t_filename	File in which to write the information. The `ocnDisplay` command opens the file, writes to the file, then closes the file. If you specify the filename without a path, the `ocnDisplay` command creates the file in the directory pointed to by your Skill Path. To find out what your Skill path is, type `getSkillPath()` at the OCEAN prompt.
p_port	Port (previously opened with `outfile`) through which to append the information to a file. You are responsible for closing the port. See the outfile command for more information.
s_command	Command that was initially used to add the items that are now being displayed. Valid values: Most simulation setup commands. The commands that are supported include `design`, `analysis`, `tran`, `ac`, `dc`, `noise`, `resultsDir`, `temp`, `option`, `desVar`, `path`, `includeFile`, `modelFile`, `stimulusFile`, `definitionFile`, `saveOption`, `envOption`, `save`, `converge`, `ic`, `forcenode`, `nodeset`, `simulator`, `setup`, `restore`, `saveSubckt`
g_commandArg1	Argument corresponding to the specified command.

g_commandArg2 Additional argument corresponding to the specified command.

Value Returned

t	Displays the information and returns t.
nil	Returns nil and prints an error message if there are problems displaying the information.

Example

```
ocnDisplay( 'optimizeGoal )
=> t
```

Displays all the optimizeGoal information.

```
ocnDisplay( 'analysis 'tran )
=> t
```

Displays only transient analyses.

```
ocnDisplay( 'save )
=> t
```

Displays all the keeps.

```
ocnDisplay( ?output myPort 'analyis )
=> t
```

Displays and writes all the analyses to the port named myPort.

ocnDspfFile

```
ocnDspfFile( t_dspfFile [t_dspfFile1 … t_dspfFileN] )
     => t_dspfFile(s)/nil
```

Description

Sets the parasitic (dspf, spf) files to be used in a Spectre OCEAN session. You can use this command to specify a list of parasitic files to be included in the control file. You can use `ocnDisplay('dspfFile)` to view the currently active parasitic (dspf, spf) files in an OCEAN session.

Note: This command is applicable for Spectre simulator. For AMS, it works only when Spectre is selected as the solver.

Arguments

t_dspfFile	The name of the parasitic (dspf, spf) file to be included.
t_dspfFile1…t_dspfFileN	The name of the additional parasitic (dspf, spf) files to be included.

Value Returned

t_dspfFile	Lists the names of the parasitic (dspf, spf) files.
nil	Returns `nil` if there are problems displaying the information.

Example

```
ocnDspfFile("/tmp/file1.dspf" "/tmp/file2.dspf")
=> ("/tmp/file1.dspf" "/tmp/file2.dspf")
```

Displays the `/tmp/file1.dspf` and `/tmp/file2.dspf` parasitic files to be used for current Spectre OCEAN session.

ocnSpefFile

```
ocnSpefFile( t_SpefFile [t_SpefFile1 … t_SpefFileN] )
     => t_SpecFile(s)/nil
```

Description

Sets the parasitic (spef) files to be used in a Spectre OCEAN session. You can use this command to specify a list of parasitic files to be included in the control file. You can use `ocnDisplay('SpefFile)` to view the currently active parasitic (spef) files in an OCEAN session.

Note: This command is applicable for Spectre simulator. For AMS, it works only when Spectre is selected as the solver.

Arguments

t_SpefFile	The name of the parasitic (spef) file to be included.
t_SpefFile1…t_SpefFileN	The name of the additional parasitic (spef) files to be included.

Value Returned

t_SpefFile	Lists the names of the parasitic (spef) files.
nil	Returns `nil` if there are problems displaying the information.

Example

```
ocnSpefFile("/tmp/file1.spef" "/tmp/file2.spef")
=> ("/tmp/file1.spef" "/tmp/file2.spef")
```

Displays the `/tmp/file1.spef` and `/tmp/file2.spef` parasitic files to be used for current Spectre OCEAN session.

ocnPspiceFile

```
ocnPspiceFile(
     t_PSpiceFile
     [t_PSpiceFile1 …
     t_PSpiceFileN]
     )
     => t_PSpiceFile(s)/nil
```

Description

Sets the PSpice files to be used in a Spectre OCEAN session. Use this command to specify a list of PSpice files to be included in the control file.

Note: This command is applicable for the Spectre simulator. For AMS, it works only when Spectre is selected as the solver.

Arguments

t_PSpiceFile	The name of the PSpice file to be included.
t_PSpiceFile1…t_PSpiceFileN	The name of the additional PSpice files to be included.

Value Returned

t_PSpiceFile	Lists the names of the PSpice files.
nil	Returns `nil` if there are problems displaying the information.

Example

```
ocnPspiceFile("/tmp/file1.sp" "/tmp/file2.sp")
=> ("/tmp/file1.sp" "/tmp/file2.sp")
```

Returns the `/tmp/file1.sp` and `/tmp/file2.sp` PSpice files to be used for the current Spectre OCEAN session.

ocnGetAdjustedPath

```
ocnGetAdjustedPath( t_libName t_cellName t_viewName t_netName )
     => t_adjustedPath/nil
```

Description

Reduces the given hierarchical net path to the shortest hierarchical name that is equivalent to this net.

Arguments

t_libName	Library name of the top cellview of the design.
t_cellName	Cell name of the top cellview of the design.
t_viewName	View name of the top cellview of the design.
t_netName	A single concatenated string for the instance hierarchy with "/" as the hierarchy separator in the string.

Value Returned

t_adjustedPath	The reduced net name. If the net is local to this cell view only, the reduced net name is the same as the provided net name.
`nil`	Returns `nil` if there is a problem returning the adjusted path.

Example

```
ocnGetAdjustedPath( "mylib" "test" "schematic" "I7/I3/gnd")
=> "/gnd"
```

The return value is simply `"/gnd"` because the gnd net is connected from this point up to the top level of hierarchy.

ocnGetInstancesModelName

```
ocnGetInstancesModelName( [l_instance] )
     => l_instance/nil
```

Description

This function returns the model name used by the instance in opened simulation results.

Arguments

l_instance	Name of the instance in the simulation result or the schematic.

Value Returned

l_instance	The list of instance names and models used by instance.
nil	Returns `nil` if no result is open.

Examples

```
ocnGetInstancesModelName()
=> (("/I8/Q4" "trpnp")
     ("/I8/Q3" "trpnp")
     ("/I8/Q2" "trpnp")
     ("/I8/Q1" "trnpn")
     ("/I8/Q0" "trnpn")
     ("/I8/C0" "capacitor")
     ("/I2" "isource")
     ("/I8/M1" "trpmos")
     ("/I8/M3" "trpmos")
     ("/I8/M2" "trnmos")
     ("/I8/M5" "trnmos")
     ("/R1" "resistor")
     ("/R0" "resistor")
     ("/I8/R0" "resistor")
     ("/V2" "vsource")
     ("/I1/V2" "vsource")
     ("/I1/V0" "vsource")
 )
```

```
ocnGetInstancesModelName("/R1")
=> ("/R1" "resistor")

ocnGetInstancesModelName(list("/R1" "/I8/Q1"))
=> (("/R1" "resistor") ("/I8/Q1" "trnpn") )
```

off

```
off( s_command [g_commandArg1] [g_commandArg2] … )
     => t / nil
```

Description

Turns off the specified information.

This command is currently available only for the analysis and restore commands. The first argument specifies the command whose information you want to turn off. If you include only this first argument, all the information for the command is turned off. If you supply subsequent arguments, only those particular pieces of information are turned off as opposed to turning off all the information for that command. The information is not deleted and can be used again.

Arguments

s_command	Command that was initially used to add the items that are now being turned off. Valid value: `restore`
g_commandArg1	Argument corresponding to the specified command.
g_commandArg2	Additional argument corresponding to the specified command.

Value Returned

`t`	Returns `t` if the information is turned off.
`nil`	Returns `nil` and prints an error message if there are problems turning off the information.

Example

```
off( 'restore )
=> t
```

Turns off the `restore` command.

```
off( restore 'tran )
=> t
```

Turns off the transient `restore` command.

option

```
option( [?categ s_categ] s_option1 g_value1 [s_option2 g_value2] … )
     => undefined/nil
```

Description

Specifies the values for built-in simulator options. You can specify values for as many options as you want.

Arguments

s_categ	Type of simulator to be used. Valid values: `analog` if the options are for an analog simulator, `digital` for a digital simulator, or `mixed` for a mixed-signal simulator Default value: `analog`
s_option1	Name of the simulator option.
g_value1	Value for the option.
s_option2	Name of an additional simulator option.
g_value2	Value for the option.

Value Returned

undefined	The return value for this command/function is undefined.
`nil`	Returns `nil` and prints an error message if there are problems setting the option.

Example

```
option( 'abstol 1e-10 )
```

Sets the `abstol` option to `1e-10`.

```
option( 'delmax 50n )
```

Sets the `delmax` option to `50n`.

```
option()
```

Returns the category list for simulation options, including analog, digital, and mixed.

```
option(?categ 'analog)
```

Returns all the simulator options for the analog simulator currently set. For example, if the set simulator is spectre, it returns the valid simulator options for spectre.

restore

```
restore( s_analysisType t_filename )
     => undefined/nil
```

Description

Tells the simulator to restore the state previously saved to a file with a `store` command.

This command is not available for the Spectre® circuit simulator, with which you can use the store/restore options: `readns`, `readforce`, `write`, or `writefinal`.

Note: `Restore` is available for the cdsSpice and hspiceS simulators.

Arguments

s_analysisType	Type of the analysis. Valid values: `dc` or `tran`
t_filename	Name of the file containing the saved state.

Value Returned

undefined	The return value for this command/function is undefined.
`nil`	Returns `nil` and prints an error message if there are problems restoring the information.

Example

```
restore( 'dc "./storeFile" )
=> ./storeFile
```

Initializes the simulator to the state saved in the `storeFile` file.

```
restore( 'tran "./tranStoreFile" )
=> ./tranStoreFile
```

Initializes the simulator to the state of a transient analysis saved in the `tranStoreFile` file.

resultsDir

```
resultsDir( t_dirName )
     => undefined/nil
```

Description

Specifies the directory where the PSF files (results) are stored.

If you do not specify a directory with this command, the PSF files are placed in `../psf` to the `netlist` directory.

Note: The directory you specify with `resultsDir` is also where the *`simulator`*`.out` file is created.

Note: Some simulators are designed to *always* put their results in a specific location. For these simulators, `resultsDir` has no effect. You might use this command when you want to run several simulations using the same design and want to store each set of results in a different location. If this command is not used, the results of an analysis are overwritten with each simulation run.

Arguments

t_dirName	Directory where the PSF files are to be stored.

Value Returned

undefined	The return value for this command/function is undefined.
nil	Returns `nil` and prints an error message if there is a problem with that directory.

Example

```
resultsDir("~/simulation/ckt/spectre/schematic/psf")=>
    "~/simulation/ckt/spectre/schematic/psf"
```

Specifies the `psf` directory as the directory in which to store the PSF files.

```
resultsDir() => "~/simulation/ckt/spectre/schematic/psf"
```

Returns the results directory.

run

```
run([?jobName t_jobName] [?drmsCmd t_drmsCmd])
     => s_jobName/nil
```

run([*analysisList*] [?jobName *t_jobName*] [?host *t_hostName*]
[?queue *t_queueName*] [?startTime *t_startTime*] [?termTime *t_termTime*]
[?dependentOn *t_dependentOn*] [?mail *t_mailingList*] [?block *s_block*]
[?notify *s_notifyFlag*] [?lsfResourceStr *s_lsfResourceStr*])
=> *s_jobName*/nil

```
run( )
     => t_dirName/nil
```

```
run(s_analysisType1 - s_analysisTypeN)
     => t_dirName/nil
```

Description

Starts the simulation or specifies a time after which an analysis should start. If distributed processing is not available on the system or is not enabled, the arguments specific to distributed processing (see *Arguments* section below for list of arguments specific to distributed processing) are ignored and the simulation runs locally. If distributed processing is available and is enabled, the environment default values are used if not specified in the `run` command arguments. The environmental default values are stored in the `.cdsenv` file.

Do not use the `run` command to start the parametric analysis. Instead, use the command that is specific to the analysis.

To start	Use this command
parametric analyses	paramRun

Arguments

analysisList — List of analyses to be run with the `run` command.

s_analysisType1 — Name of a prespecified analysis to be simulated.

s_analysisTypeN — Name of another prespecified analysis to be simulated.

The following arguments apply only when the distributed processing mode is enabled:

t_jobName	If the name given is not unique, an integer is appended to create a unique job name.
t_hostName	Name of the host on which to run the analysis. If no host is specified, the system assigns the job to an available host.
t_queueName	Name of the queue. If no queue is defined, the analysis is placed in the default queue.
t_startTime	Desired start time for the job. If dependencies are specified, the job does not start until all dependencies are satisfied.
t_termTime	Termination time for job. If the job has not completed by the specified termination time, the job is aborted.
t_dependentOn	List of jobs on which the specified job is dependent. The job is not started until dependent jobs are completed.
t_mailingList	List of users to be notified when the analysis is complete.
s_block	When *s_block* is not set to `nil`, the OCEAN script halts until the job is complete. Default value: `nil`
s_notifyFlag	When not set to `nil`, the job completion message is echoed to the OCEAN interactive window. Default value: `t`
s_lsfResourceStr	An LSF Resource Requirement string to submit a job. It is effective only in the LSF mode.
sgeHardResourceStr	Requirements for hardware resources for the job to be run in the SGE mode.
sgeSoftResourceStr	Requirements for software resources for the job to be run in the SGE mode.
sgePriority	Priority for the job being submitted in the SGE mode.
sgeNoOfProcessors	Number of processors to be used in the SGE mode.
sgeParallelEnvName	Name of the parallel environment to be used in the SGE mode.

t_drmsCmd — A DRMS (Distributed Resource Management System) command, such as a bsub command for LSF or a qsub command for SGE (Sun Grid Engine) used to submit a job. When this argument is used, all other arguments, except ?jobName will be ignored. Moreover, it will not be possible to call the OCEAN function wait on the jobs submitted using this argument.

To know more about the command option, refer to the section Submitting a Job in the chapter Using the Distributed Processing Option in the Analog Design Environment of the Virtuoso Analog Distributed Processing OptionUser Guide.

Value Returned

s_jobName — Returns the job name of the job submitted. The job name is based on the `jobName` argument. If the job name submitted is not unique, a unique identifier is appended to the job name. This value is returned for nonblocking distributed mode.

t_dirName — Returns the name of the directory in which the results are stored. This value is returned for local and blocking distributed modes.

`nil` — Returns `nil` and prints an error message if there is an error in the simulation. In this case, look at the *yourSimulator*.out file for more information. (This file is typically located in the `psf` directory.)

Example

```
run(?jobName “job1” ?drmsCmd “bsub -q lnx32”)
=> s_jobName/nil
```

where `lnx32` is the name of the queue to which the job is submitted.

```
run( )
=> t
```

Starts the simulation.

```
run(’tran, ’ac)
```

Runs only the `tran` and `ac` analyses.

```
run(’dc)
```

Runs only the `dc` analysis.

```
run( ?jobName ?block "nil")
=> 'reconFilter
```

Returns a job name of `reconFilter` for the specified job and runs that job if distributed processing is enabled. The job is submitted nonblocking. The actual job name is returned.

```
run( ?queue "fast" )
```

Submits the current design and enabled analyses as a job on the `fast` queue, assuming that distributed processing is available and enabled.

```
run( ?jobName "job1" ?queue "fast" ?host "menaka" ?startTime "22:59"
?termTime "23:25" ?mail "preampGroup")
```

Submits the current design and enabled analyses as a jobName `job1` on the fast queue host `menaka` with the job start time as `22:59` and termination time as `23:25`. A mail will be sent to preampGroup after the job ends.

```
run( ?jobName "job1" ?queue "fast" ?host "menaka" ?lsfResourceStr "mem>500")
```

Submits the current design and enabled analyses as a jobName `job1` on the `fast` queue host `menaka`, if the host has at least `500` MB of RAM memory.

save

```
save( [?categ s_categ] s_saveType [t_saveName1] … [t_saveNameN] )
     => undefined/nil
```

Description

Specifies the outputs to be saved and printed during simulation.

When specifying particular outputs with `saveName`, you can include as many outputs as you want. If you want to turn off the default of `save`, `'allv`, use the `delete( 'save )` command.

Arguments

s_categ

Type of simulator to be used.
Valid values: `analog`, `digital`
Default value: `analog`
Note: `digital` is not available.

s_saveType

Type of outputs to be saved.
Valid values:

Valid Values	Description
`v`	Specifies that a list of subsequent net names be saved.
`i`	Specifies that a list of subsequent currents be saved.
`all`	Specifies that all nets and all currents are to be saved.
`allv`	Specifies that all voltages are to be saved.
`alli`	Specifies that all currents are to be saved.

Default value: `allv`

t_saveName1

Name of the net, device, or other object.

t_saveNameN

Name of another net, device, or object.

Value Returned

undefined	The return value for this command/function is undefined.
nil	Returns `nil` and prints an error message if there is a problem saving the outputs.

Example

```
save( 'v "net34" "net45" )
```

Saves the outputs for `net34` and `net45`.

```
save( 'i "R1" "/Q1/b" )
```

Saves the currents for `R1` and `Q1/b`.

```
save( 'all )
```

Saves all the nets and currents.

```
save( 'i "q1:b" "r1:p" "mn1:d" )
```

For the spectre simulator, saves the current through the specified devices.

```
save( ?categ 'analog 'v "/vin" "/vout" )
```

Saves the output for `vin` and `vout`.

```
save( 'i "i(q1,b)" "i(r1)" "i(mn1,d)" )
```

For the Cadence-SPICE circuit simulator, saves the current through the same devices.

saveOption

```
saveOption([s_option1 g_optionValue1]…[s_optionN g_optionValueN])
    => undefined/nil
```

Description

Specifies save options to be used by the simulator.

You can include as many save options as you want. To include a save option, replace *s_option1* with the name of the desired save option and include another argument to specify the value for the option.

When you use the `saveOption` command without specifying any arguments, the command returns a list of option and value pairs.

Save options vary, depending on the simulator and interface that you are using. If you are using the Spectre® circuit simulator, for example, you can type the following at an OCEAN prompt to see which options you can set with the `saveOption` command:

```
simulator('spectre)
ocnHelp('saveOption)
```

See the *Virtuoso Spectre Circuit Simulator User Guide* for more information on these options.

Note: The `saveOption` command does not work with socket simulators. If you are using a socket simulator, you must instead specify save options with the `save` command described in "save" on page 143.

Arguments

s_option1

Save option. The save options that are available depend on which simulator you use. (See the documentation for your simulator.)

g_optionValue1

Value for the save option.

s_optionN

Any subsequent save option. The save options that are available depend on which simulator you use. (See the documentation for your simulator.)

g_optionValueN Value for the save option.

Value Returned

undefined The return value for this command/function is undefined.

`nil` Returns `nil` if there are problems specifying options.

Example

```
saveOption( 'save "lvl" 'nestlvl 10 'currents "selected" 'useprobes "yes"
'subcktprobelvl 2 ?saveahdlvars "all")
```

simulator

```
simulator( s_simulator )
     => s_simulator/nil
```

Description

Starts an OCEAN session and sets the simulator name for that session. The previous session (if any) is closed and all session information is cleared.

Arguments

s_simulator	Name of the simulator.

Value Returned

s_simulator	Returns the name of the simulator.
`nil`	Returns `nil` and prints an error message if the simulator is not registered with the Virtuoso® Analog Design Environment through OASIS. If the simulator is not registered, the simulator from the preceding session is retained.

Example

```
simulator( 'spectre )
=> spectre
```

Specifies that the Spectre® circuit simulator be used for the session.

```
simulator()
=> spectre
```

Returns the simulator that you set for the session. If a simulator was not specified, it returns `nil`.

solver

```
solver( s_solver )
     => s_solver/nil
```

Description

Sets a solver for a given AMS OCEAN session. The valid values for solver are `Spectre` and `UltraSim`. You select `Spectre` if you want to use an accurate AMS-Spectre analog engine. You select `UltraSim` if you want to use the AMS-Ultrasim or FastSPICE(UltraSim) solver for a given AMS simulation.

Note: This command is applicable only when `ams` is the simulator.

Arguments

s_solver	Name of the solver.

Value Returned

s_solver	Returns the name of the solver.
`nil`	Returns `nil` and prints an error message if the specified solver is not registered with the Virtuoso® Analog Design Environment through OASIS. If the solver is not registered, the solver from the preceding session is retained.

Example

```
solver( 'spectre )
=> spectre
```

Specifies AMS-Spectre as the solver to be used for the current AMS session.

```
solver( 'ultraSim )
=> ultraSim
```

Specifies AMS-UltraSim (UltraSim FastSPICE) as the solver to be used for the current AMS session.

stimulusFile

```
stimulusFile( t_fileName [t_fileName2 … t_fileNameN ] [?xlate g_xlate] )
     => l_fileNames/nil
```

Description

Specifies stimulus files to be used by the simulator.

When the *g_xlate* variable is set to `t`, the schematic net expressions [#net] and instance name expression [$instance] in the stimulus file are mapped into simulator names before including. When a netlist is specified as the design, this option must be set to `nil`.

Note: This command does not work with socket simulators.

Arguments

t_fileName	The name of the stimulus file to be included.
t_fileName2…t_fileNameN	The names of the additional stimulus files to be included.
g_xlate	If set to `t`, net and instance expressions are translated to simulator names. The default value of the *g_xlate* variable is `t`.

Value Returned

l_fileNames	A list of the stimulus file names is the output if the command is successful.
`nil`	Otherwise `nil` is returned

Example

```
stimulusFile( "tran.stimulus rf.stimulus" ?xlate nil)
=> ("tran.stimulus rf.stimulus")
```

Includes `tran.stimulus` and `rf.stimulus` in the simulator input file. No net and instance expressions are translated.

```
stimulusFile()
=> ("tran.stimulus" "rf.stimulus")
```

Returns the stimulusFile, if one was set earlier. Otherwise, it returns `nil`.

store

```
store( s_analysisType t_filename )
    => t_filename/nil
```

Description

Requests that the simulator store its node voltages to a file.

You can restore this file in a subsequent simulation to help with convergence or to specify a certain starting point. This command is not available for the Spectre® circuit simulator, with which you can use the store/restore options: `readns`, `readforce`, `write`, or `writefinal`.

Note: `store` is available for the cdsSpice and hspiceS simulators.

Arguments

s_analysisType	Type of the analysis. Valid values: `dc` or `tran`
t_filename	Name of the file in which to store the simulator's node voltages.

Value Returned

t_filename	Returns the filename.
`nil`	Returns `nil` and prints an error message if there are problems storing the information to a file.

Example

```
store( 'dc "./storeFile" )
=> ./storefile
```

Stores the simulator's node voltages in a file named `storeFile` in the current directory.

```
store( 'tran "./tranStoreFile" )
=> ./transtorefile
```

Stores the node voltages for a transient analysis in a file named `tranStoreFile` in the netlist (design) directory unless a full path is specified.

temp

```
temp( f_tempValue )
     => s_tempValue/nil
```

Description

Specifies the circuit temperature.

Arguments

f_tempValue	Temperature for the circuit.

Value Returned

s_tempValue	Returns the temperature specified.
nil	Returns nil and prints an error message if there are problems setting the temperature.

Example

```
temp( 125 )
=> ?125?
```

```
atof(temp( 125 ))
=> 125.0
```

Sets the circuit temperature to 125.

```
temp()
=> 125
```

Gets the value you had set for the circuit temperature. If you have not set a value for the temperature, it returns the default value.

tran

```
tran( g_fromValue g_toValue g_byValue )
     => g_byValue/nil

tran( g_toValue)
     => undefined/nil
```

Description

Specifies a transient analysis with limited options. If other analysis options are needed, use the analysis command.

To know more about this analysis, see the simulator-specific user guide.

Note: The second instance of the `tran` command is valid only with the spectre simulator.

Arguments

g_fromValue	Starting time for the analysis.
g_toValue	Ending time.
g_byValue	Increment at which to step through the analysis.

Value Returned

undefined	The return value for this command/function is undefined.
`nil`	Returns `nil` and prints an error message if the analysis is not specified.

Example

```
tran( 1u)
=> "1e-06"
```

Specifies a transient analysis to 1u for the Spectre® circuit simulator

```
tran( 0 1u 1n )
=> "1e-09"
```

Specifies a transient analysis from 0 to 1u by increments of 1n.

vcdFile

```
vcdFile( t_vcdFileName )
     => t_vcdFileName/nil
```

Description

Sets a VCD file for a given AMS or UltraSim OCEAN session. You also need to specify a VCD info file while using this command. You can specify only one VCD file for a session. You may use ocnDisplay('vcdFile) to view the currently active VCD file.

Note: This command is applicable for AMS and UltraSim simulators. For AMS, it works only when UltraSim is the solver.

Arguments

t_vcdFileName	The name of the VCD file to be used for session.

Value Returned

t_vcdFileName	The VCD file name is the output if the command is successful.
`nil`	Otherwise, `nil` is returned.

Example

```
vcdFile("/tmp/vcdFile.dat")
=> "/tmp/vcdFile.dat"
```

Specifies `/tmp/vcdFile.dat` as the VCD file to be used for current AMS-UltraSim OCEAN session.

vcdInfoFile

```
vcdInfoFile( t_vcdInfoFileName )
     => t_vcdInfoFileName/nil
```

Description

Sets a VCD info file for a given AMS or UltraSim OCEAN session when you have set UltraSim as the solver. You also need to specify a VCD file while using this command. You can specify only one VCD info file for a session. You may use ocnDisplay('vcdInfoFile) to view the currently active VCD info file.

Note: This command is applicable for AMS and UltraSim simulators. For AMS, it works only when UltraSim is the solver.

Arguments

t_vcdInfoFileName	The name of the VCD info file to be included.

Value Returned

t_vcdInfoFileName	The VCD info file name is the output if the command is successful.
`nil`	Otherwise, `nil` is returned.

Example

```
vcdInfoFile("/tmp/vcdInfoFile.dat")
=> "/tmp/vcdInfoFile.dat"
```

Specifies /tmp/vcdInfoFile.dat as the VCD file to be used for current AMS-UltraSim OCEAN session.

vecFile

```
vecFile( t_vecFile [t_vecFile1 … t_vecFileN] )
     => t_vecFile(s)/nil
```

Description

Sets the vector files to be used in an AMS or UltraSim OCEAN session. You use the vecFile command to specify a list of vector files which go to control file. You may use ocnDisplay('vecFile) to view the currently active vector files in an OCEAN session.

Note: This command is applicable for AMS and UltraSim simulators. For AMS, it works only when UltraSim is the solver.

Arguments

t_vecFile — The name of the vector file to be included.

t_vecFile1…t_vecFileN — The names of the additional vector files to be included.

Value Returned

t_vecFile — The names of the vector file(s) are listed if the command is successful.

`nil` — Otherwise, `nil` is returned.

Example

```
vecFile("/tmp/vec.dat" "/tmp/vec2.dat")
=> ("/tmp/vec1.dat" "/tmp/vec2.dat")
```

Specifies /tmp/vec.dat and /tmp/vec.dat2 as the vector files to be used for the current AMS-UltraSim OCEAN session.

hlcheck

```
hlcheck( t_value )
     => t / nil
```

Description

Sets or gets the value of the `hlcheck` option used in the `vec_include` statement in a netlist. You may use the `ocnDisplay('hlcheck)` command to view the current value of `hlcheck` in an OCEAN session associated with vector files.

Note: This command is applicable only when one or more vector files are specified in a given 'spectre' OCEAN session.

Arguments

`t_value`	Value to be set for the `hlcheck` option. Possible values include `"off"`, `"0"`, and `"1"`. The value `"off"` disables the `hlcheck` option in the `vec_include` statement.

Value Returned

`t`	Returns `t` if the `hlcheck` option is set with the value supplied as argument
`nil`	Otherwise, returns `nil` and an error message is displayed

Example

```
hlcheck( "1" )
=> t
```

```
Sets the value of the hlcheck option as 1 in the vec_include statement
```

```
hlcheck()
=> "1"
```

```
Returns the value of the hlcheck option
```

ocnAmsSetOSSNetlister

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

Description

Sets the netlister mode to OSS-based for a given `ams` OCEAN session.

Arguments

None

Value Returned

`t`	Returns `t` if successful
`nil`	Returns `nil` otherwise

Example

```
ocnAmsSetOSSNetlister()
t
```

Sets the netlister mode to OSS-based instead of Cellview-based for the current `ams` simulator session.

7

Data Access Commands

The data access commands let you open results and select different types of results to analyze. You can get the names and values of signals and components in the selected results, and you can print different types of reports.

In this chapter, you can find information on the following data access commands

dataTypes

```
dataTypes()
     => l_dataTypes/nil
```

Description

Returns the list of data types that are used in an analysis previously specified with `selectResult`.

Arguments

None.

Value Returned

l_dataTypes	Returns the list of data types.
`nil`	Returns `nil` and an error message if the list of datatypes cannot be returned.

Example

```
selectResult( 'dcOpInfo )
dataTypes() => ("bjt" "capacitor" "isource" "mos2" "resistor" "vsource")
```

Returns the data types used in the selected file, in this case, `dcOpInfo`.

Example 2:

```
selectResult( 'model )
dataTypes() => ("bjt" "mos2")
```

Returns the data types used in the selected file, in this case, `model`.

deleteSubckt

```
deleteSubckt(
     t_name
     )
     => t | nil
```

Description

Deletes the specified subcircuit instance saved using the `saveSubckt` command.

Arguments

t_name	The name of the subcircuit instance.

Value Returned

`t`	Returns `t` if the selected subcircuit instances is deleted.
`nil`	Returns `nil` if the name of the specified instance is not correct.

Examples

The following example deletes the subcircuit instance `I0`.

```
deleteSubckt("/I0")
=> t
```

displaySubckt

```
displaySubckt(
     t_args
     t_outPort
     )
     => t | nil
```

Description

Prints the subcircuit information to the output file.

Arguments

t_args	The value of this argument should always be `nil`.
t_outPort	The name of the file to save the subcircuit information. If you do not specify the location with the filename, the file is saved in the current working directory.

Value Returned

`t`	Returns `t` if the subcircuit information is printed in the specified output file.
`nil`	Returns `nil` if the name of the output file is not specified, or an error occurred.

Examples

The following example prints the subcircuit information in the `subckts.txt` file:

```
fptr = outfile("/home/krajiv/subckts.txt")
=> port:"/home/krajiv/subckts.txt"

displaySubckt(nil fptr)
=> t

close(fptr)
=> t
```

getData

```
getData( t_name [?result s_resultName [?resultsDir t_resultsDir]] )
     => x_number/o_waveform/nil
```

Description

Returns the number or waveform for the signal name specified.

The type of value returned depends on how the command is used.

Arguments

t_name	Name of the signal.
s_resultName	Results from an analysis. When specified, this argument will only be used internally and will not alter the current result which was set by the selectResult command. The default is the current result selected with the selectResult command.
t_resultsDir	Directory containing the PSF files (results). If you supply this argument, you must also supply the resultName argument. When specified, this argument will only be used internally and will not alter the current results directory which was set by the openResults command. The default is the current results directory set by the openResults command.

Value Returned

x_number	Returns an integer simulation result.
o_waveform	Returns a waveform object. A waveform object represents simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)
`nil`	Returns `nil` and an error message if the value cannot be returned.

Example

```
getData( "/net6" ) => srrWave:25178234
```

Returns the number or waveform for `net6`. In this example, the return value is equivalent to `v( "/net6" )`.

```
getData( "/V1" ?result 'ac )
=> srrWave:96879364
```

Returns the number or waveform for `V1`. In this example, the return value is equivalent to: `i( "/V1" ?result 'ac )`.

```
selectResult( 'tran ) =>
```

```
ocnPrint( getData( "net1" ) ) =>
```

The `getData( "net1" )` command passes a waveform to the `ocnPrint` command. The `ocnPrint` command then prints the data for the waveform. In this example, the return value is equivalent to:
`(v( "net1" ))`.

```
ocnPrint( getData( "net1" ?result 'tran ?resultsDir "./simulation/testcell/
spectre/schematic/psf")
```

Returns a signal on `net1` for the `tran` result stored in the path `"./simulation/testcell/spectre/schematic/psf"`.

Note: To identify the data type of the value returned by the `getData` command, you can use the `type` SKILL function. For scalar values, the type function returns the name of data type. For example, `integer` or `flonum`. For waveforms, it returns `other`.

```
x=getData("/net10")
```

```
type(x)
```

The example given above returns `other`.

```
x=ymax(VT("/net10"))
```

```
type(x)
```

This will return `flonum`.

getResult

```
getResult ( [?result s_resultName [?resultsDir t_resultsDir]] )
     => o_results/nil
```

Description

Gets the data object for a specified analysis without overriding the status of any previously executed `selectResult()` or `openResults()` commands.

Returns the data object for a particular analysis similar to the `selectResult()` function does. Unlike the `selectResult()` function, all subsequent data access commands will not internally use this information.

Arguments

s_resultName	Results from an analysis. When specified, this argument will only be used internally and will not alter the current result which was set by the selectResult command. The default is the current result selected with the selectResult command.
t_resultsDir	Directory containing the PSF files (results). If you supply this argument, you must also supply the resultName argument. When specified, this argument will only be used internally and will not alter the current results directory which was set by the openResults command. The default is the current results directory set by the openResults command.

Value Returned

o_results	Returns the object representing the selected results.
`nil`	Returns `nil` and an error message if there are problems accessing the analysis.

Example

```
getResult( ?result 'tran )
```

i

```
i( t_component [?result s_resultName [?resultsDir t_resultsDir]] )
     => o_waveform/nil
```

Description

Returns the current through the specified component.

Arguments

t_component	Name of the component.
s_resultName	Results from an analysis. When specified, this argument will only be used internally and will not alter the current result which was set by the selectResult command. The default is the current result selected with the selectResult command.
t_resultsDir	Directory containing the PSF files (results). If you supply this argument, you must also supply the resultName argument. When specified, this argument will only be used internally and will not alter the current results directory which was set by the openResults command. The default is the current results directory set by the openResults command.

Value Returned

o_waveform	Returns a waveform object. A waveform object represents simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.).
`nil`	Returns an error message and `nil` if there is a problem.

Example

```
selectResult( 'tran )
i( "/R1" )
```

Returns the current through the `R1` component.

```
ocnPrint( i( "/R1" ) )
```

Prints the current through the `R1` component.

```
ocnPrint( i( "/R1" ?result 'dc ) )
```

Prints the current through the `R1` component with respect to the `dc` swept component.

```
ocnPrint( i( "/R1" ?resultsDir "./test2/psf" ?result 'dc ) )
```

Prints the current through the `R1` component with respect to `dc` for the results from a different run (stored in `test2/psf`).

ocnHelp

```
ocnHelp( [?output t_filename | p_port][s_command] )
     => t / nil
```

Description

Provides online help for the specified command.

If no command is specified, provides information about how to use help and provides the different categories of information contained in the help library. If you provide a filename as the `?output` argument, the `ocnHelp` command opens the file and writes the information to it. If you provide a port (the return value of the SKILL `outfile` command), the `ocnHelp` command appends the information to the file that is represented by the port. If you do not specify `?output`, the output goes to standard out (stdout).

Arguments

t_filename	File in which to write the information. The `ocnHelp` command opens the file, writes to the file, and closes the file. If you specify the filename without a path, the `ocnHelp` command creates the file in the directory pointed to by your Skill Path. To find out what your Skill path is, type `getSkillPath()` at the OCEAN prompt.
p_port	Port (previously opened with `outfile`) through which to append the information to a file. You are responsible for closing the port. See the outfile command for more information.
s_command	Command for which you want help.

Value Returned

`t`	Displays the online help and returns `t`.
`nil`	Returns `nil` and an error message if help cannot be displayed.

Example

```
ocnHelp()
=> t
```

Displays information about using online help.

```
ocnHelp( 'analysis )
=> t
```

Displays help for the `analysis` command.

```
ocnHelp( ?output "helpInfo" )
=> t
```

Writes information about using online help to a file named `helpInfo`.

```
simulator('spectre)
ocnHelp('envOptions)
```

Displays a list of environment options that can be set for a simulator. First, set the simulator and then run the ocnHelp command .

ocnResetResults

```
ocnResetResults()
     => t
```

Description

Unsets the results opened by the `openResults` command. Use this command to return to the state that existed prior to using the `openResults` command.

Arguments

None.

Value Returned

`t`	Resets the results and returns `t`.

Example

`getResult( ?result 'tran )`	Returns `nil` when no results have been opened.
`openResults( "./psf" )`	Makes `getResult` return valid object.
`ocnResetResults()`	Resets the results opened by `openResults` and makes `getResult` return `nil`.

openResults

```
openResults( s_jobName | t_dirName [g_enableCalcExpressions] )
    => t_dirName/nil
```

Description

Opens simulation results stored in PSF files or opens the results from a specified job, depending on which parameter is called.

When `openResults` passes a symbol, it interprets the value as a job name and opens the results for the specified job. *`s_jobName`* is a job name and is defined when a `run` command is issued.

When `openResults` passes a text string, it opens simulation results stored in PSF files in the specified directory. The results must have been created by a previous simulation run through OCEAN or the Virtuoso® Analog Design Environment. The directory must contain a file called `logFile` and might contain a file called `runObjFile`. When you perform tasks in the design environment, the `runObjFile` is created. Otherwise, only `logFile` is created.

If you want to find out which results are currently open, you can use `openResults` with no argument. The directory for the results that are currently open is returned.

Note: If you run a successful simulation with distributed processing disabled, the results are automatically opened for you. Also, a job name is generated by every analysis, even if distributed processing is not enabled.

Arguments

`s_jobName`	The name of a distributed process job. *`s_jobName`* is a job name and is defined when a `run` command is issued.
`t_dirName`	The directory containing the PSF files.
`g_enableCalcExpressions`	An optional argument, which when set to `t`, allows the evaluation of Calculator expressions. For this argument to work, the directory mentioned in *`t_dirName`* must be a psf directory and must contain `runObjFile`. The default value for this argument is `t`.

Value Returned

t_dirName	The directory containing the PSF files.
nil	Returns nil and an error message if there are problems opening the results.

Example

```
openResults( "./simulation/opamp/spectre/schematic/psf" )
=> "./simulation/opamp/spectre/schematic/psf"
```

Opens the results in the psf directory within the specified path.

```
openResults( "./psf" )
=> psf
```

Opens the results in the psf directory in the current working directory.

```
openResults( "./psf" t )
=> psf
```

Opens the results in the psf directory in the current working directory. It also allows the evaluation of the Calculator expression.

outputParams

```
outputParams( t_compType [?result s_resultName [?resultsDir t_resultsDir]] )
    => l_outputParams/nil
```

Description

Returns the list of output parameters for the specified component.

You can use the dataTypes command to get the list of components for a particular set of results.

Note: You can use any of the parameters in *outputParams* as the second argument to the pv command.

Arguments

t_compType	Name of a component.
s_resultName	Results from an analysis. When specified, this argument will only be used internally and will not alter the current result which was set by the selectResult command. The default is the current result selected with the selectResult command.
t_resultsDir	Directory containing the PSF files (results). If you supply this argument, you must also supply the resultName argument. When specified, this argument will only be used internally and will not alter the current results directory which was set by the openResults command. The default is the current results directory set by the openResults command.

Value Returned

l_outputParams	Returns the list of parameters.
nil	Returns nil and an error message if there are no associated parameters or if the specified component (compType) does not exist.

Example

```
selectResult( 'dcOpInfo )
dataTypes() => ("bjt" "capacitor" "isource" "mos2" "resistor" "vsource")
outputParams( "bjt" )
```

Selects the `dcOpInfo` results, returns the list of components for these results, and returns the list of output parameters for the `bjt` component.

```
outputParams("bjt" ?result 'dcOpInfo ?resultsDir "/VADE615/simulation/ampTest/
spectre/schematic/psf")
```

Returns a list of output parameters for the bjt component for dcOpInfo (dc analysis with save dc operating point) results stored at the location `./psf`.

outputs

```
outputs( [?result s_resultName [?resultsDir t_resultsDir]]
     [?type t_signalType])
     => l_outputs/nil
```

Description

Returns the names of the outputs whose results are stored for an analysis. You can plot these outputs or use them in calculations.

Arguments

s_resultName	Results from an analysis. When specified, this argument will only be used internally and will not alter the current result which was set by the selectResult command. The default is the current result selected with the selectResult command.
t_resultsDir	Directory containing the PSF files (results). If you supply this argument, you must also supply the resultName argument. When specified, this argument will only be used internally and will not alter the current results directory which was set by the openResults command. The default is the current results directory set by the openResults command.
t_signalType	Data type of the signal.

Value Returned

l_outputs	Returns the list of outputs.
`nil`	Returns `nil` and an error message if there are problems returning the names of the stored outputs.

Example

```
outputs()
=> ( "net13" "net16" "net18" )
```

Returns the names of the outputs for the PSF file selected with `selectResult`.

```
outputs( ?type "V" )
```

Returns all the signal names that are node voltages. The dataType (signal) returns the data type of the signal.

```
outputs(?result "tran" ?resultsDir "./psf")
=> ( "net11" "net15" "net17")
```

Returns the names of the outputs for the tran results stored at the location `./psf`.

phaseNoise

```
phaseNoise( g_harmonic S_signalResultName [?result s_noiseResultName
     [?resultsDir t_resultsDir]] )
     => o_waveform/nil
```

Description

Returns the phase noise waveform which is calculated using information from two PSF data files.

This command should be run on the results of the Spectre pss-pnoise analysis.

Arguments

g_harmonic	List of harmonic frequencies.
S_signalResultName	Name of the result that stores the signal waveform. Use the `results()` command to obtain the list results.
s_noiseResultName	Name of the result that stores the "positive output signal" and "negative output signal" noise waveforms. When specified, this argument will only be used internally and will not alter the current result which was set by the selectResult command. The default is the current result selected with the selectResult command.
t_resultsDir	Directory containing the PSF files (results). If you supply this argument, you must also supply the S_noiseResultName argument. Both the S_signalResultName and S_noiseResultName arguments are read from this directory. When specified, this argument will only be used internally and will not alter the current results directory which was set by the openResults command. The default is the current results directory set by the openResults command.

Value Returned

o_waveform	Waveform representing the phase noise.
`nil`	Returns `nil` if there is an error.

Example

```
plot(phaseNoise(0 "pss-fd.pss"))
phaseNoise(1 "pss_fd" ?result "pnoise" ?resultsDir “./PSF")
```

pv

```
pv( t_name t_param [?result s_resultName [?resultsDir t_resultsDir]] )
    => g_value/nil
```

Description

Returns the value of the specified component parameter. You can use the outputParams command to get the list of parameters for a particular component.

Arguments

t_name	Name of the node or component.
t_param	Name of the parameter.
s_resultName	Results from an analysis. When specified, this argument will only be used internally and will not alter the current result that was set using the `selectResult` command. The default is the current result selected using the `selectResult` command. **Note:** To get the correct value of the variables while running parametric analysis, use the `designParamVals` value for the `resultName` argument.
t_resultsDir	Directory containing the PSF files (results). If you supply this argument, you must also supply the `resultName` argument. When specified, this argument will only be used internally and will not alter the current results directory that was set using the `openResults` command. The default is the current results directory set using the `openResults` command.

Value Returned

g_value	Returns the requested parameter value.
`nil`	Returns `nil` and prints an error message.

Example

```
selectResult( 'dcOpInfo )
pv( "/I0/M1" "vds" )
```

Returns the value of the `vds` parameter for the `I0/M1` component.

```
pv( "/I0/M1" "vds" ?resultsDir "/VADE/simulation/ampTest/spectre/schematic/test2/
psf" )
```

Returns the value of the `vds` parameter for the `I0/M1` component. These values are read from the results directory, `/VADE/simulation/ampTest/spectre/schematic/test2/psf`.

```
pv( "/I0/M1" "vds" ?result "dcOpInfo" ?resultDir "/VADE/simulation/ampTest/
spectre/schematic/test1/psf")
```

Returns the value of the `vds` parameter for the `I0/M1` component. These values are read from the `dcOpInfo` results saved in the results directory, `/VADE/simulation/ampTest/spectre/schematic/test1/psf`.

```
pv("top-level" "CAP" ?result "designParamVals")
```

Returns the value of the `CAP` variable for the top-level hierarchy in the design. These values are read from the default results directory.

resultParam

```
resultParam( S_propertyName [?result s_resultName [?resultsDir t_resultsDir]] )
    => L_value/nil
```

Description

Returns the value of a header property from the selected result data.

Arguments

s_propertyName
Name of the parameter

s_resultName Results from an analysis. When specified, this argument will only be used internally and will not alter the current result which was set by the selectResult command. The default is the current result selected with the selectResult command.

t_resultsDir Results from an analysis. When specified, this argument will only be used internally and will not alter the current result which was set by the selectResult command. The default is the current result selected with the selectResult command.

Value Returned

L_value Value of the parameter. The data type depends on the data type of the parameter.

nil Returns `nil` and an error message if there are problems returning the results.

Example

```
resultParam("positive output signal" ?result "pnoise.pss")
=> "pif"
resultParam("negative output signal" ?result "pnoise.pss")
=> "0"
```

Returns the name of the positive and negative output signals from PSS-noise analysis result. In this case, the data type of the returned value is a string.

```
resultParam("port1.r.value" ?result "sp")
```

```
=> 40.0
resultParam("port2.r.value" ?result "sp")
=> 40.0
```

Returns the reference impedance of the ports in a two-port network from the S-parameter analysis result. In this case, the data type of the returned value is a floating point number.

```
resultParam("positive output signal" ?result "pnoise.pss" ?resultsDir "./psf")
=> "0"
```

Returns the names of the positive output signals from the PSS-noise analysis results stored at the location `./psf`.

results

```
results( [ ?resultsDir t_resultsDir ] )
     => l_results/nil
```

Description

Returns a list of the type of results that can be selected.

Arguments

t_resultsDir	Directory containing the PSF files (results). When specified, this argument will only be used internally and will not alter the current results directory which was set by the openResults command. The default is the current results directory set by the openResults command.

Value Returned

l_results	Returns the list of result types.
`nil`	Returns `nil` and an error message if there are problems returning the results.

Example

```
results()
=> ( dc tran ac )
```

Returns the list of results available.

```
results( ?resultsDir "./psf" )
```

Returns a list of results stored at the location `./psf`.

saveSubckt

```
saveSubckt(
     t_name
     [?voltage g_voltage]
     [?current g_current]
     [?power g_power]
     [?vDepth s_vDepth]
     [?iDepth s_iDepth]
     [?pwrDepth s_pwrDepth]
     [?compress g_compress]
     [?filterRC g_filterRC]
     [?ports g_ports]
     [?userOptions g_userOptions]
     )
     => t | nil
```

Description

Saves and modifies the specified subcircuit instances and signals.

Arguments

t_name	The name of the subcircuit instance.
g_voltage	Specifies whether you want to save the voltage for the subcircuit.
g_current	Specifies whether you want to save the current for the subcircuit.
g_power	Specifies whether you want to save the power signals for the subcircuit.
s_vDepth	The hierarchy level to which you want to save the voltage signal for the subcircuit. If not specified, voltage for all the levels of hierarchy are saved.
s_iDepth	The hierarchy level to which you want to save the current signal for the subcircuit. If not specified, current for all the levels of hierarchy are saved.
s_pwrDepth	The hierarchy level to which you want to save the power signal for the subcircuit. If not specified, power signals for all the levels of hierarchy are saved.

g_compress	Specifies whether you want to reduce the size of the output file. When enabled, the spectre simulator saves the data for a signal only when the value of that signal changes.
g_filterRC	Specifies whether to filter out the nodes that are connected only to parasitic elements from the output signal list.
g_ports	Specifies whether to save the output port information for the specified subcircuit.
g_userOptions	Specify the other save options that you want to define for the signal.

Value Returned

`t`	Returns `t` if the subcircuit instance is saved.
`nil`	Returns `nil` if the name of the specified instance is not correct.

Examples

Example 1

The following example saves the voltage for five levels and current for two levels of hierarchy for the subcircuit `I0`.

```
saveSubckt("/I0" ?voltage t ?current t ?vDepth "5" ?iDepth "2")
=> t
```

Example 2

The following example saves the voltage for two levels and power signals for one level of hierarchy for the subcircuit `I1`.

```
saveSubckt("/I1" ?voltage t ?power t ?vDepth "2" ?pwrDepth "1")
=> t
```

Example 3

The following example saves the voltage for two levels and power signals for one level of hierarchy for the subcircuit `I1`, along with the port information. The output signals are compressed.

```
saveSubckt("/I1" ?voltage t ?power t ?vDepth "2" ?pwrDepth "1" ?port t ?compress t)
=> t
```

selectResult

```
selectResult( S_resultsName [n_sweepValue])
     => o_results/nil
```

Description

Selects the results from a particular analysis whose data you want to examine.

The argument that you supply to this command is a data type representing the particular type of analysis results you want. All subsequent data access commands use the information specified with `selectResult`.

Note: Refer to the results command to get the list of analysis results that you can select.

Arguments

s_resultsName	Results from an analysis.
n_sweepValue	The sweep value you wish to select for an analysis.

Value Returned

o_results	Returns the object representing the selected results.
nil	Returns `nil` and an error message if there are problems selecting the analysis.

Example

```
selectResult( 'tran )
```

Selects the results for a transient analysis.

```
sweepValues(3.0 3.333333 3.666667 4.0 4.333333 4.666667 5.0 )
selectResult("tran" "3.333333")
```

The `sweepValues` command prints a list of sweep values.

The `selectResult` command selects a specific value for a transient analysis.

```
selectResult( 'tran )
```

Selects the results for a transient analysis.

```
paramAnalysis("supply" ?start 3 ?stop 5 ?step 1.0/3)
```

```
paramRun("supply")
selectResult(( 'tran car( sweepValues() )
```

Selects the data corresponding to the first parametric run.

Note: `selectResult('tran)` would select the entire family of parametric data.

sp

```
sp( x_iIndex x_jIndex [?result s_resultName [?resultsDir t_resultsDir]] )
    => o_waveform/nil
```

Description

Returns S-parameters for N port networks.

This command should be run on the results of the Spectre sp (S-parameter) analysis.

Arguments

x_iIndex	The *i*th index of the coefficient in the scattering matrix.
x_jIndex	The *j*th index of the coefficient in the scattering matrix.
s_resultName	Results from an analysis. When specified, this argument will only be used internally and will not alter the current result which was set by the selectResult command. The default is the current result selected with the selectResult command.
t_resultsDir	Directory containing the PSF files (results). If you supply this argument, you must also supply the resultName argument. When specified, this argument will only be used internally and will not alter the current results directory which was set by the openResults command. The default is the current results directory set by the openResults command.

Value Returned

o_waveform	Waveform object representing the S-parameter.
`nil`	Returns `nil` if there is an error.

Example

```
s21 = sp(2 1)
s12 = sp(1 2)
plot(s21 s12)

s11 = sp(1 1 ?result "sp" ?resultsDir "./simResult/psf")
```

Returns the S-parameter `s11` for results of S-parameter(sp) analysis stored at the location `./simResult/psf`.

sweepNames

```
sweepNames( [o_waveForm] [?result s_resultName [?resultsDir t_resultsDir]] )
     => l_sweepName/nil
```

Description

Returns the names of all the sweep variables for either a supplied waveform, a currently selected result (via selectResult()) or a specified result.

Arguments

o_waveForm	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*`XXXXX`*). When this argument is used, the t_resultsDir and s_resultName arguments are ignored.
s_resultName	Results from an analysis. When specified, this argument will only be used internally and will not alter the current result which was set by the selectResult command. The default is the current result selected with the selectResult command.
t_resultsDir	Directory containing the PSF files (results). If you supply this argument, you must also supply the resultName argument. When specified, this argument will only be used internally and will not alter the current results directory which was set by the openResults command. The default is the current results directory set by the openResults command.

Value Returned

l_sweepName	Returns a list of the sweep names.
`nil`	Returns `nil` and prints an error message if the sweep names cannot be returned.

Example

```
selectResult('tran)
```

```
sweepNames()
=> ( "TEMPDC" "time" )
```

Returns a list of sweep variables for the selected results. In this case, the return values indicate that the data was swept over temperature and time.

```
sweepNames(?result 'ac)
=> ("TEMPDC" "freq")
```

```
sweepNames()
=> ("TEMPDC" "time")
```

```
w = VT("/vout")
sweepNames( w )
=> ( "r" "time")
```

Returns the sweep variables for the waveform `w`.

```
sweepNames(?result 'ac ?resultsDir "./test/psf")
=> ("TEMPDC" "freq")
```

Returns the sweep variables for the results of the ac analysis stored at the location `./test/psf`.

sweepValues

```
sweepValues( [o_waveForm] )
     => l_sweepValues/nil
```

Description

Returns the list of sweep values of the outermost sweep variable of either the selected results or the supplied waveform. This command is particularly useful for parametric analyses.

Arguments

o_waveForm	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)

Value Returned

l_sweepValues	Returns the list of sweep values.
`nil`	Returns `nil` and an error message if the list of sweep values cannot be returned.

Example

```
sweepValues()
=> ( -50 -15 20 55 90.0 )
```

Returns a list of sweep values for the selected results. In this case, the return values indicate the temperature over which the data was swept.

```
w = VT("/vout")

sweepNames( w )
=> ( "r" "time" )

sweepValues( w )
=> ( 2000 4000 6000 )
```

Returns a list of sweep values for the wave `w`. In this case, the return values indicate the resistance over which the data was swept.

sweepVarValues

```
sweepVarValues( [t_varName] [?result s_resultName [?resultsDir t_resultsDir]]
     => l_sweepName/nil
```

Description

Returns the list of sweep values for a particular swept variable name. This command is particularly useful for parametric analyses.

Arguments

t_varName	Name of the specific variable from which the values are retrieved.
s_resultName	Results from an analysis. When specified, this argument will only be used internally and will not alter the current result which was set by the selectResult command. The default is the current result selected with the selectResult command.
t_resultsDir	Directory containing the PSF files (results). If you supply this argument, you must also supply the resultName argument. When specified, this argument will only be used internally and will not alter the current results directory which was set by the openResults command. The default is the current results directory set by the openResults command.

Value Returned

l_sweepValues	Returns the list of sweep values.
nil	Returns nil and an error message if the list of sweep values cannot be returned.

Example

```
selectResult('tran)

sweepNames()
=> ("TEMPDC" "Vsupply" "time")

sweepVarValues("TEMPDC")
=> (0 32)
```

```
sweepNames(?result 'ac)
=> ("TEMPDC" "Vsupply" "freq")

sweepVarValues("Vsupply" ?result 'ac)
=> (5 12 15)

sweepNames(?result 'ac ?resultsDir "./simResult/psf")
=> ("TEMPDC" "freq")

sweepVarValues("TEMPDC" ?result 'ac ?resultsDir "./simResult/psf")
=> (-15 20 55)
```

v

```
v( t_net [?result s_resultName [?resultsDir t_resultsDir]] )
    => o_waveform/nil
```

Description

Returns the voltage of the specified net.

Arguments

t_net	Name of the net.
s_resultName	Results from an analysis. When specified, this argument will only be used internally and will not alter the current result which was set by the selectResult command. The default is the current result selected with the selectResult command.
t_resultsDir	Directory containing the PSF files (results). If you supply this argument, you must also supply the resultName argument. When specified, this argument will only be used internally and will not alter the current results directory which was set by the openResults command. The default is the current results directory set by the openResults command.

Value Returned

o_waveform	Returns a waveform object. A waveform object represents simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.).
`nil`	Returns an error message and `nil` if there is a problem.

Example

```
selectResult('tran)
v( "/net56" )
```

Returns the voltage for `net56`.

```
ocnPrint( v( "/net56" ) )
```

Prints tabular information representing the voltage for `net56`.

```
ocnPrint( v( "net5" ?result 'dc ) )
```

Prints the voltage of `net5` with respect to the `dc` swept component.

```
ocnPrint( v( "net5" ?resultsDir "./test2/psf" ?result 'dc ) )
```

Prints the voltage of `net5` with respect to `dc` for the results from a different run (stored in `test2/psf`).

vswr

```
vswr( x_index [?result s_resultName [?resultsDir t_resultsDir]] )
     => o_waveform/nil
```

Description

Computes the voltage standing wave ratio.

This function is a higher level wrapper for the OCEAN expression

```
(1 + mag( s( x_index x_index ))) / (1 - mag( s( x_index x_index )))
```

This command should be run on the results of the Spectre sp (S-parameter) analysis.

Arguments

x_index	Index of the port.
s_resultName	Results from an analysis. When specified, this argument will only be used internally and will not alter the current result which was set by the selectResult command. The default is the current result selected with the selectResult command.
t_resultsDir	Directory containing the PSF files (results). If you supply this argument, you must also supply the resultName argument. When specified, this argument will only be used internally and will not alter the current results directory which was set by the openResults command. The default is the current results directory set by the openResults command.

Value Returned

o_waveform	Waveform object representing the voltage standing wave ratio.
`nil`	Returns an error message or `nil` if there is a problem.

Example

```
plot( vswr(2) )
vswr1 = vswr(1 ?result "sp" ?resultsDir "./simResult/psf")
```

Returns the voltage standing wave ratio value at port `1` for the results of S-parameter(sp) analysis stored at the location `./simResult/psf`.

zm

```
zm( x_index [?result s_resultName [?resultsDir t_resultsDir]] )
     => o_waveform/nil
```

Description

Computes the port input impedance.

The `zm` function is computed in terms of the S-parameters and the reference impedance. This function is a higher level wrapper for the OCEAN expression

```
(1 + s( x_index x_index )) / (1 - s( x_index x_index ))
    * or( zref( x_index ) 50)
```

This command should be run on the results of the Spectre sp (S-parameter) analysis.

Arguments

x_index	Index of the port.
s_resultName	Results from an analysis. When specified, this argument will only be used internally and will not alter the current result which was set by the selectResult command. The default is the current result selected with the selectResult command.
t_resultsDir	Directory containing the PSF files (results). If you supply this argument, you must also supply the resultName argument. When specified, this argument will only be used internally and will not alter the current results directory which was set by the openResults command. The default is the current results directory set by the openResults command.

Value Returned

o_waveform	Waveform object representing the port input impedance.
`nil`	Returns an error message and `nil` if there is a problem.

Example

```
plot(zm(2))
zm1 = zm(1 ?result "sp" ?resultsDir "./simResult/psf")
```

Returns input impedance at port `1` for results of S-parameter (sp) analysis stored at the location `./simResult/psf`.

zref

```
zref( x_portIndex [?result s_resultName [?resultsDir t_resultsDir]] )
     => f_impedance/nil
```

Description

Returns the reference impedance for an N-port network.

This command should be run on the results of the Spectre sp (S-parameter) analysis.

Arguments

x_portIndex	Index of the port.
s_resultName	Results from an analysis. When specified, this argument will only be used internally and will not alter the current result which was set by the selectResult command. The default is the current result selected with the selectResult command.
t_resultsDir	Directory containing the PSF files (results). If you supply this argument, you must also supply the resultName argument. When specified, this argument will only be used internally and will not alter the current results directory which was set by the openResults command. The default is the current results directory set by the openResults command.

Value Returned

f_impedance	Reference impedance.
`nil`	Returns an error message and `nil` if there is a problem.

Example

```
Zref = zref(2)
zref1 = zref(1 ?result "sp" ?resultsDir "./simResult/psf")
```

Returns the reference impedance at port `1` for the results of S-parameter(sp) analysis stored at the location `./simResult/psf`.

8

Plotting and Printing Commands

This chapter contains information on the following plotting and printing commands:

This chapter also includes a topic, Plotting and Printing SpectreRF Functions in OCEAN on page 275.

addSubwindow

```
addSubwindow()
     => x_subwindowID/nil
```

Description

Adds a subwindow to the current Waveform window and returns the number for the new subwindow, which is found in the upper right corner.

Arguments

None.

Value Returned

x_subwindowID	Returns the window ID of the new subwindow.
`nil`	Returns `nil` and an error message if there is no current Waveform window.

Example

```
addSubwindow()
=>3
```

Adds a new subwindow to the Waveform window.

addSubwindowTitle

```
addSubwindowTitle( x_windowtitle)
     => t / nil
```

Description

Adds a title to the current subwindow in the active window. The current subwindow is defined using the `currentSubwindow` command.

Arguments

`x_windowtitle`	User-defined title for the subwindow.

Value Returned

`t`	The user-supplied name of the current subwindow.
`nil`	Returns `nil` if the title is not created.

Example

```
addSubwindowTitle( "waveform 2")
=> t
```

Adds the title `waveform 2` to the selected subwindow.

addTitle

```
addTitle( x_windowtitle)
     => t / nil
```

Description

Adds a title to the current active OCEAN window. The current window is defined using the `currentWindow` command.

Arguments

x_windowtitle

User-defined title for the window.

Value Returned

t	The user-supplied name of the current window.
nil	Returns `nil` if the title is not created.

Example

```
addTitle( "waveform 1" )
=> t
```

Adds the title `waveform 1` to the selected window.

addWaveLabel

```
addWaveLabel( x_waveIndex l_location t_label [?textOffset g_textOffset]
     [?color x_color] [?justify t_justify] [?fontStyle t_fontStyle]
     [?height x_height] [?orient t_orient] [?drafting g_drafting]
     [?overBar g_overbar])
     => s_labelId/nil
```

Description

Attaches a label to the specified waveform curve in the current subwindow.

Arguments

x_waveIndex	Integer identifying the waveform curve.
l_location	List of two waveform coordinates that describe the location for the label.
t_label	Label for the waveform.
g_textOffset	Boolean that specifies whether to place a marker or label. If set to `t`, a marker is placed. If set to `nil`, a label is placed. Default value: `nil`.
x_color	Label color specified as an index in the technology file. Default value: `10`
t_justify	Justification, which is specified as `"upperLeft"`, `"centerLeft"`, `"lowerLeft"`, `"upperCenter"`, `"centerCenter"`, `"lowerCenter"`, `"upperRight"`, `"centerRight"`, or `"lowerRight"`. Default value: `"lowerLeft"`
t_fontStyle	Font style, which is specified as `"euroStyle"`, `"gothic"`, `"math"`, `"roman"`, `"script"`, `"stick"`, `"fixed"`, `"swedish"`, `"raster"`, or `"milSpec"`. Default value: the font style of the current subwindow
x_height	Height of the font. Default value: the font height of the current subwindow

t_orient	Orientation of the text, specified as either `"R0"` or `"R90"`. Default value: `"R0"`
g_drafting	Boolean that specifies whether the label stays backwards or upside-down. If set to `t`, a backwards or upside-down label is displayed in a readable form. If set to `nil`, a backwards or upside-down label stays the way it is. Default value: `t`
g_overbar	Boolean that specifies whether underscores in labels are displayed as overbars. If set to `t`, underscores in labels are displayed as overbars. If set to `nil`, underbars are displayed as underbars. Default value: `nil`

Value Returned

s_labelId	Returns an identification number for the waveform label.
`nil`	Returns `nil` if there is an error.

Examples

```
addWaveLabel( 1 list( 0 0.5 ) "R5 = " )
```

Attaches the `"R5 = "` label to the specified coordinates on waveform curve `1`.

```
addWaveLabel( 2 list( 0 0.5 )  "R_6 = " ?textOffset  0:20 ?justify "lowerCenter"
?fontStyle "roman" ?height 10 ?orient "R20" ?drafting t ?overbar t)
```

Attaches the label `"R6 = "` to the specified coordinates on waveform curve. The label specifications are as follows: Justification – `lowerCenter`, Font Style – `roman`, Font Height – `10`, and Orientation – `R20`.

The label will be displayed in a readable form. The underscore in the label will be displayed as an overbar.

Additional Information

Note the following points:

- The valid label location ranges between absolute co-ordinates (0, 0) on X-axis and (1,1) on Y-axis (upper and lower bound inclusive).

- The valid marker location ranges between data co-ordinates defined by X-axis and Y-axis limits (upper and lower bound inclusive).

Case1:

```
addWaveLabel(1 list( -0.5 -0.5 ) "Label 1" ?textOffset nil)
```

The following error message appears when the specified label location (`-0.5 0.5)` is outside of the defined boundary limits of label.

```
The location specified for placing a label on the graph is invalid. Specify a valid
label location that ranges between absolute coordinates (0,0) on X-axis and (1,1)
on Y-axis (upper and lower bounds inclusive).
```

Case 2:

```
addWaveLabel(1 list( 80MHz -0.5) "Marker 1" ?textOffset t)
```

The following error message appears when the specified marker location (`80MHz -0.5`) is outside of the X- and Y-axis limits of the graph to be plotted.

```
The location specified for placing a marker on the graph is invalid. Specify a valid
marker location that ranges between data coordinates '(0,-1)' on X-axis and
'(10000,1)' on Y-axis (upper and lower bounds inclusive).
```

addWindowLabel

```
addWindowLabel( l_location t_label )
     => s_labelId/nil
```

Description

Displays a label in the current subwindow. The location for the label is specified with a list of two numbers between 0 and 1.

Arguments

l_location	List of two waveform coordinates that describe the location for the label. Valid values: `0` through `1`
t_label	Label for the waveform.

Value Returned

s_labelId	Returns an identification number for the subwindow label.
`nil`	Returns `nil` if there is an error.

Example

```
label = addWindowLabel( list( 0.75 0.75 ) "test" )
```

Adds the `test` label to the current subwindow at the specified coordinates and stores the label identification number in `label`.

clearAll

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

Description

Erases the contents of the current Waveform window and deletes the waveforms, title, date stamp, and labels stored in internal memory.

Arguments

None.

Value Returned

`t`	Returns `t` if the waveform information is deleted.
`nil`	Returns `nil` and an error message if there is no current Waveform window.

Example

```
clearAll()
=> t
```

Erases the contents of the current Waveform window.

clearSubwindow

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

Description

Erases the contents of the current subwindow.

Arguments

None.

Value Returned

`t`	Returns `t` if the contents of the subwindow are erased.
`nil`	Returns `nil` and an error message otherwise.

Example

```
clearSubwindow()
=> t
```

Erases the contents of the current subwindow.

currentSubwindow

```
currentSubwindow( ?x_subwindow x_subwindow)
     => t / nil
```

Description

Sets *x_subwindow* as the current subwindow.

Arguments

x_subwindow	(Optional) Number of the subwindow, found in the upper right corner, that is to become the current subwindow.

Value Returned

`t`	Returns `t` when the subwindow is set to *x_subwindow*. If you do not specify any argument in this function, it returns the current subwindow number.
`nil`	If no subwindow exists.

Example

```
currentSubwindow( 2 )
```

Sets subwindow 2 as the current subwindow.

currentWindow

```
currentWindow( w_windowId )
     => w_windowId/nil
```

Description

Specifies *w_windowId* as the current Waveform window.

Arguments

w_windowId	Waveform window ID.

Value Returned

w_windowId	Returns the current Waveform window ID.
nil	Returns nil and an error if the current window cannot be set.

Example

```
currentWindow( window(2) )
```

This example specifies window 2 as the current Waveform window.

```
currentWindow()
```

This example returns the current waveform window. For example, if the current waveform window is 4, this command returns the following:

```
window:4
```

dbCompressionPlot

```
dbCompressionPlot(o_wave x_harmonic x_extrapolationPoint
     [?compression x_compression] )
     => t / nil
```

Description

Plots the *n*th compression point plot. The *x_compression* argument is optional and defaults to 1 for 1dB compression, if omitted.

This command should be run on the results of the Spectre swept pss analysis.

Arguments

o_wave — The waveform for which to plot the compression.

x_harmonic — Harmonic frequency index.

x_extrapolationPoint — The extrapolation point.

x_compression — The amount of dB compression.
Default value: `1`

Value Returned

`t` — Returns `t` if the point is plotted

`nil` — returns `nil` if there was an error

Example

```
dbCompressionPlot(v("/Pif") 2 -25)
```

Plots a `1 dB` compression point plot for the waveform `v("/Pif")`.

```
dbCompressionPlot(v("/Pif") 2 -25 ?compression 3)
```

Plots a `3 dB` compression point plot for the waveform `v("/Pif")`.

dcmatchSummary

```
dcmatchSummary([?resultsDir t_resultsDir] [?result S_resultName]
     [?output t_fileName | p_port] [?paramValues ln_paramValues]
     [?deviceType ls_deviceType] [?variations ls_variations]
     [?includeInst lt_includeInst] [?excludeInst lt_excludeInst]
     [?truncateData n_truncateData] [?truncateType s_truncateType]
     [?sortType ls_sortType])
     => t_fileName/p_port/nil
```

Description

Prints a report showing the mismatch contribution of each component in a circuit. If you specify a directory with resultsDir, it is equivalent to temporarily using the `openResults` command. The `dcmatchSummary` command prints the results for that directory and resets the `openResults` command to its previous setting. If you specify a particular result with `resultName`, it is equivalent to temporarily using the `selectResult` command on the specified results. The `dcmatchSummary` command prints the results and resets the `selectResult` command to its previous setting.

This command should be run on the results of the Spectre dcmatch analysis.

Arguments

t_resultsDir	The directory containing the dcmatch-analysis results.
S_resultName	Results from an analysis for which you want to print the dcmatchSummary report.
t_fileName	File in which to write the information. The dcmatchSummary command opens the file, writes to the file and closes the file. If you specify the filename without a path, the dcmatchSummary command creates the file in the directory pointed to by your Skill Path. To find out what your Skill path is, `type getSkillPath()` at the OCEAN prompt.
p_port	Port (previously opened with `outfile`) through which to append the information to a file. You are responsible for closing the port. See the outfile command for more information.
ln_paramValues	List of values for swept parameters at which the dcmatchSummary is to be printed. In case there is just one swept parameter the value can be specified as is.

ls_deviceType	List of device type strings to be included. Valid values are a list of strings or `'all` or a single device name. Default value is `'all`.
ls_variations	An association list containing the device name and the associated variations to print. You can also specify the value ‘all to print all available variations for a device. Default value is `'all`. For Example: `'( ("bsim3v3" ("sigmaOut" "sigmaVth")) ("resistor" ("sigmaOut"))`
lt_includeInst	List of instance name strings to definitely include in the dcmatchSummary.
lt_excludeInst	List of instance name strings to exclude in the dcmatchSummary.
x_truncateData	Specifies a number that the truncateType argument uses to define the components for which information is to be printed.
s_truncateType	Specifies the method that is used to limit the data being included in the report

Valid Values	**Description**	**Sample Values for truncateData**
’top	Saves information for the number of components specified with truncateData. The components with the highest contributions are saved.	10
’relative	Saves information for all components that have a higher contribution than truncateData * maximum. Where maximum is the maximum contribution among all the devices of a given type	1.9n
‘absolute	Saves information for all the components in the selected set whose contribution are more than truncateData.	0.1

‘none	Saves information for all the components.	Not required

ls_sortType Specifies how the printed results are to be sorted. The valid values are nil, ’name, ’output.

Value Returned

t_fileName Returns the name of the port.

p_port Returns the name of the file.

`nil` Returns `nil` and an error message if the summary cannot be printed.

Example

```
dcmatchSummary( ?result ’dcmatch-mine )
```

Prints a report for non-swept DC-Mismatch analysis.

```
dcmatchSummary( ?resultsDir "/usr/simulation/lowpass/spectre/schematic" ?result
’dcmatch)
```

Prints a report for non-swept DC-Mismatch analysis for the results from a different run (stored in the schematic directory).

```
dcmatchSummary( ?resultsDir "/usr/simulation/lowpass/spectre/schematic" ?result
’dcmatch ?paramValues ‘(25) )
```

Prints a report for swept DC-Mismatch analysis at swept parameter value of 25.

```
dcmatchSummary( ?result dcmatch-mine ?output "./summary.out")
```

Prints a report for non-swept DC-Mismatch analysis in the output file `summary.out`.

```
dcmatchSummary( ?paramValues 25 ?deviceType "bsim3v3" ?variations ’(("bsim3v3"
("sigmaOut "sigmaVth" )))
```

Prints a report for swept DC-Mismatch analysis at swept parameter value of 25 for bsim3v3 deviceType and sigmaOut and sigmaVth variations.

```
dcmatchSummary( ?paramValues 25 ?truncateType ’top ?truncateData 1)
```

Prints a report for swept DC-Mismatch analysis at swept parameter value of 25 printing only the component having the highest contribution.

```
dcmatchSummary( ?paramValues 25 ?sortType ’name )
```

Prints a report for swept DC-Mismatch analysis at swept parameter value of 25 sorted on name.

deleteSubwindow

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

Description

Deletes the current subwindow from the current Waveform window.

Arguments

None.

Value Returned

`t`	Returns `t` if the current subwindow is deleted.
`nil`	Returns `nil` and an error message if there is no current subwindow.

Example

```
deleteSubwindow()
=> t
```

Deletes the current subwindow from the Waveform window.

deleteWaveform

```
deleteWaveform( {x_index | all_string } )
     => t / nil
```

Description

Deletes the specified waveform curve or all the waveform curves from the current subwindow of a Waveform window.

Arguments

x_index	Integer identifying a particular waveform curve.
all_string	The string `"all"` specifying that all waveform curves are to be deleted.

Value Returned

t	Returns `t` if the curves are deleted.
nil	Returns `nil` and an error message if the curves are not deleted.

Example

```
deleteWaveform( '1 )
=> t
```

Deletes waveform `1` from the current subwindow.

```
deleteWaveform( "all" )
=> t
```

Deletes all the curves from the current subwindow.

displayMode

```
displayMode( t_mode )
     => t / nil
```

Description

Sets the display mode of the current subwindow.

Arguments

t_mode	String representing the display mode for the subwindow. Valid values: `strip`, `composite`, or `smith`. **Note:** This also works if a plot is not open.

Value Returned

`t`	Returns `t` when the display mode of the subwindow is set.
`nil`	Returns `nil` and an error message if the display mode cannot be set.

Example

```
displayMode( "composite" )
=> t
```

Sets the current subwindow to display in `composite` mode.

getAsciiWave

```
getAsciiWave( t_filename x_xColumn x_yColumn [x_xskip] [x_yskip])
     => o_wave/nil
```

Description

Reads in an Ascii file of data and generates a waveform object from the specified data. The X-axis data must be real numbers. The Y-axis data can be real or complex values. Complex values are represented as `(real imag)` or `complex(real imag)`. This function skips blank lines and comment lines. Comments are defined as lines beginning with a semicolon.

Arguments

t_filename	The name of the Ascii file to be read in.
x_xColumn	The column in the data file that contains the X-axis data.
x_yColumn	The column in the data file that contains the Y-axis data.
x_xskip	The number of lines to skip in the X column.
x_yskip	The number of lines to skip in the Y column.

Value Returned

o_wave	The waveform object
`nil`	Returns `nil` if the function fails.

Example

```
getAsciiWave("~/mydatafile.txt " 1 2 )
=> srrWave:32538648
```

Reads in an ascii file `~/mydatafile.txt`, which has x-axis data in the first column and y-axis data in the second column, and returns a waveform object.

```
getAsciiWave("~/mydatafile.txt " 1 2 ?xskip 1 ?yskip 2)
=> srrWave:32538656
```

Reads in an ascii file `~/mydatafile.txt`, which has x-axis data in the first column and y-axis data in the second column and skips `1` line in the `x_xcolumn` and `2` lines in the `y_ycolumn`, and returns a waveform object.

graphicsOff

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

Description

Disables the redrawing of the current Waveform window.

You might use this command to freeze the Waveform window display, send several plots to the window, and then unfreeze the window to display all the plots at once.

Arguments

None.

Value Returned

`t`	Returns `t` if redrawing is disabled.
`nil`	Returns `nil` if there is an error, such as there is no current Waveform window.

Example

```
graphicsOff()
=> t
```

Disables the redrawing of the Waveform window.

graphicsOn

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

Description

Enables the redrawing of the current Waveform window.

Arguments

None.

Value Returned

`t`	Returns `t` if redrawing is enabled.
`nil`	Returns `nil` if there is an error, such as there is no current Waveform window.

Example

```
graphicsOn()
=> t
```

Enables the redrawing of the current Waveform window.

hardCopy

```
hardCopy(w_windowId)
     => t / nil
```

Description

Sends a Waveform window plot to a printer or a file. To plot to a printer specify a printer name using the `?hcPrinterName` argument of the `hardCopyOptions` command. To plot to a file, specify a file name using the `?hcOutputFile` argument of the `hardCopyOptions` command.

Note: You must first set any plotting options with the hardCopyOptions command.

Arguments

w_windowId	The window ID of the waveform window whose plot is to be sent to a printer or a file. The default value is the window ID of the current window.

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` if there is an error.

Example

```
hardCopy()
=> t
```

Sends a waveform plot to the printer or to a file.

```
w = newWindow()
plot(v("/vout"))
hardCopy(w)
```

Sends the waveform plot of `w` to the printer or to a file.

hardCopyOptions

```
hardCopyOptions(
     [?hcCopyNum x_hcCopyNum]
     [?hcOffsetHeight x_hcOffsetHeight]
     [?hcOffsetWidth x_hcOffsetWidth]
     [?hcOrientation s_hcOrientation]
     [?hcOutputFile g_hcOutputFile]
     [?hcPrinterName s_hcPrinterName]
     [?hcTmpDir t_hcTmpDir]
     [?hcPaperSize s_hcPaperSize]
     [?hcMakeExactCopy g_hcMakeExactCopy]
     [?hcQuality x_hcQuality]
     [?hcOptimizeForWindows g_hcOptimizeForWindows]
     [?hcImageWidth x_hcImageWidth]
     [?hcImageHeight x_hcImageHeight]
     [?hcImageSizeUnits s_hcImageSizeUnits]
     [?hcImageResolution x_ImageResolution]
     [?hcResolutionUnits s_hcResolutionUnits]
     [?hcImageAspectRatio x_hcImageAspectRatio]
     [?hcUseExistingBackground g_hcUseExistingBackground]
     [?hcDisplayTitle g_hcDisplayTitle]
     [?hcDisplayLegend g_hcDisplayLegend]
     [?hcDisplayAxes g_hcDisplayAxes]
     [?hcDisplayGrids g_hcDisplayGrids]
     [?hcSaveEachSubwindowSeparately g_hcSaveEachSubwindowSeparately]
     )
     => g_value / nil
```

Description

Sets the graph window hardcopy plotting options.

The option takes effect for any graph window or subwindow that is opened after the option is set.

Arguments

x_hcCopyNum	The number of copies to plot. Valid values: any positive integer Default value: `1`
x_hcOffsetHeight	The vertical margin. Valid values: any positive integer Default value: `0`

x_hcOffsetWidth	The horizontal margin. Valid values: any positive integer Default value: `0`
s_hcOrientation	The plot orientation. **Note:** This option works only when you print a graph window and does not work when you save the window to a graph file. Valid values: `'portrait`, `'landscape`, `'automatic` Default value: `'automatic`
g_hcOutputFile	Name of the output file. The output file can be created in one of the following file formats: ■ BMP – Windows Device Independent Bitmap (`.bmp`) ■ PNG – Portable Network Graphics(`.png`) ■ PS – PostScript (`.ps`) ■ TIFF – Tagged Image File Format (`.tif`) ■ TIFF – Tagged Image File Format (`.tif`) ■ EPS – Encapsulated Post Script (`.eps`) ■ PDF – Portable Document Format (`.pdf)` ■ PPM – Portable PixMap File (`.ppm`) ■ JPG – Joint Photographic Experts Group (`.jpg`) ■ SVG – Scalable Vector Graphics (`.svg`) ■ XPM – X PixMap (`.xpm`) Valid values: a string or `nil` Default value: `nil`
s_hcPrinterName	The name of the printer. Valid values: a string or `nil` Default value: `nil`
t_hcTmpDir	The name of a temporary directory to be used for scratch space. Valid values: name of a temporary directory Default value: `"/usr/tmp"`

s_hcPaperSize	The paper size. The available paper sizes are—`letter`, `legal`, `executive`, `folio`, `ledger`, `tabloid`, `a0`, `a1`, `a2`, `a3`, `a4`, `a5`, `a6`, `a7`, `a8`, `a9`, `b0`, `b1`, `b2`, `b3`, `b4`, `b5`, `b6`, `b7`, `b8`, `b9`, `b10`, `c5e`, `comm10e`, `dle`. Default value: `a4`
g_hcMakeExactCopy	Saves the exact copy of all subwindows. Only `?hcQuality` and `?hcOutputFil`e arguments work with this option. This option does not work for the `eps` file format. Valid values: `t` or `nil` Default value: `nil`
x_hcQuality	Modifies the quality of the image. This option works only for the `.jpeg` file format. This option does not work for the `eps` file format. Valid values: `20` to `100%` Default value: `85`
g_hcOptimizeForWindows	Enables the image to be imported in the Microsoft office application. This option is available when you select the image type as Encapsulated PostScript (`*.eps`). This option simplifies the image output so that it can be ready by Microsoft Office 2003 and 2007 applications Valid values: `t` or `nil` Default value: `t`
x_hcImageWidth	Sets the width of the image. Valid values: Any positive integer value. Default value: `800` pixels
x_hcImageHeight	Sets the height of the image Valid values: Any positive integer value. Default value: `600` pixels
s_hcImageSizeUnits	Specifies the unit for image size (height and width) Valid values: `inch`, `cm`, `mm`, `picas`, `pixels`, and `points` Default value: `pixels`
x_hcImageResolution	Sets the image resolution. This option works only for the `bmp`, `jpeg`, `png`, `ppm`, `tif`, and `xpm` file formats. It does not work for `eps`, `pdf`, and `svg` file formats. Valid values: Any positive integer value. Default value: `96`

s_hcResolutionUnits	Sets the units for image resolution. This option works only for the `bmp`, `jpeg`, `png`, `ppm`, `tif`, and `xpm` file formats. It does not work for `eps`, `pdf`, and `svg` file formats. Valid values: `pixels/cm` and `pixels/in` Default value: `pixels/in`
g_hcImageAspectRatio	Enables the aspect ratio, which is the ratio of the width of the image to its height. Valid values: `t` or `nil` Default value: `nil`
g_hcUseExistingBackground	Enables to use the existing background in the graph image. Valid values: `t` or `nil` Default value: `nil`
g_hcDisplayTitle	Displays the trace title in the graph image. Valid values: `t` or `nil` Default value: `t`
g_hcDisplayLegend	Displays the trace legend in the graph image. Valid values: `t` or `nil` Default value: `t`
g_hcDisplayAxes	Displays the axes in the graph image. Valid values: `t` or `nil` Default value: `t`
g_hcDisplayGrids	Displays the grids in the graph image. Valid values: `t` or `nil` Default value: `t`
g_hcSaveEachSubwindowSeparately	Saves all subwindows in a graph to a single file or multiple files. Valid values: `t` (multiple files) or `nil` (single file) Default value: `t`

Value Returned

g_value	Returns the new value of the option.
`nil`	Returns `nil` if there is an error.

Example

```
hardCopyOptions( ?hcCopyNum 1 )
```

Plots one copy of the window or subwindow.

```
hardCopyOptions(?hcCopyNum 3 ?hcOutputFile "myOutFile.bmp")
```

Plots three copies of the window or subwindow and sends them to the file `myOutFile.bmp`.

ip3Plot

```
ip3Plot( o_wave x_sigHarmonic x_refHarmonic x_extrapolationPoint )
     => t / nil
```

Description

Plots the IP3 curves.

This command should be run on the results of the Spectre swept pss and pac analysis.

Refer to the “Simulating Mixers” chapter of the *Virtuoso Spectre Circuit Simulator RF Analysis User Guide* for more information on ip3Plot.

Arguments

o_wave	Waveform for which to plot the ip3.
x_sigHarmonic	Index of the third order harmonic.
x_refHarmonic	Index of the first order (fundamental) harmonic.
x_extrapolationPoint	Extrapolation point.

Value Returned

`t`	Returns `t` if the curves are plotted.
`nil`	Returns `nil` if there is an error.

Example

```
ip3Plot(v("/net28") 47 45 -25)
```

newWindow

```
newWindow()
     => w_windowID/nil
```

Description

Creates a new Waveform window and returns the window ID.

Arguments

None.

Value Returned

w_windowId	Returns the window ID of the new Waveform window.
`nil`	Returns `nil` and an error message if the new Waveform window cannot be created.

Example

```
newWindow()
=> window:3
```

Creates a new Waveform window that is numbered 3 in the upper right corner.

noiseSummary

```
noiseSummary(s_type [?result s_resultName [?resultsDir t_resultsDir]]
     [?frequency f_frequency] [?weight f_weight] [?output t_fileName | p_port]
     [?noiseUnit t_noiseUnit] [?truncateData x_truncateData]
     [?truncateType s_truncateType] [?digits x_digits]
     [?percentDecimals x_percentDecimals] [?from f_from] [?to f_to]
     [?deviceType ls_deviceType] [?weightFile t_weightFile]
     [?paramValues ls_paramValues])
     => t_fileName/p_port/nil
```

Description

Prints a report showing the noise contribution of each component in a circuit.

This command should be run on the results of the Spectre noise analysis.

Arguments

s_type — Type of noise-analysis results for which to print the report. Valid values: `spot`, to specify noise at a particular frequency, or `integrated`, to specify noise integrated over a frequency range.

s_resultName — Results from an analysis. When specified, this argumentwill only be used internally and will not alter the current result which was set by the selectResult command. The default is the current result selected with the selectResult command.

t_resultsDir — Directory containing the PSF files (results). If you supply this argument, you must also supply the resultName argument. When specified, this argument will only be used internally and will not alter the current results directory which was set by the openResults command. The default is the current results directory set by the openResults command.

f_frequency — Frequency value of interest.

f_weight — Waveform representing the function with which the integral is weighted.
Default value: `1.0`

t_fileName — File in which to write the information. The `noiseSummary` command opens the file, writes to the file, and closes the file. If you specify the filename without a path, the `noiseSummary` command creates the file in the directory pointed to by your Skill Path. To find out what your Skill path is, type `getSkillPath()` at the OCEAN prompt.

p_port — Port (previously opened with `outfile`) through which to append the information to a file. You are responsible for closing the port. See the outfile command for more information.

t_noiseUnit — Specifies the type of noise unit to be saved. Valid values: `"V^2"` for `V^2/Hz` or `"V"` for `V/sqrt( Hz )`

x_truncateData — Specifies a number that the `truncateType` argument uses to define the components for which information is to be printed.

s_truncateType — Specifies the method that is used to limit the data being included in the report.

Valid Values	Description	Sample Values
`'top`	Saves information for the number of components specified with `truncateData`. The components with the highest contributions are saved.	`10`
`'level`	Prints components which have noise contribution higher than that specified by `?truncateData`.	`10u`
`'relative`	Prints components which have noise contribution (percent) higher than that specified by `?truncateData`.	`.1`
`'none`	Saves information for all the components.	

x_digits — Number of significant digits with which the contributors are printed.

x_percentDecimals
Number of decimals printed for any relative contribution.

f_from For integrated noise, the start value for frequency.

f_to For integrated noise, the end value for frequency.

ls_deviceType
List of device type strings to be included.
Valid values: a list of strings or `'all`

t_weightFile Absolute or relative path of the file that contains information about weights. This data is used to compute weighted noise. If the values are provided for both parameters, `weight` and `weightFile`, the value for `weight` gets precedence.

ls_paramValues List of values where each value co-relates to a specific sweep variable name. This field must be used when the data is parametric. The order of this list must coincide with the list returned by the `sweepNames` function excluding the frequency variable.

Value Returned

t_fileName Returns the name of the port.

p_port Returns the name of the file.

`nil` Returns `nil` and an error message if the summary cannot be printed.

Example

```
noiseSummary( 'integrated ?result 'noiseSweep-noise )
```

Prints a report for an integrated noise analysis.

```
noiseSummary( 'integrated ?resultsDir
    "/usr/simulation/lowpass/spectre/schematic"
    ?result 'noise)
```

Prints a report for an `integrated` noise analysis for the results from a different run (stored in the `schematic` directory).

```
noiseSummary( 'spot ?resultsDir
    "/usr/simulation/lowpass/spectre/schematic"
    ?result 'noise ?frequency 100M )
```

Prints a report for a `spot` noise analysis at a frequency of `100M`.

```
noiseSummary('integrated ?truncateType 'none ?digits 10
?weightFile "./weights.dat")
```

Prints the weighted noise for an integrated noise analysis using information in the `weight` file `weights.dat`.

```
noiseSummary('integrated ?output "./NoiseSum1" ?noiseUnit "V" ?truncateData 20
?truncateType 'top ?from 10 ?to 10M ?deviceType list("bjt" "mos" "resistor"))
```

Prints a report for an integrated noise analysis in the frequency range `10-10M` for `20` components with deviceType `bjt`, `mos` or `resistor`.

```
noiseSummary( 'integrated ?from 1 ?to 100M  ?truncateType 'top ?truncateData 20
?deviceType 'all ?noiseUnit "V^2" ?output "./filename.ns" ?paramValues list(2.47e-
9))
```

Prints a report for an integrated noise analysis at a specific swept value.

ocnPrint

```
ocnPrint( [?output t_filename | p_port] [?precision x_precision]
     [?numberNotation s_numberNotation] [?numSpaces x_numSpaces] [?width
     x_width] [?from x_from] [?to x_to] [?step x_step] [?linLog t_linLog]
     o_waveform1 [o_waveform2 ...] )
     => t / nil
```

Description

Prints the text data of the waveforms specified in the list of waveforms.

If you provide a filename as the `?output` argument, the `ocnPrint` command opens the file and writes the information to it. If you provide a port (the return value of the SKILL `outfile` command), the `ocnPrint` command appends the information to the file that is represented by the port. There is a limitation of *ocnPrint* for precision. It works upto 30 digits for the Solaris port and 18 digits for HP and AIX.

Arguments

t_filename	File in which to write the information. The `ocnPrint` command opens the file, writes to the file, and closes the file. If you specify the filename without a path, the OCEAN environment creates the file in the directory pointed to by your Skill Path. To find out what your Skill path is, type `getSkillPath()` at the OCEAN prompt.
p_port	Port (previously opened with `outfile`) through which to append the information to a file. You are responsible for closing the port. See the outfile command for more information.
x_precision	The number of significant digits to print. This value overrides any global precision value set with the `setup` command. Valid values: `1` through `16` Default value: `6` **Note:** To print the specified significant number of digits, ensure that the value of the `x_width` argument is the same or greater than the value of the `x_precision` argument.
s_numberNotation	The notation for print ed information. This value overrides any global format value set with the `setup` command. Valid values: `'suffix`, `'engineering`, `'scientific`, `'none` Default value: `'suffix`

The format for each value is `'suffix`: `1m`, `1u`, `1n`, etc.; `'engineering`: `1e-3`, `1e-6`, `1e-9`, etc.; `'scientific`: `1.0e-2`, `1.768e-5`, etc.; `'none`.

The value `'none` is provided so that you can turn off formatting and therefore greatly speed up printing for large data files. For the fastest printing, use the `'none` value and set the `?output` argument to a filename or a port, so that output does not go to the CIW.

x_numSpaces	The number of spaces between columns. Valid values: `1` or greater Default value: `4`
x_width	The width of each column. Valid values: `4` or greater Default value: `14`
x_from	The start value at x axis for the waveform to be printed.
x_to	The end value at x axis for the waveform to be printed.
x_step	The step by which text data to be printed is incremented.
t_linLog	The scale to be used for printing. Valid values: `Linear`, `Log` Default value: `Linear`
o_waveform1	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)
o_waveform2	Additional waveform object.

Value Returned

`t`	Returns `t` if the text for the waveforms is printed.
`nil`	Returns `nil` and an error message if the text for the waveforms cannot be printed.

Example

```
ocnPrint( v( "/net56" ) )
=> t
```

Prints the text for the waveform for the voltage of `net56`.

```
ocnPrint( vm( "/net56" ) vp( "/net56" ) )
=> t
```

Prints the text for the waveforms for the magnitude of the voltage of `net56` and the phase of the voltage of `net56`.

```
ocnPrint( ?output "myFile" v( "net55" ) )
=> t
```

Prints the text for the specified waveform to a file named `myFile`.

```
ocnPrint( ?output "./myOutputFile" v("net1") ?from 0 ?to 0.5n ?step 0.1n )
```

Prints the text for the specified waveform from `0` to `0.5n` on the x axis in the incremental steps of `0.1n`.

ocnSetAttrib

```
ocnSetAttrib( [?XAxisLabel xLabel] [?YAxisLabel yLabel] [?XScale xscale] [?YScale
     yscale] [?XLimit xlimit] [?YLimit ylimit] [?YRange yrange]
     [?Origin origin] )
     => t / nil
```

Description

Sets the waveform window plotting attributes.

Arguments

XAxisLabel	Label (symbol or string) for the X axis in the waveform window.
YAxisLabel	Label (symbol or string) for the Y axis associated with the *stripNumber* in the waveform window.
XScale	Scale of the X axis in the waveform window. Valid values (symbols): `'auto`, `'log`, and `'linear`
YScale	Scale of the Y axis associated with the *stripNumber* in the waveform window. Valid values (symbols): `'log` and `'linear`
XLimit	Displays limits of the X axis in the waveform window. Valid values: List of two numbers or `'auto` (symbol). The first number in the list indicates the minimum limit and the second indicates the maximum limit. `'auto` sets the limit to autoscale.
YLimit	Displays limits of the Y axis associated with the *stripNumber* in the waveform window. Valid values: List of two numbers or `'auto` (symbol). The first number in the list indicates the minimum limit and the second indicates the maximum limit. `'auto` sets the limit to autoscale.
YRange	Y range (integer) of the waveforms associated with the *stripNumber* in the waveform window.
Origin	Axes origin of the waveform window. Valid values: List of two numbers.

Note: The valid range for *stripNumber* is 1-20.

Value Returned

`t`	Returns `t` if the valus of all arguments are set successfully.
`nil`	Returns `nil` if one or more arguments fail to set as specified.

Example

```
ocnSetAttrib( ?XAxisLabel 'XMylabel ?YAxisLabel 'YMyLabelt ?stripNumber 2
            )
=> t
```

Sets the X and Y axis labels to *XMylabel* and *YMyLabel*, respectively.

```
ocnSetAttrib(?XScale 'log ?YScale 'linear ?stripNumber 2 )
=> t
```

Sets the scale of X and Y axis to *log* and *linear*, respectively.

```
ocnSetAttrib(?XScale 'auto ?XLimit '(3 7) ?YLimit 'auto ?stripNumber 2 )
=> t
```

Sets the scale of X axis to autoscale. Sets the Y display limits to autoscale.

ocnWriteLsspToFile

```
ocnWriteLsspToFile(
     ?filename t_filename
     ?net1 input_node_name
     ?term1 input_src_terminal
     ?net2 output_node_name
     ?term2 output_src_terminal
     ?format t_format
     ?datafmt t_data_format
     ?port1 port1_name
     ?port2 port2_name
     ?result1 result1_name
     ?result2 result2_name
     )
     => nil
```

Description

Writes the large signal S-Parameter results to a file in Touchstone or Spectre format.

Arguments

t_filename	Name of the file in which results are to be written.
input_node_name	Name of the input node.
input_src_terminal	Name of the input source terminal.
output_node_name	Name of the output node.
output_src_terminal	Name of the output source terminal.
t_format	Format of file in which results are to be written. Possible values: `touchstone`, `spectre` Default value: '`touchstone`
t_data_format	Format of data being written. Possible values: `magphase`, `dbphase`, `realimag` Default value: `realimag`
port1_name	Name of the first port.
port2_name	Name of the second port.

result1_name Name of the first pss result.

result2_name Name of the second pss result.

Value Returned

`t` Specifies that the results are written to the specified file successfully.

`nil` Returns `nil` if the results are not written.

Example

```
ocnWriteLsspToFile "lssp.sp2" "/net026" "/PORT1/PLUS" "/RFOUT"
"/PORT2/PLUS" ?format "touchstone" ?datafmt "realimag" ?port1 50 ?port2 50
?result1 "sweeplssp1_lssp1_fd-sweep" ?result2 "sweeplssp2_lssp2_fd-sweep")
```

ocnYvsYplot

```
ocnYvsYplot([?wavex o_wavex ?wavey o_wavey] [?exprx o_exprx ?expry o_expry]
     [?titleList l_titleList] [?colorList l_colorList])
     => wave/nil
```

Description

Plots a wave against another wave or an expression against another expression.

This is currently supported for a family of waveforms generated from simple parametric simulation results data. It is not supported for a family of waveforms generated from parametric simulation with paramset, Corners or MonteCarlo results data.

Arguments

o_wavex	Reference wave against which the wave provided needs to be plotted.
o_wavey	Wave to be plotted against the reference wave.
o_exprx	Reference expression against which the expression provided needs to be plotted.
o_expry	Expression to be plotted against the reference expression.
l_titleList	List of waveform titles. If the waveform is simple, only one label will be required. If the waveform is param, a list of labels needs to be provided.
l_colorList	List specifying the colors for the waveforms. If you do not supply this argument, the default colors are used. The colors that are available are defined in your technology file. Valid Values: "`y1`" through "`y66`".

Value Returned

`wave`	Returns the waveform specified.
`nil`	Returns `nil` if the plot could not be generated.

Example

```
wy = VT("/vout")
wx = VT("/vin")
ex = "VT('/vin')"
ey = "VT('/vout')"
ocnYvsYplot(?wavex wx ?wavey wy ?titleList '("simpleWave") ?colorList '(y1))
```

Plots wave wy against wave wx with the title being simpleWave and the color being `y1`.

```
ocnYvsYplot(?exprx ex ?expry ey ?titleList '("simpleWave") ?colorList '(y2))
```

Plots expression ey against expression ex with the title being simpleWave and the color being `y2`.

plot

```
plot( o_waveform1 [o_waveform2 ...] [?expr l_exprList] [ ?strip x_stripNumber
    ] )
    => t / nil
```

Description

Plots waveforms in the current subwindow. If there is no Virtuoso Visualization and Analysis XL window, this command opens one.

Note: `plot` is implemented as a macro and not as a SKILL function. Therefore, the functions that expect a function name as an argument will not accept `plot` as a valid argument. For example, the following call to the function `apply` is not valid:

```
apply('plot)
```

Arguments

o_waveform1	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: srrWave:XXXXX.)
o_waveform2	Additional waveform object.
l_exprList	List of strings used to give names to the waveform objects.
x_stripNumber	An integer using which you can plot waveforms selectively on different strips and subwindows. If you specify an integer, it is used as the strip for all waveforms. To use the strip option for multiple waveforms, you can specify a list of strip numbers.

Important

Virtuoso Visualization and Analysis XLdoes not support *stripNumber*.

Value Returned

`t`	Returns `t` if the waveforms are plotted.
`nil`	Returns `nil` and an error message if the waveforms cannot be plotted.

Additional Information

Following are the scenarios that show how the `plot` and `displayMode` functions work together:

Case 1: When no waveform plot is open and you plot a waveform, `w1`, and then plot another waveform, `w2`, both the waveforms are plotted in one strip. Now if you set the `displayMode('strip)`function, the waveforms are plotted in two different strips.

1. `plot(w1)`
2. `plot(w2)`

   `w1` and `w2` are plotted in one strip.
3. `displayMode('strip)`

   `w1` and `w2` are plotted in two strips.
4. `plot(w3)`

   `w3` is plotted in a new strip. Note that the `?strip` argument is not required in this case.

Case 2: When no waveform plot is open and you set `displayMode('strip)`:

1. `displayMode('strip)`
2. `plot(w1)`
3. `plot(w2)`

   `w2` is plotted in a new strip. Note that the explicit `?strip` argument is not required in this case.

Case 3: When no waveform plot is open:

1. `plot(w1)`
2. `plot(w2)`

   `w1` and `w2` are plotted in one strip.

3. `plot(w3 ?strip 2)`

   `w3` is plotted in a new strip because the plot function contains the `?strip 2` argument.

4. `plot(w4)`

   `w4` is plotted in the same strip as in which `w3` is plotted.

Case 4:When no waveform plot is open:

1. `plot(w1 ?strip 1)`
2. `plot(w2 ?strip 2)`

   `w2` is plotted in strip 2.

3. `plot(w3 ?strip 1)`

   `w3` is plotted in strip 1.

4. `plot(w4 ?strip 2)`

   `w4` is plotted in strip 2, which now becomes an active strip.

5. `displayMode('strip)`

   This divides the traces contained in strip 2 into individual strips, because strip 2 is the active strip now. Now, if you plot new waveforms in this strip, they are plotted in new strips. However, strip 1continues to have two signals, `w1` and `w3`.

6. `plot(w5)`

   `w5` is plotted in a new strip.

7. `plot(w6)`

   `w6` is plotted in a new strip.

Case 5: When no waveform plot is open:

1. `window=awvCreatePlotWindow()`
2. `awvSetDisplayMode(window "strip")`
3. `plot(w1)`
4. `plot(w2)`
5. `plot(w3)`
6. `plot(w4)`

`w1`, `w2`, `w3`, and `w4` are plotted in four different strips.

Case 6: Plot digital and analog data and set `displayMode(`composite)`. This combines all analog signals into a single strip. Now, if you set the `displayMode (`strip)`, the analog signals are divided into individual strips. Note that these operations are applicable only on the active strip.

Example

```
plot(v( "/net56" ) )
```

Plots the waveform for the voltage of `net56`.

```
plot( vm( "/net56" ) vp( "/net56" ) )
```

Plots the waveforms for the magnitude of the voltage of `net56` and the phase of the voltage of `net56`.

```
plot( v( "OUT" ) i( "VFB" ) ?expr list( "voltage" "current" ) )
```

Plots the waveforms, but changes one legend label from `v("OUT")` to `voltage` and changes the other legend label from `i("VFB")` to `current`.

```
plot( v( "OUT" ) i( "VFB" ) )
```

Plots the waveforms `v( "OUT" )` and `i( "VFB" )` on the Y axes `1` and `2`, respectively.

```
plot(wave1 wave2 wave3 ?strip list(1 2 2))
```

Plots `wave1` to `strip 1`, and `wave2` and `wave3` to `strip 2`.

plotStyle

```
plotStyle( S_style )
     => t / nil
```

Description

Sets the plotting style for all the waveforms in the current subwindow.

If the plotting style is `bar` and the display mode is `smith`, the plotting style is ignored until the display mode is set to `strip` or `composite`.

Arguments

`S_style` Plotting style for the subwindow.
Valid values: `auto`, `scatterplot`, `bar`, `joined`

Argument	Description
`auto`	The appropriate plotting style is automatically chosen.
`scatterplot`	Data points are not joined.
`bar`	Vertical bars are drawn at each data point that extend from the point to the bottom of the graph.
`joined`	Each data point is joined to adjacent data points by straight-line segments.

Value Returned

`t` Returns `t` if the plotting style is set.

`nil` Returns `nil` and an error message if the plotting style is not set.

Example

```
plotStyle( 'auto )
=> t
```

Sets the plot style to `auto`.

printGraph

```
printGraph(
     [?window x_window]
     [?printerName s_hcPrinterName]
     [?horizontalMargin x_horizontalMargin]
     [?verticalMargin x_verticalMargin]
     [?numCopy x_numCopy]
     [?paperSize x_paperSize]
     [?orientation s_orientation]
     [?fileName s_fileName]
     [?tempDir s_tempDir]
     [?matchWindow g_matchWindow]
     [?numGraphsPerPage x_numGraphsPerPage]
     [?printMarkerTable g_printMarkerTable]
     [?markerTableLocation s_markerTableLocation]
     [?enableHeader g_enableHeader]
     [?enableFooter g_enableFooter]
     [?headerLeftText s_headerLeftText]
     [?headerCenterText s_headerCenterText]
     [?headerRightText s_headerRightText]
     [?footerLeftText s_footerLeftText]
     [?footerCenterText s_footerCenterText]
     [?footerRightText s_footerRightText]
     [?printColor g_printColor]
     [?doubleSidedPrint g_doubleSidedPrint]
     [?duplexMode s_duplexMode]
     [?pageOrder s_pageOrder]
     => t / nil
```

Description

Prints the graph plotted in the specified window.

Arguments

window	Window ID of the waveform window whose plot is to be sent to a printer or a file. The default value is the window ID of the current window.
printerName	Name of the printer to be used for printing. Valid values: a string or `nil` Default value: `nil`

horizontalMargin	Horizontal margin. Valid values: any positive integer Default value: `0`
verticalMargin	Vertical margin Valid values: any positive integer Default value: `0`
numCopy	Number of copies to be printed. Valid values: any positive integer Default value: `1`
paperSize	Size of paper used for printing. Valid values: `letter`, `legal`, `executive`, `folio`, `ledger`, `tabloid`, `a0`, `a1`, `a2`, `a3`, `a4`, `a5`, `a6`, `a7`, `a8`, `a9`, `b0`, `b1`, `b2`, `b3`,`b4`, `b5`, `b6`, `b7`, `b8`, `b9`, `b10`, `c5e`, `comm10e`, and `dle`. Default value: `a4`
orientation	Paper orientation. This option works only when you print a graph window and does not work when you save the window to a graph file. Valid values: `potrait`, `landscape`, and `automatic` Default value: `automatic`
fileName	Name of the output file. The output file can be created in one of the following file formats: PS – PostScript (`.ps`) PDF – Portable Document Format (`.pdf`) Valid values: any string value or `nil` Default value: `nil`
tempDir	Name of a temporary directory to be used for scratch space. Valid values: name of a temporary directory Default value: `"/usr/tmp"`
matchWindow	Specifies whether the print output is exactly similar to the current graph window. This option is used if you want to print all the subwindows in a PDF file in the same order in which they are arranged in the graph. Valid values: `t` or `nil` Default value: `nil`

numGraphsPerPage Specifies how many graphs are to be printed per page.
Valid values: Integer values `1`, `2`, `3`, `4`, `8`, `12`, `16`, `20`.

printMarkerTable Specifies whether the marker table is to be printed.
Valid values: `t` or `nil`.
Default value: `nil`.

MarkerTableLocation Specifies the location of the marker table on the page to be printed.
Valid values: `belowGraph` and `separatePage`.
Default value: `belowGraph`.

enableHeader Specifies whether the page contains a header.
Valid values: `t` or `nil`

enableFooter Specifies whether the page contains a footer.
Valid values: `t` or `nil`

headerLeftText Sets the text to be printed to the left of header.
Valid values: any string value
Default value: " "

headerCenterText Sets the text to be printed in the center of the header.
Valid values: Any string value and the following macros—
`$TOTALPAGES`, `$TITLE`, `$USERID`, `$PRINTER`, `$PAGE`, `$DATE`, `$DATETIME`, `$AUTHOR`, `$TIME`.
Default value: $`TITLE`

headerRightText Sets the text to be printed to the right of the header.
Valid values: Any string value and the following macros—
`$TOTALPAGES`, `$TITLE`, `$USERID`, `$PRINTER`, `$PAGE`, `$DATE`, `$DATETIME`, `$AUTHOR`, `$TIME`.
Default value: $`DATETIME`

footerLeftText Sets the text to be printed to the left of footer.
Valid values: Any string value and the following macros—
`$TOTALPAGES`, `$TITLE`, `$USERID`, `$PRINTER`, `$PAGE`, `$DATE`, `$DATETIME`, `$AUTHOR`, `$TIME`.
Default value: `Printed on $PRINTER by $USERID`

footerCenterText Sets the text to be printed in the center of the footer.
Valid values: Any string value and the following macros—
`$TOTALPAGES`, `$TITLE`, `$USERID`, `$PRINTER`, `$PAGE`, `$DATE`,

`$DATETIME`, `$AUTHOR`, `$TIME`.
Default value: " "

footerRightText	Sets the text to be printed on the right of the footer. Valid values: Any string value and the following macros—`$TOTALPAGES`, `$TITLE`, `$USERID`, `$PRINTER`, `$PAGE`, `$DATE`, `$DATETIME`, `$AUTHOR`, `$TIME`. Default value: `Page $PAGE of $TOTALPAGES`
printColor	Specifies whether the print is to be colored Valid values: `t` or `nil`
doubleSidedPrint	Specifies whether both the sides of paper is used for printing. Valid values: `t` or `nil`
duplexMode	Specifies the duplex printing mode. Valid values: `none`, `auto`, `shortSide`, `longSide`. Default value: `none`.
pageOrder	Specifies the order in which pages are printed. Valid values: `collate` and `reverse` Default value: `collate`

Value Returned

`t`	Returns `t` if the function runs successfully.
`nil`	Returns `nil` if there is an error.

Examples

```
printGraph()
```

Prints the current graph window with the default printing options.

```
printGraph(?printerName "ind001" ?paperSize "a4" ?orientation
'portrait)
```

Prints the current graph window by using the printer, `ind001`, with paper size `a4` and orientation `portrait`.

pzFrequencyAndRealFilter

```
pzFrequencyAndRealFilter(o_wave [?freqfilter f_fval] [?realfilter f_rval])
    => o_waveform / nil
```

Description

Returns a filtered Pole or Zero waveform from the pole zero simulation data. Filtering is done on the basis of given maximum frequency and minimum real value.

Note: This command also works for the parametric or sweep data.

Arguments

o_wave	Input Pole or Zero waveform (complex points) from the simulation data of PZ analysis.
f_val	Maximum pole and zero frequency value to filter out poles and zeros that are outside the frequency band of interest (FBOI) and that do not influence the transfer function in the FBOI.
f_rval	Minimum real value which is used to filter out poles and zeros whose real value are less than or equal to the value specified.

Values Returned

o_waveform	Returns a Pole or Zero waveform.
`nil`	Returns `nil` if there is an error.

Examples

```
pzFrequencyAndRealFilter(wave ?freqfilter 1e+24 ?realfilter 2e+10)
=> srrWave:175051584
```

Returns a filtered Pole or Zero waveform, which is filtered on the basis of given maximum frequency and minimum real value.

pzPlot

```
pzPlot( [?resultsDir t_resultsDir] [?result S_resultName] [?plot S_toPlot]
     [?freqfilter f_fval] [?realfilter f_rval])
     => t / nil
```

Description

Plots a report showing the poles and zeros of the network. If you specify a directory with `resultsDir`, the *pzPlot* command plots the results for that directory. The *`S_toPlot`* option can be used to plot only poles, only zeros or both poles and zeros information.

This command should be run on the results of the Spectre pz (pole-zero) analysis.

Note: This command also works for the parametric or sweep data.

Arguments

t_resultsDir	Directory containing the results. If you specify a directory with resultsDir, the *pzPlot* command plots the results for that directory.
S_resultName	Pointer to results from the analysis for which you want to plot the report.
S_toPlot	Use this option to plot only poles, only zeros or both poles and zeros information. Valid values: `'poles`, `'zeros`, `'polesZeros`.
f_fval	Maximum pole and zero frequency value to filter out poles and zeros that are outside the frequency band of interest (FBOI) and that do not influence the transfer function in the FBOI.
f_rval	Real value which is used to filter out poles and zeros whose real value are less than or equal to the value specified.

Value Returned

`t`	Returns `t` if it plots a report.
`nil`	Returns `nil` otherwise.

Example

```
pzPlot(?resultsDir "/usr/simulation/lowpass/spectre/schematic" ?result 'pz)
```

Plots a report for all the poles and zeros for the specified results.

```
pzPlot(?resultsDir "/usr/simulation/lowpass/spectre/schematic" ?plot 'poles)
```

Plots a report containing only poles for the specified results.

```
pzPlot( ?plot 'zeros ?realfilter -1.69e-01)
```

Plots a report for all those zeros whose real values are greater than the real value specified.

```
pzPlot( ?plot 'polesZeros ?freqfilter 2.6e-01 )
```

Plots a report for all those poles and zeros whose frequency is within the frequency band of interest (2.6e-01).

pzSummary

```
pzSummary( [?resultsDir t_resultsDir] [?result S_resultName]
     [?print S_toPrint] [?freqfilter f_fval] [?realfilter f_rval]
     [?output t_output])
     => t / nil
```

Description

Prints a report with the poles and zeros of the network. If you specify a directory with `resultsDir`, the *pzSummary* command prints the results for that directory. Use the *`S_toPrint`* option to print only poles, only zeros or both poles and zeros information.

This command should be run on the results of the Spectre pz (pole-zero) analysis.

Note: This command also works for the parametric or sweep data.

Arguments

`t_resultsDir`	Directory containing the results. If you specify a directory with resultsDir, the *pzSummary* command plots the results for that directory.
`S_resultName`	Pointer to results from the analysis for which you want to print the report.
`S_toPlot`	Use this option to plot only poles, only zeros or both poles and zeros information. Valid values: `'poles`, `'zeros`, `'polesZeros`.
`f_fval`	Maximum pole and zero frequency value to filter out poles and zeros that are outside the frequency band of interest (FBOI) and that do not influence the transfer function in the FBOI.
`f_rval`	Real value which is used to filter out poles and zeros whose real value are less than or equal to the value specified.
`t_output`	Provides an option to write the output to a file. The possible values can be a file name or a port name.

Value Returned

`t`	Returns `t` if it prints a report.

`nil` Returns `nil` otherwise.

Example

```
pzSummary(?resultsDir "/usr/simulation/lowpass/spectre/schematic" ?result 'pz)
```

Prints a report for all the poles and zeros for the specified results.

```
pzSummary(?resultsDir "/usr/simulation/lowpass/spectre/schematic" ?print 'poles)
```

Prints a report containing only poles for the specified results.

```
pzSummary( ?print 'zeros ?realfilter -1.69e-01)
```

Prints a report for all those zeros whose real values are less than or equal to the real value specified.

```
pzSummary( ?print 'polesZeros ?freqfilter 2.6e-01 )
```

Prints a report for all those poles and zeros whose frequency is within the frequency band of interest (2.6e-01).

```
pzSummary( ?output "/tmp/file")
```

Prints results in the file.

```
pzSummary( ?output "file")
```

Prints results in a file located in the current working folder.

```
pzSummary( ?output oFile)
where, oFile=outfile("/tmp/file")
```

Prints to the opened file

```
pzSummary( ?output nil)
pzSummary( ?output t)
pzSummary( ?output 32)
```

Prints results on the CIW or Ocean command-line.

removeLabel

```
removeLabel( l_id )
     => t / nil
```

Description

Removes the label, or all the labels identified in a list, from the current subwindow.

Arguments

l_id	List of labels to remove.

Value Returned

`t`	Returns `t` when the label or labels are removed.
`nil`	Returns `nil` if there is an error.

Example

```
label = addWindowLabel( list( 0.75 0.75 ) "test" )
```

Adds the `"test"` label to the current subwindow at the specified coordinates and stores the label identification number in `label`.

```
removeLabel( label )
```

Removes the label whose identification number is stored in `label`. In this case, the `"test"` label is removed.

report

```
report([?output t_filename | p_port] [?type t_type] [?name t_name]
    [?param t_param] [?format s_reportStyle] [?report s_reportStyle]
    [?maxLineWidth charsPerLine])
    => t / nil
```

Description

Prints a report of the information contained in an analysis previously specified with `selectResult`.

You can use this command to print operating-point, model, or component information. If you provide a filename as the `?output` argument, the `report` command opens the file and writes the information to it. If you provide a port (the return value of the SKILL `outfile` command), the `report` command appends the information to the file that is represented by the port.

Note: You can use the `dataTypes` command to see what types of reports you can choose. For Spectre® circuit simulator operating points, be sure to choose `dcOpInfo`.

Arguments

t_filename	File in which to write the information. The `report` command opens the file, writes to the file, and closes the file. If you specify the filename without a path, the OCEAN environment creates the file in the directory pointed to by your Skill Path. To find out what your Skill path is, type `getSkillPath()` at the OCEAN prompt.
p_port	Port (previously opened with `outfile`) through which to append the information to a file. You are responsible for closing the port. See the outfile command for more information.
t_type	Type of information to print, such as all bjts.
t_name	Name of the node or component.
t_param	Name of the parameter to print. It is also a list.
s_reportStyle	Specifies the format of the output. Valid values: `spice` and `paramValPair` Default value: `paramValPair`

The `spice` format looks like this:

	Param1	Param2	Param3
Name1	value	value	value
Name2	value	value	value
Name3	value	value	value

The `paramValPair` format looks like this:

Name1
Param1=value Param2=value Param3=value

Name2
Param1=value Param2=value Param3=value

Name3
Param1=value Param2=value Param3=value

charsPerLine Number of characters to be printed per line.

Value Returned

`t` Returns `t` if the information is printed.

`nil` Returns `nil` and an error message if the information cannot be printed.

Example

The following example shows how to display a report by using the results of an analysis already run. First, run the results() command to get a list of the type of results that exist in the current results directory.

```
results()
= > ( dcOpInfo tran ac dc)
```

From the list of result types returned by the previous function, select a particular type of results for which you want to print the report.

```
selectResult( 'dcOpInfo )
= > t
```

Use the report function to print the results. The following examples show how to print different details in a report:

```
report()
= > t
```

Prints all the operating-point parameters.

```
report( ?type "bjt" )
= > t
```

Prints all the `bjt` operating-point parameters.

```
report( ?type "bjt" ?param "ib" )
= > t
```

Prints the `ib` parameter for all bjts.

```
report( ?type "bjt" ?name "/Q1" ?param "ib" )
= > t
```

Prints the `ib` parameter for the `bjt` named `Q1`.

```
report( ?output "myFile" )
=> t
```

Prints all the operating-point parameters to a file named `myFile`.

```
report( ?output myAlreadyOpenedPort )
=> t
```

Prints all the operating-point parameters to a port named `myAlreadyOpenedPort`.

The `report()` can also be used by providing the set of parameters as a list as follows:

```
Type : bsim3v3
Params : cdg cgb gm ids
```

report(?type "bsim3v3" ?param "cdg")
report(?type "bsim3v3" ?param '("cdg" "cgb"))
report(?type "bsim3v3" ?param '("cdg" "cgb" "gm" "ids"))

```
report( ?format 'spice ?maxLineWidth 200 )
=> t
```

Prints the report in spice format wrapping at column 200.

saveGraphImage

```
saveGraphImage(
     [?window x_window]
     [?fileName x_fileName]
     [?exactcopy g_exactCopy
     [?quality x_quality]
     [?msOptimize g_msOptimize]
     [?width x_width]
     [?height x_height]
     [?units s_units]
     [?resolution x_resolution]
     [?resolutionUnits s_resolutionUnits]
     [?aspectRatio g_aspectRatio]
     [?enableTitle g_enableTitle]
     [?enableLegend g_enableLegend]
     [?enableAxes g_enableAxes]
     [?enableGrids g_enableGrids]
     [?backgroundColor s_backgroundColor]
     [?saveAllSubwindows g_saveAllSubwindows]
     [?saveEachSubwindowSeparately g_saveEachSubwindowSeparately]
     )
     => x_fileName / nil
```

Description

Saves the graph as an image.

Arguments

window	Window ID of the waveform window whose plot is to be saved in a file. The default value is the window ID of the current window.
fileName	Name of the output file to be created. The output file can be created in one of the following file formats: BMP – Windows Device Independent Bitmap (`.bmp`) PNG – Portable Network Graphics(`.png`) PS – PostScript (`.ps`) TIFF – Tagged Image File Format (`.tif`) EPS – Encapsulated Post Script (`.eps`) PDF – Portable Document Format (`.pdf`) PPM – Portable PixMap File (`.ppm`) JPG – Joint Photographic Experts Group (`.jpg`) SVG – Scalable Vector Graphics (`.svg`) XPM – X PixMap (`.xpm`)

	Valid values: any string value or `nil` Default value: `nil`. **Note:** If *`fileName`* argument is not specified, the graph image is saved in a `image.png` file.
`exactCopy`	Saves the exact copy of all subwindows. Only `?quality` and `?fileName` arguments work with this option. This option does not work for the `eps` file format. Valid values: `t` or `nil` Default value: `nil`
`quality`	Modifies the quality of the image. This option works only for the `.jpeg` file format. This option does not work for the `eps` file format. Valid values: `20` to `100%` Default value: `85%`
`msOptimize`	Enables the image to be imported in the Microsoft office application. This option is available when you select the image type as Encapsulated PostScript (`*.eps`). This option simplifies the image output so that it can be ready by Microsoft Office 2003 and 2007 applications Valid values: `t` or `nil` Default value: `t`
`width`	Sets the width of the image. Valid values: Any positive integer value. Default value: `800` pixels for `bmp`, `png`, `tiff`, `ppm` and `xpm` file formats and `8.33` inches for `pdf`, `svg` and `eps` file formats.
`height`	Sets the height of the image Valid values: Any positive integer value. Default value: `600` pixels for `bmp`, `png`, `tiff`, `ppm` and `xpm` file formats and `6.25` inches for `pdf`, `svg` and `eps` file formats.
`units`	Specifies the unit for image size (height and width) Valid values: `inch`, `cm`, `mm`, `picas`, `pixels`, and `points` Default value: `pixels` for `bmp`, `png`, `tiff`, `ppm` file formats and `xpm` and `inch` for `pdf`, `svg` and `eps` file formats.
`resolution`	Sets the image resolution. This option works only for the `bmp`, `jpeg`, `png`, `ppm`, `tif`, and `xpm` file formats. It does not work for `eps`, `pdf`, and `svg` file formats.

Valid values: Any positive integer value.
Default value: `96`

resolutionUnits
Sets the units for image resolution. This option works only for the `bmp`, `jpeg`, `png`, `ppm`, `tif`, and `xpm` file formats. It does not work for `eps`, `pdf`, and `svg` file formats.
Valid values: `pixels/cm` and `pixels/in`
Default value: `pixels/in`

aspectRatio
Enables the aspect ratio, which is the ratio of the width of the image to its height.
Valid values: `t` or `nil`
Default value: `nil`

enableBackground
Enables to use the existing background in the graph image.
Valid values: `t` or `nil`
Default value: `t`

enableTitle
Displays the trace title in the graph image.
Valid values: `t` or `nil`
Default value: `t`

enableLegend
Displays the trace legend in the graph image.
Valid values: `t` or `nil`
Default value: `t`

enableAxes
Displays the axes in the graph image.
Valid values: `t` or `nil`
Default value: `t`

enableGrids
Displays the grids in the graph image.
Valid values: `t` or `nil`
Default value: `t`

backgroundColor
Specify the background color.
Default value: `nil`, which means graph image is saved with the current background color
Valid values: All the valid color values are defined at the following location: http://www.w3.org/TR/SVG/types.html#ColorKeywords
For example, `red`, `blue`, `green`, `black`, `white`, `gray`, `cyan`, `magenta`, `yellow`, and `lightgray`

saveAllSubwindows Saves all subwindows or the current subwindow.
Default value: `t`
Valid values: `t` (current window is saved) or `nil` (all windows are saved)

saveEachSubwindowSeparately Specifies whether to save each subwindow in a separate image file or in the same image file.
Valid values: `t` or `nil`
Default value: `nil`

Value Returned

x_fileName Returns the name of the output file.

`nil` Returns `nil` if there is an error.

Examples

- `saveGraphImage()`

  Saves the current graph window with the default saving options.

- `saveGraphImage(?fileName "ViVA.jpg" ?enableTitle t ?enable Legend nil ?enableAxes nil ?enableBackground nil)`

  Saves the current graph window in the `ViVA.jpg` file with only trace title enabled.

- `saveGraphImage(?window currentWindow() ?fileName "ViVA.jpg" ?backgroundColor "light grey")`

  Saves the current graph window in the `ViVA.jpg` file with background color as light gray.

Additional Information

Following are the guidelines supported by the `saveGraphImage` function:

- Arguments *exactCopy*, *quality*, *resolution*, and *resolutionUnits* are ignored for `eps` file format.
- Only *fileName* and *quality* arguments can be used with *exactCopy* argument. All other arguments are ignored.
- Argument *quality* can be used only with `jpeg` file format. It is ignored for other formats.

- Arguments *resolution* and *resolutionUnits* cannot be set for `eps`, `pdf`, and `svg` file formats.
- Argument *msOptimize* can be set to `nil` only for `eps` file format.
- Argument *enableGrid*s cannot be set to `true` when *enableAxes* is `nil`.

xLimit

```
xLimit( l_minMax )
     => t / nil
```

Description

Sets the X axis display limits for the current subwindow. This command does not take effect if the display mode is set to `smith`.

Arguments

l_minMax	List of two numbers in waveform coordinates that describe the limits for the display. The first number is the minimum and the second is the maximum. If this argument is set to `nil`, the limit is set to `auto`.

Value Returned

`t`	Returns `t` when the X axis display limits are set.
`nil`	Returns `nil` and an error message if the X axis display limits are not set.

Example

```
xLimit( list( 1 100 ) )
=> t
```

Sets the X axis to display between 1 and 100.

yLimit

```
yLimit( l_minMax [?stripNumber x_stripNumber])
     => t / nil
```

Description

Sets the Y axis display limits for the waveforms associated with a particular Y axis and strip in the current subwindow.

If you do not specify *x_stripNumber*, the limits are applied when the subwindow is in `composite` mode.

Arguments

l_minMax	List of two numbers in waveform coordinates that describe the limits for the display. The first number is the minimum and the second is the maximum. If this argument is set to `nil`, the limit is set to `auto`.
x_stripNumber	Specifies the strip in which the y display is to be limited in the range specified by *l_minMax*. Valid values: `1` through `20`

Value Returned

`t`	Returns `t` if the Y axis display limits are set.
`nil`	Returns `nil` and an error message if the Y axis display limits cannot be set.

Example

```
yLimit( list( 4.5 7.5 ) )
=> t
```

Sets Y axis `1` to display from `4.5` to `7.5`.

Plotting and Printing SpectreRF Functions in OCEAN

You can access SpectreRF functions in OCEAN by using the `getData` function and then plot or print them in OCEAN using the `ocnPrint` and `plot` functions.

To take an example, after performing a spectre sp analysis in the Analog Design Environment, click *Results – Direct Plot – Main Form*. In the S-Parameter Results form, select the function and other options that you want to plot. Also, select the *Add to Outputs* option under the *Plot* button. Then, click *OK*. The expression will be added to the *Outputs* pane of the ADE window. When all the desired expressions are created in the *Outputs* pane, use the *ADE – Session – Save Ocean Script* command to create the OCEAN script for these plots.

To plot the expression in OCEAN, use the following command:

```
plot(<expression in Output pane>)
```

For example,

```
plot(Gmax())     for Gmax in S-parameter analysis
```

You can print the functions using the `ocnPrint` command. For example:

```
ocnPrint( Gmax() Kf() )
```

After a spectre sp noise analysis, use the following command to access the sp noise data.

```
selectResult("sp_noise")
```

A sample OCEAN script to help you print or plot NFmin (minimum noise figure), N F (noise figure), and RN (noise resistance) results follows. Plotting NNR (normalized noise resistance) is very similar to plotting RN.

```
; start ocean with Spectre as the as the simulator.
simulator( 'spectre )
;specify design and model path
design(  "/usr1/mnt4/myhome/simulation/myckt/schematic/netlist/myckt.c")
path( "/usr1/mnt4/myhome/models" )
; specify analysis used:  sp with noise
analysis('sp ?start "100M"  ?stop "10G"  ?donoise "yes"
?oprobe "/PORT1"  ?iprobe "/PORT0"  )
;set design variables
desVar(   "r2" 37)
desVar(   "r1" 150)
;set temperature
temp( 25 )
;run sp noise analysis with the above desVar list.
run()
```

```
printf("\n simulation has finished.")
printf("\n selecting sp noise results")
selectResult("sp_noise")
printf("\n print NFmin and plot NF")
NFmin = getData("NFmin")
NF = getData("NF")
ocnPrint( NFmin )
plot( NF )
printf("\n plot Rn")
Rn = getData("RN" ?result "sp_noise")
plot( Rn ?expr '( "Rn" ) )
exit
```

For more information, see the section *Periodic Noise Analysis* and the appendix *Plotting Spectre S-Parameter Simulation Data* in the *Virtuoso Spectre Circuit Simulator RF Analysis User Guide*.

For more information on these functions, click these links: getData, sp, ocnPrint,and plot.

9

OCEAN Aliases

The aliases in this chapter provide you with shortcuts to commonly used pairs of commands. By default, these aliases operate on results previously selected with selectResult. However, you can also use an alias on a different set of results. For example, to specify a different set of results for the `vm` alias, use the following syntax.

```
vm( t_net [?result s_resultName] )
```

where *s_resultName* is the name of the datatype for the particular analysis you want.

You can use the `vm` alias on results stored in a different directory as follows:

```
vm( t_net [?resultsDir t_resultsDir] [?result s_resultName] )
```

where *t_resultsDir* is the name of a different directory containing PSF results, and *s_resultName* is the name of a datatype contained in that directory. (If you specify another directory with *t_resultsDir*, you must also specify the particular results with *s_resultName*.)

List of Aliases

Alias	Syntax	Description
`vm`	`vm(` *t_net [*?resultsDir *t_resultsDir] [*?result *s_resultname]*) => *o_waveform*/ `nil`	Aliased to `mag(v())`. Gets the magnitude of the voltage of a net.
`vdb`	`vdb(` *t_net [*?resultsDir *t_resultsDir] [*?result *s_resultname]*) => *o_waveform*/ `nil`	Aliased to `db20(v())`. Gets the power gain in decibels from net `in` to net `out`.
`vp`	`vp(` *t_net [*?resultsDir *t_resultsDir] [*?result *s_resultname]*) => *o_waveform*/ `nil`	Aliased to `phase(v())`. Gets the phase of the voltage of a net.

List of Aliases, *continued*

vr	vr(*t_net [*?resultsDir *t_resultsDir] [*?result *s_resultname]*) => *o_waveform*/ nil	Aliased to real(v()). Gets the real part of a complex number representing the voltage of a net.
vim	vim(*t_net [*?resultsDir *t_resultsDir] [*?result *s_resultname]*) => *o_waveform*/ nil	Aliased to imag(v()). Gets the imaginary part of a complex number representing the voltage of a net.
im	im(*t_component [*?resultsDir *t_resultsDir] [*?result *s_resultName]*) => o_*waveform*/ nil	Aliased to mag(i()). Gets the magnitude of the AC current through a component.
ip	ip(*t_component [*?resultsDir *t_resultsDir] [*?result *s_resultName]*) => o_*waveform*/ nil	Aliased to phase(i()). Gets the phase of the AC current through a component.
ir	ir(*t_component [*?resultsDir *t_resultsDir] [*?result *s_resultName]*) => o_*waveform*/ nil	Aliased to real(i()). Gets the real part of a complex number representing the AC current through a component.
iim	iim(*t_component [*?resultsDir *t_resultsDir] [*?result *s_resultName]*) => o_*waveform*/ nil	Aliased to imag(i()). Gets the imaginary part of a complex number representing the AC current through a component.

10

Predefined and Waveform (Calculator) Functions

This chapter contains information about the following functions. Some additional predefined data access commands are described in the *Virtuoso Analog Design Environment L SKILL Language Reference*.

Predefined Arithmetic Functions

Several functions are predefined in the Virtuoso® SKILL language. The full syntax and brief definitions for these functions follows the table.

Predefined Arithmetic Functions

Synopsis	Result
General Functions	
`add1(n)`	`n` + 1
`abs`	| `n` |
`sub1(n)`	`n` – 1
`exp(n)`	e raised to the power `n`
linRg(*n_from*, *n_to*, *n_by*)	Returns list of numbers in linear range from *n_from* to *n_to* in *n_by* steps
`log(n)`	Natural logarithm of `n`

Predefined Arithmetic Functions

Synopsis	Result
logRg(*n_from*, *n_to*, *n_by*)	Returns list of numbers in log10 range from *n_from* to *n_to* in *n_by* steps
max(n1 n2 …)	Maximum of the given arguments
min(*n1 n2* …)	Minimum of the given arguments
mod(*x1 x2*)	*x1* modulo *x2*, that is, the integer remainder of dividing *x1* by *x2*
round(*n*)	Integer whose value is closest to *n*
sqrt(*n*)	Square root of *n*
Trigonometric Functions	
sin(*n*)	sine, argument *n* is in radians
cos(*n*)	cosine
tan(*n*)	tangent
asin(*n*)	arc sine, result is in radians
acos(*n*)	arc cosine
atan(*n*)	arc tangent
Random Number Generator	
random(*x*)	Returns a random integer between 0 and *x*-1. If random is called with no arguments, it returns an integer that has all of its bits randomly set.
srandom(*x*)	Sets the initial state of the random number generator to *x*.

abs

```
abs( n_number )
     => n_result
```

Description

Returns the absolute value of a floating-point number or integer.

Arguments

n_number	Floating-point number or integer.

Value Returned

n_result	The absolute value of *n_number*.

Example

```
abs( -209.625)
=> 209.625
```

```
abs( -23)
=> 23
```

acos

```
acos( n_number )
     => f_result
```

Description

Returns the arc cosine of a floating-point number or integer.

Arguments

n_number	Floating-point number or integer.

Value Returned

f_result	Returns the arc cosine of *n_number*.

Example

```
acos(0.3)
=> 1.266104
```

add1

```
add1( n_number )
    => n_result
```

Description

Adds `1` to a floating-point number or integer.

Arguments

n_number	Floating-point number or integer to increase by `1`.

Value Returned

n_result	*n_number* plus `1`.

Example

```
add1( 59 )
=> 60
```

Adds 1 to 59.

asin

```
asin( n_number )
     => f_result
```

Description

Returns the arc sine of a floating-point number or integer.

Arguments

n_number	Floating-point number or integer.

Value Returned

f_result	The arc sine of *n_number*.

Example

```
asin(0.3)
=> 0.3046927
```

atan

```
atan( n_number )
     => f_result
```

Description

Returns the arc tangent of a floating-point number or integer.

Arguments

n_number	Floating-point number or integer.

Value Returned

f_result	The arc tangent of *n_number*.

Example

```
atan(0.3)
=> 0.2914568
```

cos

```
cos( n_number )
     => f_result
```

Description

Returns the cosine of a floating-point number or integer.

Arguments

n_number	Floating-point number or integer.

Value Returned

f_result	The cosine of *n_number*.

Example

```
cos(0.3)
=> 0.9553365
```

```
cos(3.14/2)
=> 0.0007963
```

exp

```
exp( n_number )
     => f_result
```

Description

Raises `e` to a given power.

Arguments

n_number	Power to raise `e` to.

Value Returned

f_result	The value of `e` raised to the power *n_number*.

Example

```
exp( 1 )
=> 2.718282
```

```
exp( 3.0 )
=> 20.08554
```

int

```
int( n_arg )
     => x_result
```

Description

Returns the largest integer not larger than the given argument.

Note: This function works on vector as well as waveform data. The function is applied to individual elements of the vector and waveform data.

Arguments

n_arg	A numeric value (which can be integer or floating point number).

Value Returned

x_result	The value of the largest integer not larger than the value specified by `n_arg`.

Example

```
int( 3.01 )
=> 3
```

```
int( 3.99 )
=> 3
```

linRg

```
linRg( n_from n_to n_by )
     => l_range/nil
```

Description

Returns a list of numbers in the linear range from *n_from* to *n_to* incremented by *n_by*.

Arguments

n_from	Smaller number in the linear range.
n_to	Larger number in the linear range.
n_by	Increment value when stepping through the range.

Value Returned

l_range	List of numbers in the linear range.
nil	Returned if error.

Example

```
range = linRg(-30 30 5)
(-30 -25 -20 -15 -10 -5 0 5 10 15 20 25 30)
```

log

```
log( n_number )
     => f_result
```

Description

Returns the natural logarithm of a floating-point number or integer.

Arguments

n_number	Floating-point number or integer.

Value Returned

f_result	The natural logarithm of *n_number*.

Example

```
log( 3.0 )
=> 1.098612
```

logRg

```
logRg( n_from n_to n_by )
     => l_range/nil
```

Description

Returns a list of numbers in the log10 range from *n_from* to *n_to* advanced by *n_by*.

The list is a geometric progression where the multiplier is 10 raised to the 1/*n_by* power. For example if *n_by* is 0.5, the multiplier is 10 raised to the 2nd power or 100.

Arguments

n_from	Smaller number in the linear range.
n_to	Larger number in the linear range.
n_by	Increment value when stepping through the range.

Value Returned

l_range	List of numbers in the linear range.
nil	Returned if error.

Example

```
logRg(1 1M 0.5)
(1.0 100.0 10000.0 1000000.0)
```

max

```
max( n_num1 n_num2 [n_num3 …] )
     => n_result
```

Description

Returns the maximum of the values passed in. Requires a minimum of two arguments.

Arguments

n_num1	First value to check.
n_num2	Next value to check.
[*n_num3…*]	Additional values to check.

Value Returned

n_result	The maximum of the values passed in.

Example

```
max(3 2 1)
=> 3
```

```
max(-3 -2 -1)
=> -1
```

min

```
min( n_num1 n_num2 [n_num3 …] )
     => n_result
```

Description

Returns the minimum of the values passed in. Requires a minimum of two arguments.

Arguments

n_num1	First value to check.
n_num2	Next value to check.
[*n_num3…*]	Additional values to check.

Value Returned

n_result	The minimum of the values passed in.

Example

```
min(1 2 3)
=> 1
```

```
min(-1 -2.0 -3)
=> -3.0
```

mod

```
mod( x_integer1 x_integer2 )
     => x_result
```

Description

Returns the integer remainder of dividing two integers. The remainder is either zero or has the sign of the dividend.

Arguments

x_integer1	Dividend.
x_integer2	Divisor.

Value Returned

x_result	The integer remainder of the division. The sign is determined by the dividend.

Example

```
mod(4 3)
=> 1
```

random

```
random( [x_number] )
     => x_result
```

Description

Returns a random integer between `0` and *`x_number`* minus `1`.

If you call `random` with no arguments, it returns an integer that has all of its bits randomly set.

Arguments

`x_number`	An integer.

Value Returned

`x_result`	Returns a random integer between `0` and *`x_number`* minus `1`.

Example

```
random( 93 )
=> 26
```

round

```
round( n_arg )
     => x_result
```

Description

Rounds a floating-point number to its closest integer value.

Arguments

n_arg	Floating-point number.

Value Returned

x_result	The integer whose value is closest to *n_arg*.

Example

```
round(1.5)
=> 2
round(-1.49)
=> -1
round(1.49)
=> 1
```

sin

```
sin( n_number )
     => f_result
```

Description

Returns the sine of a floating-point number or integer.

Arguments

n_number	Floating-point number or integer.

Value Returned

f_result	The sine of *n_number*.

Example

```
sin(3.14/2)
=> 0.9999997
```

```
sin(3.14159/2)
=> 1.0
```

Floating-point results from evaluating the same expressions might be machine-dependent.

sqrt

```
sqrt( n_number )
     => f_result
```

Description

Returns the square root of a floating-point number or integer.

Arguments

n_number	Floating-point number or integer.

Value Returned

f_result	The square root of *n_number*.

Example

```
sqrt( 49 )
=> 7.0
```

```
sqrt( 43942 )
=> 209.6235
```

srandom

```
srandom( x_number )
     => t
```

Description

Sets the seed of the random number generator to a given number.

Arguments

x_number	An integer.

Value Returned

`t`	This function always returns `t`.

Example

```
srandom( 89 )
=> t
```

sub1

```
sub1( n_number )
     => n_result
```

Description

Subtracts `1` from a floating-point number or integer.

Arguments

n_number	Floating-point number or integer.

Value Returned

n_result	Returns *n_number* minus `1`.

Example

```
sub1( 59 )
=> 58
```

Subtracts 1 from 59.

tan

```
tan( n_number )
     => f_result
```

Description

Returns the tangent of a floating-point number or integer.

Arguments

n_number	Floating-point number or integer.

Value Returned

f_result	The tangent of *n_number*.

Example

```
tan( 3.0 )
=> -0.1425465
```

xor

```
xor( g_in1 g_in2 )
     => g_res
```

Description

Returns the XOR value of the boolean inputs.

Arguments

g_in1	The first boolean input.
g_in2	The second boolean input.

Value Returned

g_res	The resultant XOR value.

Example

```
xor(nil nil)
=> nil
xor(t nil)
=> t
xor(nil t)
=> t
xor(t t)
=> nil
```

Waveform (Calculator) Functions

The calculator commands are described in this section.

average

```
average( o_waveform )
    => n_average/o_waveformAverage/nil
```

Description

Computes the average of a waveform over its entire range.

Average is defined as the integral of the expression `f(x)` over the range of x, divided by the range of x.

For example, if `y=f(x), average(y)=`

$$\frac{\int_{from}^{to} f(x)dx}{to - from}$$

where `from` is the initial value for x and `to` is the final value.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)

Value Returned

n_average	Returns a number representing the average value of the input waveform.
o_waveformAverage	Returns a waveform (or family of waveforms) representing the average value if the input is a family of waveforms.
`nil`	Returns `nil` and an error message otherwise.

Example

```
average( v( "/net9" ) )
```

Gets the average voltage (Y-axis value) of `/net9` over the entire time range specified in the simulation analysis.

abs_jitter

```
abs_jitter(o_waveform t_crossType n_threshold ?xUnit t_xUnit ?yUnit t_yUnit
    ?Tnom n_Tnom)
    => o_waveform/nil
```

Description

Calculates the absolute jitter values in the intput waveform for the given threshold. The output waveform can be expressed in degrees, radians, or unit intervals (UI). The absolute jitter can be plotted as a function of cycle number, crossing time, or reference clock time.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)
t_crossType	The points at which the curves of the waveform intersect with the threshold. While intersecting, the curve may be either rising or falling. Valid values: `rising` and `falling`, respectively. Default `crossType` is `rising`.
n_threshold	The threshold value against which the at which the input waveform intersects to calculate the absolute jitter.
t_xUnit	The unit defined for X-axis of the output waveform. Valid values: `s` (time) and `cycle`. Default: `s` Cycle numbers refer to the n'th occurrence where the waveform crosses the given threshold.
t_yUnit	The unit defined for Y-axis of the output waveform. Valid values: `rad` (radians), `UI` (unit intervals), and `S` (degrees) Default value: `rad`.
n_Tnom	The nominal time period of the input waveform. The waveform is expected to be a periodic waveform that contains noise. If *Tnom* is `nil`, the `abs_jitter` function finds the approximate average time period of the input waveform. Default value: `nil`.

Value Returned

o_waveform	Returns a waveform representing the absolute jitter value for the given threshold.
nil	Returns nil and an error message otherwise.

Example

```
abs_jitter(v("net9" "rising" 1.0 ?xUnit "cycle" ?yUnit "UI" )
```

Gets the absolute jitter /net9 for the threshold value 1.0. Tnom value is selected as nil.

awvCreateBus

```
awvCreateBus(w_bus l_wavelist r_radix)
```

Definition

Creates a bus with the given digital signals and radix.

Arguments

w_bus	Name of the digital waveform representing a bus.
l_wavelist	List of the digital waveforms in the bus.
r_radix	Radix of the bus.

Value Returned

None.

Example

Following are the examples to create a digital binary bus with name *bus*.

```
awvCreateBus("bus" list( awvAnalog2Digital( v("/data<0> " ?result
"tran-tran") nil nil 0.5 nil "centre")

awvAnalog2Digital( v("/datab<1> " ?result "tran-tran")  nil nil 0.5
nil "centre")

awvAnalog2Digital( v("/data<1> " ?result "tran-tran") nil nil 0.5 nil
"centre")

awvAnalog2Digital( v("/datab<0> " ?result "tran-tran")  nil nil 0.5
nil "centre") ) "Binary")
```

awvPlaceXMarker

```
awvPlaceXMarker( w_windowId n_xLoc [?subwindow x_subwindowId] )
     => t_xLoc/t/nil
```

Description

Places a vertical marker at a specific x-coordinate in the optionally specified subwindow of the specified window.

Arguments

w_windowId	Waveform window ID.
n_xLoc	The x-coordinate at which to place the marker.
x_subwindowId	Waveform subwindow ID.

Value Returned

t_xLoc	Returns a string of x-coordinates if the command is successful and the vertical marker info form is opened.
t	Returns this when the command is successful but the vertical marker info form is not opened.
nil	Returns `nil` or an error message.

Example

```
awvPlaceXMarker( window 5)
=> "5"
```

Vertical marker info form is opened when the command is executed.

```
awvPlaceXMarker( window 6 ?subwindow 2)
=> t
```

Vertical marker info form is not opened.

awvPlaceYMarker

```
awvPlaceYMarker(w_windowId n_yLoc [?subwindow x_subwindowId])
     => t_yLoc/t/nil
```

Description

Places a horizontal marker at a specific y-coordinate in the optionally specified subwindow of the specified window.

Arguments

w_windowId	Waveform window ID.
n_yLoc	The y-coordinate at which to place the marker.
x_subwindowId	Waveform subwindow ID.

Value Returned

t_yLoc	Returns a string of y-coordinates if the command is successful and the horizontal marker info form is opened.
t	Returns this when the command is successful but the horizontal marker info form is not opened.
nil	Returns `nil` or an error message.

Example

```
awvPlaceYMarker( window 5)
=> "5"
```

Horizontal marker info form is opened when the command is executed.

```
awvPlaceYMarker( window 6 ?subwindow 2)
=> t
```

Horizontal marker info form is not opened.

awvRefreshOutputPlotWindows

```
awvRefreshOutputPlotWindows(s_session)
```

Description

Refreshes all existing plot windows (with new simulation data, if any) attached with the session *s_session*.

Arguments

s_session	Currently active environment variable.

Value Returned

None.

b1f

```
b1f( o_s11 o_s12 o_s21 o_s22 )
     => o_waveform/nil
```

Description

Returns the alternative stability factor in terms of the supplied parameters.

Arguments

o_s11	Waveform object representing s11.
o_s12	Waveform object representing s12.
o_s21	Waveform object representing s21.
o_s22	Waveform object representing s22.

Value Returned

o_waveform	Waveform object representing the alternative stability factor.
nil	Returns `nil` and an error message otherwise.

Example

```
s11 = sp(1 1)
s12 = sp(1 2)
s21 = sp(2 1)
s22 = sp(2 2)
plot(b1f(s11 s12 s21 s22))
```

bandwidth

```
bandwidth( o_waveform n_db t_type )
     => n_value/o_waveform/nil
```

Description

Calculates the bandwidth of a waveform.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)
n_db	Positive number that defines the bandwidth.
t_type	Type of input filter. Valid values: `"low"`, `"high"` or `"band"`.

Value Returned

n_value	Returns a number representing the value of the bandwidth if the input argument is a single waveform.
o_waveform	Returns a waveform (or family of waveforms) representing the bandwidth if the input argument is a family of waveforms.
`nil`	Returns `nil` and an error message otherwise.

Example

```
bandwidth( v( "/OUT" ) 3 "low")
```

Gets the 3 dB bandwidth of a low-pass filter.

```
bandwidth( v( "/OUT" ) 4 "band" )
```

Gets the 4 dB bandwidth of a band-pass filter.

clip

```
clip( o_waveform n_from n_to )
     => o_waveform/nil
```

Description

Restricts the waveform to the range defined by *n_from* and *n_to*.

You can use the clip function to restrict the range of action of other commands. If *n_from* is `nil`, *n_from* is taken to be the first X value of the waveform, and if *n_to* is `nil`, *n_to* is taken to be the last X value of the waveform. If both *n_to* and *n_from* are `nil`, the original waveform is returned.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)
n_from	Starting value for the range on the X axis.
n_to	Ending value for the range on the X axis.

Value Returned

o_waveform	Returns a waveform object if the input argument is a waveform object or returns a family of waveforms if the input argument is a family of waveforms.
`nil`	Returns `nil` and an error message otherwise.

Example

```
x = clip( v( "/net9" ) 2m 4m )
plot( x )
```

Plots the portion of a waveform that ranges from 2 ms to 4 ms.

```
plot( clip( v( "/net9" ) nil nil ) )
```

Plots the original waveform.

```
plot( clip( v( "/net9" ) nil 3m ) )
```

Plots the portion of a waveform that ranges from 0 to 3 ms.

clipX

```
clipX( o_waveform n_from n_to )
     => o_waveform/nil
```

Description

Restricts the waveform to the range defined by *n_from* and *n_to*.

The *clipX* works in the same manner as the *clip* function works, with an exception that *clipX* does not extrapolate values where as *clip* extrapolates values beyond the range.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)
n_from	Starting value for the range on the X axis.
n_to	Ending value for the range on the X axis.

Value Returned

o_waveform	Returns a waveform object if the input argument is a waveform object or returns a family of waveforms if the input argument is a family of waveforms.
`nil`	Returns `nil` and an error message otherwise.

closeResults

```
closeResults(t_dirName)
    => t/nil
```

Definition

Closes the simulation results stored in the input results directory. The function closes all the internal resources opened by the tool that are related to the results directory. It is recommended that you must call this function before deleting a results directory, moving the directory to any other location, or renaming a results directory.

After calling the `closeResults` function, the OCEAN commands, such as `selectResults`, `getData`, `pv`, which can also be called without passing the `resultsDir` argument and run based upon previously called `openResults` call, stops working if called without passing the `resultsDir` argument.

Arguments

`t_dirName`	Name of the directory which was earlier used in the `openResults` function.

Values Returned

`t`	If the results database has been closed successfully.
`nil`	If the results database has not been closed successfully.

compare

```
compare( o_waveform1 o_waveform1 [f_abstol [f_reltol]] )
    => o_comparisonWaveform/nil
```

Description

Compares the two given waveforms based on the specified values for absolute and relative tolerances. This function compares only the sections of the two waveforms where the X or independent axes overlap.

The following situations are possible:

- If neither relative nor absolute tolerance is specified, the function returns the difference of the two waveforms (*o_waveform1 - o_waveform2*).
- If only the absolute tolerance is specified, the function returns the difference of the two waveforms only when the absolute value of the difference is greater than the absolute tolerance (|*o_waveform1 - o_waveform2*| > *f_abstol*); otherwise it returns a zero waveform.
- If only the relative tolerance is specified, the function returns the difference of the two waveforms only when the absolute value of the difference is greater than the product of the relative tolerance and the larger of the absolute values of the two waveforms (|*o_waveform1 - o_waveform2*| > *f_reltol* * max(|*o_waveform1*|, |*o_waveform2*|)); otherwise it returns a zero waveform.
- If both relative and absolute tolerances are specified, the function returns the difference of the two waveforms only when the absolute value of the difference is greater than the sum of the separately calculated tolerance components (|*o_waveform1 - o_waveform2*| > *f_abstol* + *f_reltol* * max(|*o_waveform1*|, |*o_waveform2*|)); otherwise it returns a zero waveform.

Note: The function also compares parametric waveforms. However, for a successful comparison of parametric waveforms, the family tree structures of the two input waveforms should be the same. For both the input waveforms, the number of child waveforms at each level should also be the same, except at the leaf level where the elements are simple scalars. This is an obvious condition to obtain a meaningful comparison.

Arguments

o_waveform1	Waveform 1.
o_waveform2	Waveform 2.

f_abstol — Absolute tolerance.
Default value: `0.0`

f_reltol — Relative tolerance.
Default value: `0.0`

Value Returned

o_comparisonWaveform
Returns the difference of the two given waveforms based on the specified values of the relative and absolute tolerances.

`nil` — Returns `nil` and an error message otherwise.

Example

```
compare( wave1 wave2 2.2 0.4 )
=> srrWave:175051528
```

Returns the difference of the waveforms `wave1` and `wave2` based on the specified absolute and relative tolerances of `2.2` and `0.4`, respectively.

compression

```
compression( o_waveform  [ ?x f_x ]  [ ?y f_y ]  [ ?compression f_compression ]
     [ ?io s_measure ] )
     => f_compPoint/nil
```

Description

Performs an *n*th compression point measurement on a power waveform.

The `compression` function uses the power waveform to extrapolate a line of constant slope (dB/dB) according to a specified input or output power level. This line represents constant small-signal power gain (ideal gain). The function finds the point where the power waveform drops *n* dB from the constant slope line and returns either the X coordinate (input referred) value or the Y coordinate (output referred) value.

Arguments

o_waveform	Waveform object representing output power (in dBm) versus input power (in dBm).
f_x	The X coordinate value (in dBm) used to indicate the point on the output power waveform where the constant-slope power line begins. This point should be in the linear region of operation. Default value: Unless *f_y* is specified, defaults to the X coordinate of the first point of the *o_waveform* wave.
f_y	The Y coordinate value (in dBm) used to indicate the point on the output power waveform where the constant-slope power line begins. This point should be in the linear region of operation. Default value: Unless *f_x* is specified, defaults to the Y coordinate of the first point of the *o_waveform* wave.
f_compression	The delta (in dB) between the power waveform and the ideal gain line that marks the compression point Default value: `1`
s_measure	Symbol indicating whether the measurement is to be input referred (`'input`) or output referred (`'output`) Default value: `'input`

Value Returned

f_compPoint	Depending on the setting of *s_measure*, returns either input referred or output referred compression point.
`nil`	Returns `nil` and an error message otherwise.

Example

```
xloc = compression( wave ?x -25 ?compress 1)
yloc = compression( wave ?x -25 ?measure "Output")

; Each of following returns a compression measurement:

compression(dB20(harmonic(v("/Pif" ?result "pss_fd") 2)))

compression(dbm(harmonic(spectralPower(v("/Pif"
    ?result "pss_fd")/ 50.0
    v("/Pif" ?result "pss_fd")) 2)))

compression(dbm(harmonic(spectralPower(v("/Pif"
    ?result "pss_fd")/resultParam("rif:r"
    ?result "pss_td") v("/Pif"
    ?result "pss_fd")) 2)))

compression(dbm(harmonic(spectralPower(i("/rif/PLUS"
    ?result "pss_fd") v("/Pif" ?result "pss_fd")) 2))
    ?x -25 ?compress 0.1 ?measure "Output")
```

compressionVRI

```
compressionVRI( o_vport x_harm [?iport o_iport] [?rport f_rport]
     [?epoint f_epoint] [?gcomp f_gcomp] [?measure s_measure] )
     => o_waveform/n_number/nil
```

Description

Performs an *n*th compression point measurement on a power waveform.

Use this function to simplify the declaration of a compression measurement. This function extracts the specified harmonic from the input waveform(s), and uses `dBm(spectralPower((i or v/r),v))` to calculate a power waveform. The function passes this power curve and the remaining arguments to the `compression` function to complete the measurement.

The `compression` function uses the power waveform to extrapolate a line of constant slope (dB/dB) according to a specified input or output power level. This line represents constant small-signal power gain (ideal gain). The function finds the point where the power waveform drops *`n`* dB from the constant slope line and returns either the X coordinate (input referred) value or the Y coordinate (output referred) value.

Arguments

`o_vport`	Voltage across the output port. This argument must be a family of spectrum waveforms (1 point per harmonic) created by parametrically sweeping an input power (in dBm) of the circuit.
`x_harm`	Harmonic index of the voltage wave contained in *`o_vport`*. When *`o_iport`* is specified, also applies to a current waveform contained in *`o_iport`*.
`o_iport`	Current into the output port. This argument must be a family of spectrum waveforms (1 point per harmonic) created by parametrically sweeping an input power (in dBm) of the circuit. When specified, the output power is calculated using voltage and current. Default value: `nil`
`f_rport`	Resistance into the output port. When specified and *`o_iport`* is `nil`, the output power is calculated using voltage and resistance. Default value: `50`

f_epoint	The X coordinate value (in dBm) used to indicate the point on the output power waveform where the constant-slope power line begins. This point should be in the linear region of operation. Default value: the X coordinate of the first point of the *o_waveform* wave
f_gcomp	The delta (in dB) between the power waveform and the ideal gain line that marks the compression point. Default value: `1`
s_measure	Symbol indicating if measurement is to be input referred (`'input`) or output referred (`'output`). Default value: `'input`

Value Returned

o_waveform	Returns a waveform when *o_waveform1* is a family of waveforms.
f_number	Returns a number when *o_waveform1* is a waveform.
`nil`	Returns `nil` and an error message otherwise.

Example

```
Each of the following returns a compression measurement:
compressionVRI(v("/Pif" ?result "pss_fd") 2)
compressionVRI(v("/Pif" ?result "pss_fd") 2
    ?rport resultParam("rif:r" ?result "pss_td"))
compressionVRI(v("/Pif" ?result "pss_fd") 2
    ?iport i("/rif/PLUS" ?result "pss_fd") ?epoint -25
    ?gcomp 0.1 ?measure "Output")
```

compressionVRICurves

```
compressionVRICurves( o_vport x_harm [?iport o_iport] [?rport f_rport]
     [?epoint f_epoint] [?gcomp f_gcomp] )
     => o_waveform/nil
```

Description

Constructs the waveforms associated with an *n*th compression measurement.

Use this function to simplify the creation of waveforms associated with a compression measurement. This function extracts the specified harmonic from the input waveform(s), and uses `dBm(spectralPower((i or v/r),v))` to calculate a power waveform.

The `compressionVRICurves` function uses the power waveform to extrapolate a line of constant slope (1dB/1dB) according to a specified input or output power level. This line represents constant small-signal power gain (ideal gain). The function shifts the line down by *`n`* dB and returns it, along with the power waveform, as a family of waveforms.

This function only creates waveforms and neither performs a compression measurement nor includes labels with the waveforms. Use the `compression` or `compressionVRI` function for making measurements.

Arguments

`o_vport`	Voltage across the output port. This argument must be a family of spectrum waveforms (1 point per harmonic) created by parametrically sweeping an input power (in dBm) of the circuit.
`x_harm`	Harmonic index of the wave contained in *`o_vport`*. When *`o_iport`* is specified, also applies to a current waveform contained in *`o_iport`*.
`o_iport`	Current into the output port. This argument must be a family of spectrum waveforms (1 point per harmonic) created by parametrically sweeping an input power (in dBm) of the circuit. When specified, the output power is calculated using voltage and current Default value: `nil`
`f_rport`	Resistance into the output port. When specified and *`o_iport`* is `nil`, the output power is calculated using voltage and

	resistance. Default value: `50`
f_epoint	The X coordinate value (in dBm) used to indicate the point on the output power waveform where the constant-slope power line begins. This point should be in the linear region of operation. Default value: the X coordinate of the first point of the *o_waveform* wave
f_gcomp	The delta (in dB) between the power waveform and the ideal gain line that marks the compression point. Default value: `1`

Value Returned

o_waveform	Returns a family of waveforms containing the output power and tangent line.
`nil`	Returns `nil` and an error message otherwise.

Example

```
Each of following examples returns curves related to a compression measurement:
compressionVRICurves(v("/Pif" ?result "pss_fd") 2)
compressionVRICurves(v("/Pif" ?result "pss_fd") 2
    ?rport resultParam("rif:r" ?result "pss_td"))
compressionVRICurves(v("/Pif" ?result "pss_fd") 2
    ?iport i("/rif/PLUS" ?result "pss_fd") ?epoint -25
    ?gcomp 0.1)
```

complex

```
complex( f_real f_imaginary )
     => o_complex
```

Description

Creates a complex number of which the real part is equal to the real argument, and the imaginary part is equal to the imaginary argument.

Arguments

f_real	The real part of the complex number.
f_imaginary	The imaginary part of the complex number.

Value Returned

o_complex	Returns the complex number.

Example

```
complex( 1.0 2.0 )
=> complex( 1, 2 )
```

complexp

```
complexp( g_value )
     => t / nil
```

Description

Checks if an object is a complex number. The suffix `p` is added to the name of a function to indicate that it is a predicate function.

Arguments

g_value	A skill object.

Value Returned

`t`	Returns `t` when *g_value* is a complex number.
`nil`	Returns `nil` if there is an error.

Example

```
complexp( (complex 0 1) )
=> t
complexp( 1.0 )
=> nil
```

conjugate

```
conjugate( {o_waveform | n_x} )
     => o_waveform/n_y/nil
```

Description

Returns the conjugate of a waveform or number.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)
n_x	Complex or imaginary number.

Value Returned

o_waveform	Returns the conjugate of a waveform if the input argument is a waveform.
n_y	Returns the result of *n_x* being mirrored against the real axis (X axis) if the input argument is a number.
`nil`	Returns `nil` and an error message otherwise.

Example

For this example, assume that the first three statements are true for the `conjugate` function that follows them.

```
x=complex(-1 -2)
real(x) = -1.0
imag(x) = -2.0
conjugate(x) = complex(-1, 2)
```

Returns the conjugate of the input complex number.

convolve

```
convolve( o_waveform1 o_waveform2 n_from n_to t_type n_by )
    => o_waveform/n_number/nil
```

Description

Computes the convolution of two waveforms.

Convolution is defined as

$$\int_{from}^{to} f1(s)f2(t-s)ds$$

`f1` and `f2` are the functions defined by the first and second waveforms.

Note: The `convolve` function is numerically intensive and might take longer than the other functions to compute.

Arguments

o_waveform1	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)
o_waveform2	Additional waveform object.
n_from	Starting point (X-axis value) of the integration range.
n_to	Ending point (X-axis value) of the integration range.
t_type	Type of interpolation. Valid values: `"linear"` or `"log"`.
n_by	Increment.

Value Returned

o_waveform	Returns a waveform object representing the convolution if one of the input arguments is a waveform. Returns a family of waveforms if either of the input arguments is a family of waveforms.
n_number	Returns a value representing the convolution if both of the input arguments are numbers.
`nil`	Returns `nil` and an error message otherwise.

Example

```
sinWave = expr( n sin( n ) linRg( 0 20 0.01 ) )
triWave = artListToWaveform( '( ( -4, 0 ) ( -3, 1 ) ( -2, 0 ) ( -1, -1 ) ( 0, 0 )
( 1, 1 ) ( 2, 0 ) ( 3, -1 ) ( 4, 0 ) )
plot( convolve( sinWave triWave 0 10 "linear" 1 ) )
```

Gets the waveform from the convolution of the sine waveform and triangle waveform within the range of 0 to 10.

cPwrContour

```
cPwrContour( o_iwave o_vwave x_harm [?iwaveLoad o_iwaveLoad]
     [?vwaveLoad o_vwaveLoad] [?maxPower f_maxPower] [?minPower f_minPower]
     [?numCont x_numCont] [?refImp f_refImp] [?closeCont g_closeCont]
     [?modifier s_modifier] )
     => o_waveform/nil
```

Description

Constructs constant power contours for Z-Smith plotting. The trace of each contour correlates to reference reflection coefficients that all result in the same power level.

The *x_harm* harmonic is extracted from all the input waveforms. Power is calculated using the `spectralPower` function. The reference reflection coefficients are calculated using voltage, current, and a reference resistance.

Arguments

o_iwave	Current used to calculate power, expected to be a two-dimensional family of harmonic waveforms.
o_vwave	Voltage used to calculate power, expected to be a two-dimensional family of harmonic waveforms.
x_harm	Harmonic index of the waves contained in *o_iwave* and *o_vwave*.
o_iwaveLoad	Current used to calculate reflection coefficient, expected to be a two-dimensional family of harmonic waveforms. Default value: *o_iwave*
o_vwaveLoad	Voltage used to calculate reflection coefficient, expected to be a two-dimensional family of harmonic waveforms. Default value: *o_vwave*
f_maxPower	Maximum power magnitude value for contours. Default value: automatic
f_minPower	Minimum power magnitude value for contours. Default value: automatic

x_numCont	Total number of contours returned. Default value: `8`
f_refImp	Reference resistance used to calculate reflection coefficients. Default value: `50`
g_closeCont	Boolean indicating when to close the contours. When `nil`, largest segment of each contour is left open. Default value: `nil`
s_modifier	Symbol indicating the modifier function to apply to the calculated power. The modifier function can be any single argument OCEAN function such as `'db10` or `'dBm`. Default value: `'mag`

Value Returned

o_waveform	Returns a family of waveforms (contours) for Z-Smith plotting.
`nil`	Returns `nil` and an error message otherwise.

Example

The following example plots constant output power contours according to output:

```
cPwrContour(i("/I8/out" ?result "pss_fd") v("/net28"
    ?result "pss_fd")1)
```

The following example plots constant output power contours according to output reflection coefficients:

```
cPwrContour(i("/I8/out" ?result "pss_fd") v("/net28"
    ?result "pss_fd") 1 ?maxPower 0.002 ?minPower 0.001 ?numCont 9)
```

The following example plots constant input power contours according to output reflection coefficients:

```
cPwrContour(i("/C25/PLUS" ?result "pss_fd") v("/net30"
    ?result "pss_fd") 1 ?iwaveLoad i("/I8/out" ?result "pss_fd")
    ?vwaveLoad v("/net28" ?result "pss_fd") ?refImp 50.0
    ?numCont 9 ?modifier "mag")
```

cReflContour

```
cReflContour( o_iwave o_vwave x_harm [?iwaveLoad o_iwaveLoad]
     [?vwaveLoad o_vwaveLoad] [?maxRefl f_maxRefl] [?minRefl f_minRefl]
     [?numCont x_numCont] [?refImp f_refImp] [?closeCont g_closeCont] )
     => o_waveform/nil
```

Description

Constructs constant reflection coefficient magnitude contours for Z-Smith plotting. The trace of each contour correlates to reference reflection coefficients that all result in the same reflection coefficient magnitude.

The *x_harm* harmonic is extracted from all the input waveforms. Reflection coefficient magnitude is calculated using voltage, current, reference resistance, and the `mag` function. The reference reflection coefficients are calculated separately by using voltage, current, and a reference resistance.

Arguments

o_iwave	Current used to calculate reflection coefficient magnitude, expected to be a two-dimensional family of spectrum waveforms.
o_vwave	Voltage used to calculate reflection coefficient magnitude, expected to be a two-dimensional family of spectrum waveforms.
x_harm	Harmonic index of the waves contained in *o_iwave* and *o_vwave*.
o_iwaveLoad	Current used to calculate reference reflection coefficient, expected to be a two-dimensional family of harmonic waveforms. Default value: *o_iwave*
o_vwaveLoad	Voltage used to calculate reference reflection coefficient, expected to be a two-dimensional family of spectrum waveforms. Default value: *o_vwave*
f_maxRefl	Maximum reflection coefficient magnitude value for contours. Default value: automatic
f_minRefl	Minimum reflection coefficient magnitude value for contours. Default value: automatic

x_numCont Total number of contours returned.
Default value: `8`

f_refImp Reference resistance used to calculate reflection coefficients.
Default value: `50`

g_closeCont Boolean indicating when to close the contours. When `nil`, the largest segment of each contour is left open.
Default value: `nil`

Value Returned

o_waveform Returns a family of waveforms (contours) for Z-Smith plotting.

`nil` Returns `nil` and an error message otherwise.

Example

The following example plots constant output reflection coefficient contours according to output reflection coefficients:

```
cReflContour(i("/I8/out" ?result "pss_fd") v("/net28"
    ?result "pss_fd") 1)
```

The following example plots constant output reflection coefficient contours according to output reflection coefficients:

```
cReflContour(i("/I8/out" ?result "pss_fd") v("/net28"
    ?result "pss_fd") 1 ?maxRefl 0.7 ?minRefl 0.1 ?numCont 7)
```

The following example plots constant output reflection coefficient contours according to output reflection coefficients:

```
cReflContour(i("/C25/PLUS" ?result "pss_fd")
    v("/net30" ?result "pss_fd") 1
    ?iwaveLoad i("/I8/out" ?result "pss_fd")
    ?vwaveLoad v("/net28" ?result "pss_fd") ?refImp 50.0
    ?numCont 9)
```

cross

```
cross( o_waveform n_crossVal x_n s_crossType [g_multiple [s_Xname]] )
     => o_waveform/g_value/nil
```

Description

Computes the X-axis value at which a particular crossing of the specified edge type of the threshold value occurs.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)
n_crossVal	Y-axis value at which the corresponding values of X are calculated.
x_n	Number that specifies which X value to return. If *x_n* equals `1`, the first X value with a crossing is returned. If *x_n* equals `2`, the second X value with a crossing is returned, and so on. If you specify a negative integer for *x_n*, the X values with crossings are counted from right to left (from maximum to minimum). If you specify *x_n* equals to 0, it returns all occurrences of the crossing events.
s_crossType	Type of the crossing. Valid values: `'rising`, `'falling`, `'either`.
g_multiple	An optional boolean argument that takes the value `nil` by default. If set to `t`, the value specified for the `x_n` argument is ignored and the function returns all occurrences of the crossing event.
s_xName	An optional argument that is used only when *g_multiple* is set to `t`. It takes the value `time` by default. It controls the contents of the x vector of the waveform object returned by the function. Valid values: `'time`, `'cycle`

Value Returned

o_waveform	Returns a waveform if the input argument is a family of waveforms.
g_value	Returns the X-axis value of the crossing point if the input argument is a single waveform.
`nil`	Returns `nil` and an error message otherwise.

Example

```
cross( v( "/net9" ) 2.5 2 'rising )
```

Gets the time value (X axis) corresponding to specified voltage `"/net9"=2.5V` (Y axis) for the second rising edge.

```
cross( v( "/net9" ) 1.2 1 'either )
```

Gets the time value (X axis) corresponding to specified voltage `"/net9"=1.2V` (Y axis) for the first edge, which can be a rising or falling edge.

```
cross(VT("/out") 2.5 0 0 t "time") (s)
```

Returns multiple occurrences of crossing events specified against time-points at which each crossing event occurs.

```
cross(VT("/out") 2.5 0 0 t "cycle") (s)
```

Returns multiple occurrences of crossing events specified against cycle numbers, where a cycle number refers to the n'th occurrence of the crossing event in the input waveform.

db10

```
db10( {o_waveform | n_number} )
     => o_waveform/n_number/nil
```

Description

Returns 10 times the log10 of the specified waveform object or number. This function can also be written as dB10.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)
n_number	Number.

Value Returned

o_waveform	Returns a waveform object if the input argument is a waveform object or returns a family of waveforms if the input argument is a family of waveforms.
n_number	Returns a number if the input argument is a number.
`nil`	Returns `nil` and an error message otherwise.

Example

```
db10( ymax( v( "/net9" ) ) )
```

Returns a waveform representing `log10(ymax(v("/net9"))` multiplied by 10.

```
db10( 1000 )
=> 30.0
```

Gets the value `log10(1000)` multiplied by 10, or 30.

db20

```
db20( {o_waveform | n_number} )
     => o_waveform/n_number/nil
```

Description

Returns 20 times the log10 of the specified waveform object or number. This function can also be written as dB20.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)
n_number	Number.

Value Returned

o_waveform	Returns a waveform object if the input argument is a waveform object or returns a family of waveforms if the input argument is a family of waveforms.
n_number	Returns a number if the input argument is a number.
`nil`	Returns `nil` and an error message otherwise.

Example

```
db20( ymax( v( "/net9" ) ) )
```

Returns a waveform representing 20 times `log10(ymax(v("/net9")).`

```
db20( 1000 )
=> 60.0
```

Returns the value of 20 times `log10( 1000 )`, or 60.

dbm

```
dbm( {o_waveform | n_number} )
     => o_waveform/n_number/nil
```

Description

Returns 10 times the log10 of the specified waveform object plus 30. This function can also be written as dBm.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)
n_number	Number.

Value Returned

o_waveform	Returns a waveform object if the input argument is a waveform object or returns a family of waveforms if the input argument is a family of waveforms.
n_number	Returns a number if the input argument is a number.
`nil`	Returns `nil` and an error message otherwise.

Example

```
dbm( ymax( v( "/net9" ) ) )
```

Returns a waveform representing 10 times `log10(ymax(v("/net9"))` plus 30.

delay

```
delay( ?wf1 o_waveform1 ?value1 n_value1 ?edge1 s_edge1 ?nth1 x_nth1 ?td1 n_td1
     ?wf2 o_waveform2 ?value2 n_value2 ?edge2 s_edge2 ?nth2 x_nth2 {[?td2 n_td2]
     | [?td2r0 n_td2r0]} ?stop n_stop @rest args
     [g_histoDisplay][x_noOfHistoBins])
     => o_waveform/n_value/nil
```

Description

Calculates the delay between a trigger event and a target event.

The `delay` command computes the delay between two points using the `cross` command.

Arguments

o_waveform1	First waveform object.
n_value1	Value at which the crossing is significant for the first waveform object.
s_edge1	Type of the edge that must cross *n_value1*. Valid values: `'rising`, `'falling`, `'either`
x_nth1	Number that specifies which crossing is to be the trigger event. For example, if *x_nth1* is `2`, the trigger event is the second edge of the first waveform with the specified type that crosses *n_value1*.
n_td1	Time at which to start the delay measurement. The simulator begins looking for the trigger event, as defined by *o_waveform1*, *n_value1*, *t_edge1*, and *x_nth1*, only after the *n_td1* time is reached.
o_waveform2	Second waveform object.
n_value2	Value at which the crossing is significant for the second waveform.
s_edge2	Type of the edge for the second waveform. Valid values: `'rising`, `'falling`, `'either`

x_nth2	Number that specifies which crossing is to be the target event. For example, if *x_nth2* is `2`, the target event is the second edge of the second waveform with the specified type that crosses *n_value2*.
n_td2	Time to start observing the target event. *n_td2* is specified relative to the trigger event. This parameter cannot be specified at the same time as *n_td2r0*. The simulator begins looking for the target event, as defined by *o_waveform2*, *n_value2*, *t_edge2*, and *x_nth2*, only after the *n_td2* time is reached. If you specify neither *n_td2* nor *n_td2r0*, the simulator begins looking for the target event at `t = 0`.
n_td2r0	Time to start observing the target event, relative to `t = 0`. Only applicable if both *o_waveform1* and *o_waveform2* are specified. This parameter cannot be specified at the same time with *n_td2*. The simulator begins looking for the target event, as defined by *o_waveform2*, *n_value2*, *t_edge2*, and *x_nth2*, only after the *n_tdr0* time is reached. If you specify neither *n_td2* nor *n_td2r0*, the simulator begins looking for the target event at `t = 0`. `?td2 and ?td2r0 take precedence over other options.`
n_stop	Time to stop observing the target event.
args	Variable list of arguments passed to the `delay` function (as created from the Calculator UI). These variables also include support for multiple occurrences of the delay event.
g_histoDisplay	When set to t, returns a waveform that represents the statistical distribution of the riseTime data in the form of a histogram. The height of the bars (bins) in the histogram represents the frequency of the occurrence of values within the range of riseTime data.

	Valid values: `t` `nil` Default value: `nil`
x_noOfHistoBins	Denotes the number of bins represented in the histogram representation. Valid values: Any positive integer Default value: `nil`

Note: *g_histoDisplay* and *x_noOfHistoBins* are added for backward compatibility only. It will be deprecated in future releases. Use the histo function for plotting the histogram of the resulting function.

Value Returned

o_waveform	Returns a waveform representing the delay if the input argument is a family of waveforms.
n_value	Returns the delay value if the input argument is a single waveform.
`nil`	Returns `nil` and an error message otherwise.

Example

```
delay( ?wf1 wf1 ?value1 2.5 ?nth1 2 ?edge1 'either ?wf2 wf2 ?value2 2.5 ?nth2 1
?edge2 'falling )
```

Calculates the delay starting from the time when the second edge of the first waveform reaches the value of 2.5 to the time when the first falling edge of the second waveform crosses 2.5.

```
delay( ?td1 5 ?wf2 wf2 ?value2 2.5 ?nth2 1 ?edge2 'rising ?td2 5)
```

Calculates the delay starting when the time equals 5 seconds and stopping when the value of the second waveform reaches 2.5 on the first rising edge 5 seconds after the trigger.

```
delay( ?wf1 wf1 ?value1 2.5 ?nth1 1 ?edge1 'rising ?td1 5 ?wf2 wf2 ?value2 2.5 ?nth2
1 ?edge2 'rising ?td2 0)
```

Waits until after the time equals 5 seconds, and calculates the delay between the first and the second rising edges of *wf2* when the voltage values reach 2.5.

```
delay(VT("/out"), 2.5, 1, 'rising, VT("/in"), 2.5, 1, 'rising', 1, 1, t)
```

Computes the delay between the rising edges of `VT("/out")` and `VT("/in")` when the waveforms cross their respective threshold values (that is, `2.5`).

```
delay(VT("/out") 1.5 1 "rising" VT("/out") 1.5 2 "rising" 1 1 t "trigger") (s)
```

Returns multiple occurrences of delay specified against trigger time-points at which each delay event occurs.

```
delay(VT("/out") 1.5 1 "rising" VT("/out") 1.5 2 "rising" 1 1 t "target") (s)
```

Returns multiple occurrences of delay specified against target time-points at which each delay event occurs.

```
delay(VT("/out") 1.5 1 "rising" VT("/out") 1.5 2 "rising" 1 1 t "cycle") (s)
```

Returns multiple occurrences of delay specified against cycle numbers, where a cycle number refers to the n'th occurrence of the delay event in the input waveform.

deriv

```
deriv( o_waveform )
     => o_waveform/nil
```

Description

Computes the derivative of a waveform with respect to the X axis.

Note the following:

- After the second derivative, the results become inaccurate because the derivative is obtained numerically.
- Use the magnitude value instead of dB in frequency domain.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)

Value Returned

o_waveform	Returns a waveform object representing the derivative with respect to the X axis of the input waveform. Returns a family of waveforms if the input argument is a family of waveforms.
`nil`	Returns `nil` and an error message otherwise.

Example

```
plot( deriv( VT( "/net8" ) ) )
```

Plots the waveform representing the derivative of the voltage of `"/net8"`.

```
plot( deriv( mag(VF( "/OUT" ) ) ) )
```

Plots the waveform representing the derivative of the frequency of `"/OUT"`.

dft

```
dft( o_waveform n_from n_to x_num [t_windowName [n_param1]] )
     => o_waveform/nil
```

Description

Computes the discrete Fourier transform and fast Fourier transform of the input waveform.

The waveform is sampled at the following *n* timepoints:

```
from, from + deltaT, from + 2 * deltaT,…,
from + (N - 1) * deltaT
```

The output of `dft` is a frequency waveform, `W(f)`, which has `(N/2 + 1)` complex values—the DC term, the fundamental, and `(N/2 - 1)` harmonics.

Note: The last time point, `(from + (N - 1) * deltaT)`, is `(to - deltaT)` rather than `to`. The `dft` command assumes that `w(from)` equals `w(to)`.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)
n_from	Starting value for the `dft` computation.
n_to	Ending value for the `dft` computation.
x_num	Number of timepoints. If *x_num* is not a power of 2, it is forced to be the next higher power of 2.
t_windowName	Variable representing different methods for taking a `dft` computation. Valid values: `Rectangular`, `ExtCosBell`, `HalfCycleSine`, `Hanning` or `Cosine2`, `Triangle` or `Triangular`, `Half3CycleSine` or `HalfCycleSine3`, `Hamming`, `Cosine4`, `Parzen`, `Half6CycleSine` or `HalfCycleSine6`, `Blackman`, or `Kaiser`.

	For more information about *windowName*, see the information about Discrete Fourier Transform (dft) in the *Virtuoso Analog Design Environment L User Guide*.
n_param1	Smoothing parameter. Applies only if the *t_windowName* argument is set to `Kaiser`.

Value Returned

o_waveform	Returns a waveform representing the magnitude of the various harmonics for the specified range of frequencies. Returns a family of waveforms if the input argument is a family of waveforms.
`nil`	Returns `nil` and an error message otherwise.

Example

```
plot( dft( v( "/net8" ) 10u 20m 64 "rectangular" ) )
```

Computes the discrete Fourier transform, fast Fourier transform, of the waveform representing the voltage of `"/net8"`. The computation is done from `10u` to `20m` with 64 timepoints. The resulting waveform is plotted.

dftbb

```
dftbb( o_waveform1 o_waveform2 f_timeStart f_timeEnd x_num
     ?windowName t_windowName ?smooth x_smooth ?cohGain f_cohGain
     ?spectrumType s_spectrumType)
     => o_waveformComplex/nil
```

Description

Computes the discrete Fourier transform (fast Fourier transform) of a complex signal.

Arguments

o_waveform1 — Time domain waveform object with units of volts or amps.

o_waveform2 — Time domain waveform object with units of volts or amps.

f_timeStart — Start time for the spectral analysis interval. Use this parameter and `f_timeEnd` to exclude part of the interval. For example, you might set these values to discard initial transient data.

f_timeEnd — End time for the spectral analysis interval.

x_num — The number of time domain points to use. The maximum frequency in the Fourier analysis is directly proportionate to `x_num` and inversely proportional to the difference between `f_timeStart` and `f_timeEnd`.

t_windowName — The window to be used for applying the moving window FFT. Valid values: `Rectangular`, `ExtCosBell`, `HalfCycleSine`, `Hanning`, `Cosine2`, `Triangle` or `Triangular`, `Half3CycleSine` or `HalfCycleSine3`, `Hamming`, `Cosine3`, `Cosine4`, `Parzen`, `Half6CycleSine` or `HalfCycleSine6`, `Blackman`, or `Kaiser`. Default value: `Hanning`.

x_smooth — The Kaiser window smoothing parameter. If there are no requests, there is no smoothing.
Valid values: `0 <= x_smooth <= 15`
Default value: `1`

f_cohGain — A scaling parameter. A non-zero value scales the power spectral density by 1/(f_cohGain).
Valid values: `0 <= f_cohGain <= 1`. You can use `1` if you do

	not want the scaling parameter to be used. Default value: `1`
t_spectrumType	A string that can be either `singleSided` or `doubleSided`. When this option is single-sided, the resultant waveform is only on one side of the y axis starting from 0 to N-1. When it is double-sided, it is symmetric to the Y axis from -N/2 to (N/2) -1.

Value Returned

o_waveformComplex	The discrete Fourier transform for baseband signals of the two waveforms returned when the command is successful.
`nil`	Returns `nil` and an error message otherwise.

Example

```
dftbb(VT("/net32") VT("/net11") , 0, 16m, 12000, ?windowName 'Hanning,?smooth 1,
?cohGain 1, ?spectrumType "SingleSided")
```

dnl

```
dnl( o_dacSignal o_sample|o_pointList|n_interval [?mode t_mode] [?threshold
     n_threshold] [?crossType t_crossType] [?delay f_delay] [?method t_method]
     [?units x_units] [?nbsamples n_nbsamples] )
     => n_dnl/nil
```

Description

Computes the differential non-linearity of a transient simple or parametric waveform.

Arguments

o_dacSignal	Waveform for which the differential non-linearity is to be calculated.
o_sample	Waveform used to obtain the points for sampling the *dacSignal*. These are the points at which the waveform crosses the threshold while either `rising` or `falling` (defined by the *crossType* argument) with the *delay* added to them.
n_pointList	List of domain values at which the sample points are obtained from the *dacSignal*.
n_interval	The sampling interval.
t_mode	The mode for calculating the threshold. Valid values: `auto` and `user`. Default value: `auto`. If set to `user`, an `n_threshold` value needs to be provided. If set to `auto`, `n_threshold` is calculated internally.
n_threshold	The threshold value against which the differential non-linearity is to be calculated. It needs to be specified only when the *mode* is selected as `user`.
t_crossType	The points at which the curves of the waveform intersect with the threshold. While intersecting, the curve may be either rising or falling. Valid values: `rising` and `falling`, respectively. Default `crossType` is `rising`.

f_delay	The delay time after which the sampling begins. Valid values: Any valid time value. Default value: `0`.
t_method	The method to be used for calculation. Valid values: `end` (end-to-end) and `fit` (straight line). Default value: `end`.
x_units	Unit for expressing the output waveform. Valid values: `abs` (absolute) and `lsb` (multiples of least significant bit). Default value: `abs`.
n_nbsamples	Number of samples used for calculating the non-linearity. If not specified, the samples are taken against the entire data window.

Note: For each of the three ways in which the sample points can be specified, only a few of the other optional arguments are meaningful, as indicated below:

- For *o_sample*, the arguments *t_mode*, *n_threshold*, *t_crossType*, *f_delay*, *t_method*, and *x_units* are meaningful.
- For *n_pointList*, the arguments *t_method* and *x_units* are meaningful.
- For *n_interval*, the arguments *t_method*, *x_units*, and *n_nbsamples* are meaningful.

Value Returned

n_dnl	Returns the differential waveform.
`nil`	Returns `nil` and an error message otherwise.

Example

```
dnl( wave1 wave2 ?crossType "rising" ?delay 0.4 )
=> srrWave:175051544
```

Returns the differential non-linearity for `wave1` by taking the points at which `wave2` crosses the internally calculated threshold while `rising` as the sample points and adding a delay of `0.4` to them.

dutyCycle

```
dutyCycle( o_waveform [?mode t_mode] [?threshold n_threshold] [?xName t_xName]
     [?outputType t_outputType] )
     => o_waveform/f_average/nil
```

Description

Computes the duty cycle for a given waveform as a function of time or cycle.

Note: Duty cycle is the ratio of the time for which the signal remains 'high' and the time period of the signal.

Arguments

o_waveform	Waveform, expression, or a family of waveforms.
t_mode	The mode for calculating the threshold. Valid values: `auto` and `user`. Default value: `auto`. If set to `user`, an `n_threshold` value needs to be provided. If set to `auto`, `n_threshold` is calculated internally.
n_threshold	The threshold value. It needs to be specified only when the *mode* is selected as `user`.
t_xName	The X-axis of the output waveform. Valid values: `time` and `cycle`. Default value: `time`.
outputType	Type of output. Valid values: `average` and `plot`. If set to `average`, the output is an average value. If set to `plot`, the output is a waveform. In both the cases, the output is expressed in terms of a percentage. Default value: `plot`.

Value Returned

o_waveform	Returns a waveform that represents duty cycle as a function of time.
f_average	Returns the average duty cycle value as a percentage.
`nil`	Returns `nil` if the duty cycle cannot be calculated.

Example

```
dutyCycle( wave1 )
=> srrWave:175051552
```

Returns the duty cycle as a function of time for the wave `wave1`.

```
dutyCycle( wave1 ?outputType "average" )
=> 52.1066
```

Returns the average (in percentage) of the duty cycle values for the wave `wave1`.

evmQAM

```
evmQAM( o_waveformI o_waveformQ n_tDelay n_samplingT x_levels g_normalize )
    => o_waveform/nil
```

Description

Processes the I and Q waveform outputs from the transient simulation run to calculate the Error Vector Magnitude (EVM) for multi-mode modulations. The function plots the I versus Q scatterplot. EVM is a useful measurement to describe the overall signal amplitude and phase modulated signal quality. It is based on a statistical error distribution normalized from an ideal digital modulation. Quadrature Amplitude Modulation (QAM) is a typical modulation scheme where EVM is useful. The EVM is calculated by detecting the I and Q signal levels corresponding to the four possible I and Q symbol combinations and calculating the difference between the actual signal level and the ideal signal level.

Note: This function is not supported for families of waveforms.

Arguments

o_waveformI	The waveform for the I signal.
o_waveformQ	The waveform for the Q signal.
n_tDelay	The start time (a numerical value) for the first valid symbol. This can be obtained from the Waveform Viewer window by recording the time of the first minimum or first maximum (whichever is earlier) on the selected signal stream.
n_samplingT	A sampling time (a numerical value) for the symbol. Each period is represented by a data rate. The data rate at the output is determined by the particular modulation scheme being used.
x_levels	The modulation levels. Valid values: `4`, `16`, `64`, `256` Default value: `4`
g_normalize	An option to see the scatter plot normalized to the ideal values `+1` and `-1` (for example, when superimposing scatter plots from different stages in the signal flow, where the levels may be quite different but you want to see relative degradation or improvement in the scatter). This option does not affect the calculation of the EVM number.

Valid values: `nil`, `t`
Default value: `t`

Value Returned

o_waveform — Returns a waveform object representing the EVM value computed from the input waveforms.

`nil` — Returns `nil` and an error message if the function is unsuccessful.

Example

```
evmQAM( v("samp_out_Q"), v("samp_out_I") 1.5u, 181.81n, 4, t )
```

Calculates the EVM value for the modulation level `4` in normalized form.

evmQpsk

```
evmQpsk( o_waveform1 o_waveform2 n_tDelay n_sampling g_autoLevelDetect
     n_voltage n_offset g_normalize )
     => o_waveform/nil
```

Description

Processes the I and Q waveform outputs from the transient simulation run to calculate the Error Vector Magnitude (EVM) and plot the I versus Q scatterplot. EVM is a useful measurement to describe the overall signal amplitude and phase modulated signal quality. It is based on a statistical error distribution normalized from an ideal digital modulation. Quadrature Phase Shift Keying (QPSK) is a typical modulation scheme where EVM is useful. The EVM is calculated by detecting the I and Q signal levels corresponding to the four possible I and Q symbol combinations and calculating the difference between the actual signal level and the ideal signal level.

Note: This function is not supported for families of waveforms.

Arguments

o_waveform1	The waveform for the I signal.
o_waveform2	The waveform for the Q signal.
n_tDelay	The start time for the first valid symbol. This can be obtained from the Waveform Viewer window by recording the time of the first minimum or first maximum (whichever is earlier) on the selected signal stream.
n_sampling	A period for the symbol. Each period is represented by a data rate. The data rate at the output is determined by the particular modulation scheme being used.
g_autoLevelDetect	An option to indicate that you want the amplitude (*n_voltage*) and DC offset (*n_offset*) to be automatically calculated. Amplitude is calculated by averaging the rectified voltage level of the signal streams and DC offset by averaging the sum of an equal number of positive and negative symbols in each signal stream. These values are used to determine the EVM value. If this value is set to `nil`, you must specify values for *n_voltage* and *n_offset*.

	Valid values: `'nil, 't` Default value: `'t`
n_voltage	The amplitude of the signal.
n_offset	The DC offset value.
g_normalize	An option to see the scatter plot normalized to the ideal values `+1` and `-1` (for example, when superimposing scatter plots from different stages in the signal flow, where the levels may be quite different but the you want to see relative degradation or improvement in the scatter). This option does not affect the calculation of the EVM number. Valid values: `nil,t` Default value: `nil`

Value Returned

o_waveform	Returns a waveform object representing the EVM value computed from input waveforms.
`nil`	Returns `nil` and an error message if the function is unsuccessful.

Example

```
evmQpsk( v("samp_out_Q"), v("samp_out_I") 1.5u, 181.81n, t, nil, nil, nil)
```

Calculates the EVM value when *g_autoLevelDetect* is set to `t`. In this case, no values are specified for *n_voltage* and *n_offset*.

```
evmQpsk( v("samp_out_Q"), v("samp_out_I") 1.5u, 181.81n, nil, 1.3, 0, nil)
```

Calculates the EVM value when *g_autoLevelDetect* is set to `nil`. In this case, values are specified for *n_voltage* and *n_offset*.

eyeDiagram

```
eyeDiagram ( o_waveform n_start n_stop n_period ?advOptions t_advOptions )
     => o_waveform/nil
```

Description

Returns an eye-diagram plot of the input waveform signal. It returns the waveform object of the eye-diagram plot. Using an advanced option, the function also calculates the maximum vertical and horizontal opening of the eye formed when the input waveform is folded by the specified period to form the eye.

Arguments

o_waveform	Input waveform signal.
n_start	The X-axis start value from where the eye-diagram plot is to begin.
n_stop	The X-axis stop value where the eye-diagram plot is to terminate.
n_period	The period after which the waveform is to be folded to form the eye.
t_advOptions	The option to specify whether the vertical or horizontal opening of the eye is to be calculated. Valid values: `vertical`, `horizontal` Default value: `nil`

Note: If *t_advOptions* is specified, the function approximates vertical eye height and horizontal eye width to assume the symmetry of the eye. The function returns the most optimum results for single eye scenarios.

Value Returned

o_waveform	Returns a waveform object representing the eye-diagram plot of the input waveform
`nil`	Returns `nil` and an error message otherwise

Example

```
eyeDiagram( v("/out" ) 0n 500n 12.5n )
```

Returns a waveform that represents an eye-diagram plot.

```
eyeDiagram( v("/out" ) 0n 500n 12.5n ?advOptions “vertical” )
```

Calculates the maximum vertical opening of the eye that is formed when the input waveform is folded after `12.5n`

```
eyeDiagram( v("/out" ) 0n 500n 12.5n ?advOptions “horizontal” )
```

Calculates the maximum horizontal opening of the eye that is formed when the input waveform is folded after `12.5n`

eyeMeasurement

```
eyeMeasurement(eyeDiagram ( o_waveform n_start n_stop n_period ) n_threshold
     n_sample g_xTypePercent0 n_startX0 n_startY0 g_yTypePercent0 n_endX0
     n_endY0 g_xTypePercent1 n_startX1 n_startY1 g_yTypePercent1 n_endX1
     n_endY1 n_noofBins t_measure)
     => o_waveform/nil
```

Description

Evaluates the measurements for the eye diagram plot.

Arguments

o_waveform	The eye diagram waveform.
n_start	The X-axis start value from where the eye diagram plot is to begin.
n_stop	The X-axis stop value where the eye diagram plot is to terminate.
n_period	The period after which the waveform is to be folded to form the eye.
n_threshold	The Y-axis level (for example voltage) that represents the switching threshold of the signal, typically half the signal range. This is used to compute statistical information about the threshold.
n_sample	The time interval after which the signals are divided in the eye diagram plot. If this field is left blank, the data within the level 1 and level 0 regions are used to analyze the amplitude variation of the signal. This means there is some sensitivity to the actual spacing between the data points in the signal, which is caused by the variable time steps in the simulator. If the points are clustered in the curve portion, the distribution can be skewed. To perform the analysis, the sampling interval you specify in this field is divided into even time points.
g_xTypePercent0	Level0 X-range specified whether specified in "%". If the value is `t`, it signifies the "%" value and if the value is nil, it signifies the absolute value.

n_startX0	Level0 X-range start value.
n_startY0	Level0 Y-range start value.
g_yTypePercent0	Level0 Y-range specified whether specified in "%". If the value is `t`, it signifies the "%" value and if the value is nil, it signifies the absolute value.
n_endX0	Level0 X-range end value.
n_endY0	Level0 Y-range end value.
g_xTypePercent1	Level1 X-range specified whether specified in "%". If the value is `t`, it signifies the "%" value and if the value is nil, it signifies the absolute value.
n_startX1	Level1 X-range start value.
n_startY1	Level1 Y-range start value.
g_yTypePercent1	Level1 Y-range specified whether specified in "%". If the value is `t`, it signifies the "%" value and if the value is nil, it signifies the absolute value.
n_endX1	Level1 X-range end value.
n_endY1	Level1 Y-range end value.
n_noofBins	Number of signal bins to be displayed in the eye diagram plot. These signals bins are used to form the horizontal (threshold crossing times) and vertical (amplitude variation) histograms.
t_measure	Computes one of the measurement values described below: *Level0 Mean*—Mean of the Y-values within the level0 region. *Level0 Stdde*v—Standard deviation of the Y-values within the level0 region. *Level1 Mean*—Mean of the Y-values within the level1 region. *Level1 Stddev*—Standard deviation of the Y-values within the level1 region.

Eye amplitude—Mean to mean amplitude of the eye, computed as: `Meanlevel1 - Meanlevel0`

Eye height—Vertical opening of the eye, computed as: `(Meanlevel1 - 3¿level1) - (Meanlevel0 - 3¿level0)`

Eye signalToNoise—Signal to noise ratio of the eye, computed as: `(Meanlevel1 - Meanlevel0) / (¿level1 + ¿level0)`

Threshold crossing stddev—Threshold crossing standard deviation is computed only when there is a single transition region in the eye diagram because it is analyzed over the entire period.

Threshold crossing average—This is computed over the entire period.

Eye width—Represents the opening of the eye in the X direction. It is computed as:

```
(Meantransition2 - 3¿transition2) -
(Meantransition1 - 3¿transition1)
```

Eye Rise Time—Two thresholds taken at the 20% and 80% points between the level0 mean and level1 mean. At each of these two thresholds, a horizontal histogram is computed, which is an analysis of the crossing points of these two thresholds, and the resulting rise time is the difference in the mean crossing point at each of these two thresholds.

Eye Fall Time— Signal measured between the percent high and percent low of the difference between the initial and final value.

Value Returned

`o_waveform` Returns the computed scalar value or a waveform for the specific measure that was passed.

`nil` Returns `nil` and an error message otherwise.

Example

The following command computes the threshold crossing average for the eye diagram for signal /`von` from `10n` to `40n` with a period of `194p`:

```
eyeMeasurement(eyeDiagram(v("/von" ?result "tran") 1e-08 4e-08
1.94e-10) 1e-08 4e-08 1.94e-10 0.65 nil nil 0.0 0.0 t 1.94e-10 50.0
t 40.0 50.0 t 60.0 100.0 10.0 "thresholdCrossingAverage")
```

edgeTriggeredEyeDiagram

```
edgeTriggeredEyeDiagram(o_waveform n_start n_stop o_triggerWave n_threshold
     s_edgeType n_triggerOffset ?intensityPlot g_intensityPlot)
     => o_waveform/nil
```

Desription

Returns a signal triggered at the beginning of the eye diagram instead of a fixed period.

Arguments

o_waveform	The eye diagram waveform.
n_start	The X-axis start value from where the eye diagram plot is to begin.
n_stop	The X-axis stop value where the eye diagram plot is to terminate.
o_triggerWave	The waveform that is used for triggering the eye diagram.
n_threshold	The Y-axis value of trigger wave at which the corresponding cross points of the trigger wave are calculated.
s_edgeType	Type of the crossing. Valid values: `rising`, `falling`, `either`.
n_triggerOffset	The value by which the trigger wave should be l-shifted to align with the input waveform signal.
g_intensityPlot	Controls the intensity based plotting of the eye diagram.

Value Returned

o_waveform	Returns the computed scalar value or a waveform for the specific measure that was passed.
`nil`	Returns `nil` and an error message otherwise.

Examples

In the following example `VT("/out")` is an input waveform for which eye diagram is to be determined from `0n` to `10n`. The period to wrap or fold the eye diagram is determined by the cross points of the trigger waveform `VT("/clk")` at the given threshold.

```
edgeTriggeredEyeDiagram(VT("/out") 0n 10n VT("/clk") 2.5 "either" 0n)
```

The above function returns a waveform with the relevant edge Trigger eye diagram attributes set so that when plotted the edge trigger eye diagram is displayed.

The following example shows that an offset of `1n` signifies that `VT("/clk")` is be l-shifted by `1n`, `lshift(VT("/clk") 1n)`, before determining the cross points. Also. intensity-based plotting is turned on.

```
edgeTriggeredEyeDiagram(VT("/out") 0n 10n VT("/clk") 2.5 "rising" 1n
?intensityPlot t)
```

flip

```
flip( o_waveform )
    => o_waveform/nil
```

Description

Returns a waveform with the X vector values negated.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)

Value Returned

o_waveform	Returns a waveform object representing the input waveform mirrored about its Y axis. Returns a family of waveforms if the input argument is a family of waveforms.
`nil`	Returns `nil` and an error message otherwise.

Example

```
plot( flip( v("/net4") ) )
```

Plots the waveform for the voltage of `"/net4"` with the X vector values negated.

fourEval

```
fourEval( o_waveform n_from n_to n_by [?g_baseBand] )
     => o_waveform/nil
```

Description

Evaluates the Fourier series represented by an expression.

This function is an inverse Fourier transformation and thus the inverse of the `dft` command. The `fourEval` function transforms the expression from the frequency domain to the time domain.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)
n_from	Starting point on the X axis at which to start the evaluation.
n_to	Increment.
n_by	Ending point on the X axis.
g_baseBand	Accepts boolean values `t` or `nil`. The default value is `nil`. When set to `t`, the function evaluates the baseband version of the inverse of the `dft` function by converting the unsymmetrical spectrum to a symmetrical one.

Value Returned

o_waveform	Returns a waveform object representing the inverse Fourier transformation of the input waveform. Returns a family of waveforms if the input argument is a family of waveforms. Returns the baseband version of the inverse of the `dft` function if `baseBand` is set to `t`.
`nil`	Returns `nil` and an error message otherwise.

Example

```
plot( fourEval( v( "/net3" ) 1k 10k 10 )
```

Plots the waveform representing the inverse Fourier transformation of the voltage of `"/net3"` from `1k` to `10k`.

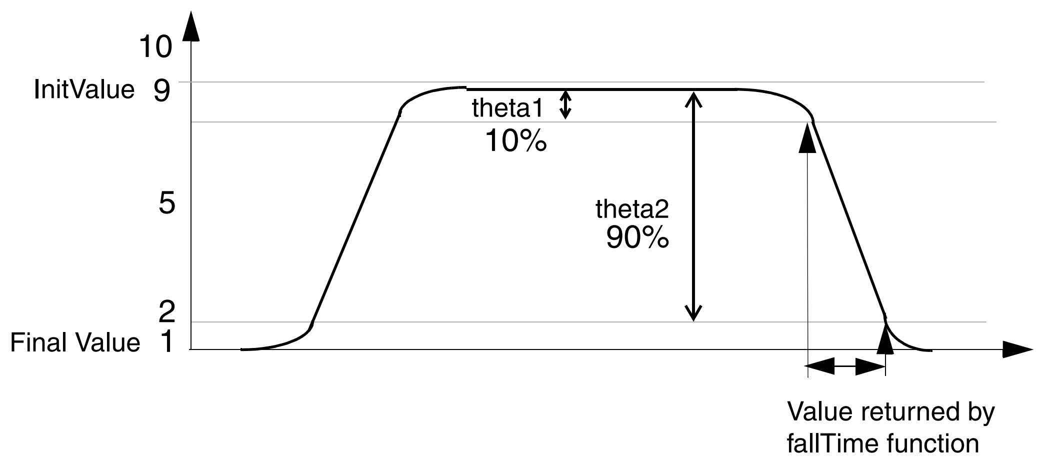
fallTime

```
fallTime( o_waveform n_initVal g_initType n_finalVal g_finalType n_theta1
     n_theta2 [g_multiple [s_Xname][g_histoDisplay][x_noOfHistoBins] ] )
     => o_waveform/n_value/nil
```

Description

Returns the fall time measured between `theta1` (percent high) to `theta2` (percent low) of the difference between the initial value and the final value.

The `fallTime` function can also be used to compute the rise time if `initVal` is lower than `finalVal`.



Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:XXXXX`.)
n_initVal	Initial value at which to start the computation.
g_initType	Specifies how *n_initVal* functions. Valid values: a non-nil value specifies that the initial value is taken to be the value of the waveform, interpolated at *n_initVal*, and the waveform is clipped from below as

follows:
o_waveform = clip(*o_waveform g_initVal* nil)

where `nil` specifies that *n_initVal* is defined by the X value entered. (The command gets the Y value for the specified X value and uses that value for *n_initVal*.)

n_finalVal	Final value at which to end the computation.
g_finalType	Specifies how the *n_finalVal* argument functions. Valid values: a non-nil value specifies that the final value is taken to be the value of the waveform, interpolated at *n_finalVal*, and the waveform is clipped from above, as follows: *o_waveform* = clip(*o_waveform* nil *n_finalVal*) where `nil` specifies that the *n_finalVal* argument is defined by the X value entered. (The command gets the Y value for the specified X value and uses that value for *n_finalVal*.)
n_theta1	Percent high.
n_theta2	Percent low.
g_multiple	An optional boolean argument that takes the value `nil` by default. If set to `t`, the function returns multiple occurrences of the fallTime event.
s_xName	An optional argument that is used only when *g_multiple* is set to `t`. It takes the value `time` by default. It controls the contents of the x vector of the waveform object returned by the function. Valid values: `'time`, `'cycle`
g_histoDisplay	When set to t, returns a waveform that represents the statistical distribution of the fallTime data in the form of a histogram. The height of the bars (bins) in the histogram represents the frequency of the occurrence of values within the range of fallTime data. Valid values: `t nil` Default value: `nil`
x_noOfHistoBins	Denotes the number of bins represented in the histogram representation.

Valid values: Any positive integer
Default value: `nil`

Note: *g_histoDisplay* and *x_noOfHistoBins* are added for backward compatibility only. It will be deprecated in future releases. Use the histo function for plotting the histogram of the resulting function.

Value Returned

o_waveform	Returns a waveform representing the fall time for a family of waveforms if the input argument is a family of waveforms or if *g_multiple* is set to `t`.
n_value	Returns a value for the fall time if the input is a single waveform.
`nil`	Returns `nil` and an error message otherwise.

Example

```
fallTime( v( "/net8" ) 9 nil 1 nil 10 90 )
```

Computes the fall time for the waveform representing the voltage of `"/net8"` from `9` to 1.

freq

```
freq( o_waveform t_crossType [?threshold n_threshold] [?mode t_mode]
     [?xName xName][g_histoDisplay][x_noOfHistoBins] )
     => o_outputWave/nil
```

Description

Computes the frequency of the input waveform(s) as a function of time or cycle.

Arguments

o_waveform	Waveform, expression, or a family of waveforms.
t_crossType	The points at which the curves of the waveform intersect with the threshold. While intersecting, the curve may be either rising or falling. For the `freq` function, you may specify the frequency to be calculated against either the rising points or the falling points by setting `crossType` to `rising` or `falling`, respectively. The default `crossType` is `rising`.
n_threshold	The threshold value against which the frequency is to be calculated. This needs to be specified only when the `mode` selected is `user`.
t_mode	The mode for calculating the threshold. This is `auto`, by default, in which case `n_threshold` is calculated internally. It can alternatively be set to `user`, in which case, an `n_threshold` value needs to be provided.
t_xName	The X-axis of the output waveform. The default value is `time` but `cycle` is also a valid value.
g_histoDisplay	When set to t, returns a waveform that represents the statistical distribution of the riseTime data in the form of a histogram. The height of the bars (bins) in the histogram represents the frequency of the occurrence of values within the range of riseTime data. Valid values: `t nil` Default value: `nil`
x_noOfHistoBins	Denotes the number of bins represented in the histogram representation.

Valid values: Any positive integer
Default value: `nil`

Note: *g_histoDisplay* and *x_noOfHistoBins* are added for backward compatibility only. It will be deprecated in future releases. Use the histo function for plotting the histogram of the resulting function.

Value Returned

o_outputWave	Returns the frequency as a function of time or cycle.
`nil`	Returns `nil` if the frequency cannot be calculated.

Example

```
freq( wave1 “rising” ?mode “user” ?threshold 18.5 ?xName “cycle” )
=> srrWave: 170938688
```

Returns the frequency for `wave1` with the `threshold` at `18.5` against `cycle` on the x-axis.

freq_jitter

```
freq_jitter( o_waveform t_crossType [?mode t_mode] [?threshold n_threshold]
     [binSize n_binSize] [?xName t_xName] [?outputType t_outputType] )
     => o_waveform/f_val/nil
```

Description

Calculates the frequency jitter.

Arguments

o_waveform	Waveform, expression, or a family of waveforms.
t_crossType	The points at which the curves of the waveform intersect with the threshold. While intersecting, the curve may be either rising or falling. Valid values: `rising` and `falling`. Default value: `rising`.
t_mode	The mode for calculating the threshold. Valid values: `auto` and `user`. If set to `user`, an `n_threshold` value needs to be provided. If set to `auto`, `n_threshold` is calculated internally. Default value: `auto`.
n_threshold	The threshold value against which the frequency is to be calculated. It needs to be specified only when the `mode` selected is `user`.
n_binSize	The width of the moving average window.The deviation of value at the particular point from the average of this window is the jitter.
t_xName	The X-axis of the output waveform. Valid values: `time` and `cycle`. Default value: `time`.
t_outputType	Type of output. Valid values: `sd` and `plot`. If set to `sd`, the output is a standard deviation jitter. If set to `plot`, the output is a waveform. Default value: `plot`.

Value Returned

o_waveform	Returns the frequency jitter values as a function of time or cycle when the *outputType* is set to `plot`.
f_val	Returns the standard deviation value when the *outputType* is set to `sd`.
`nil`	Returns `nil` otherwise.

Example

```
freq_jitter( wave1 “rising” ?mode “user” ?threshold 1 ?binSize 2 ?xName “cycle”
?outputType “sd” )
=> 0.1338585
```

Returns the standard deviation for the frequency jitter of `wave1` with the threshold of 1 against the cycle on the x-axis.

frequency

```
frequency( o_waveform )
     => o_waveform/n_value/nil
```

Description

Computes the reciprocal of the average time between two successive midpoint crossings of the rising waveform.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)

Value Returned

o_waveform	Returns a waveform representing the frequency of a family of waveforms if the input argument is a family of waveforms.
n_value	Returns a number representing the frequency of the specified waveform.
`nil`	Returns `nil` and an error message otherwise.

Example

```
frequency( v( "/net12" ) )
```

Returns the frequency of `"/net12"`.

ga

```
ga( o_s11 o_s12 o_s21 o_s22 [ ?gs n_gs] )
     => o_waveform/nil
```

Description

Returns the available gain in terms of the supplied parameters and the optional source reflection coefficient (Gs).

Arguments

o_s11	Waveform object representing s11.
o_s12	Waveform object representing s12.
o_s21	Waveform object representing s21.
o_s22	Waveform object representing s22.
n_gs	Source reflection coefficient. Default value: `0`

Value Returned

o_waveform	Waveform object representing the available gain.
`nil`	Returns `nil` and an error message otherwise.

Example

```
s11 = sp(1 1)
s12 = sp(1 2)
s21 = sp(2 1)
s22 = sp(2 2)
plot(ga(s11 s12 s21 s22))
```

gac

```
gac( o_s11 o_s12 o_s21 o_s22 g_level g_frequency )
     => o_waveform/nil
```

Description

Computes the available gain circles.

The *g* data type on *g_level* and *g_frequency* allows either the level or the frequency to be swept while the other remains fixed.

Arguments

o_s11	Waveform object representing s11.
o_s12	Waveform object representing s12.
o_s21	Waveform object representing s21.
o_s22	Waveform object representing s22.
g_level	Level in dB. It can be specified as a scalar or a vector. If it is specified as a vector, the level is swept. The `linRg` function can be called to generate a linear range. For example, `linRg( -30 30 5 )` is the same as `list(-30 -25 -20 -15 -10 -5 0 5 10 15 20 25 30)` and the *g_level* argument can be specified as either of the above. In that case, an available gain circle is calculated at each one of the 13 levels.
g_frequency	Frequency, which can be specified as a scalar or a linear range. If it is specified as a linear range, the frequency is swept. The linear range is specified as a list with three values: the start of the range, the end of the range, and the increment. For example, `list(100M 1G 100M)` specifies a linear range with the following values: { 100M, 200M, 300M, 400M, 500M, 600M, 700M, 800M, 900M, 1G } In that case, an available gain circle is calculated at each one of the 10 frequencies.

Value Returned

o_waveform	Waveform object representing the available gain circles.
nil	Returns `nil` and an error message otherwise.

Example

```
s11 = sp(1 1 ?result "sp")
s12 = sp(1 2 ?result "sp")
s21 = sp(2 1 ?result "sp")
s22 = sp(2 2 ?result "sp")
plot(gac(s11 s12 s21 s22 linRg(-30 30 5) 900M))
```

gainBwProd

```
gainBwProd( o_waveform )
     => o_waveform/n_value/nil
```

Description

Calculates the gain-bandwidth product of a waveform representing the frequency response of interest over a sufficiently large frequency range.

Returns the product of the zero-frequency-gain and 3dB-gain-frequency.
.

$$\text{gainBwProd (gain)} = A_o * f2$$

The gain-bandwidth product is calculated as the product of the DC gain A_o and the critical frequency f2. The critical frequency f2 is the smallest frequency for which the gain equals $1/\sqrt{2}$ times the DC gain A_o.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*`XXXXX`*.)

Value Returned

o_waveform	Returns a waveform representing the gain-bandwidth product for a family of waveforms if the input argument is a family of waveforms.
n_value	Returns a value for the gain-bandwidth product for the specified waveform.
`nil`	Returns `nil` and an error message otherwise.

Example

```
gainBwProd( v( "/OUT" ) )
```

Returns the gain-bandwidth product for the waveform representing the voltage of the `"/OUT"` net.

gainMargin

```
gainMargin( o_waveform [g_stable])
     => o_waveform/n_value/nil
```

Description

Computes the gain margin of the loop gain of an amplifier.

The first argument is a waveform representing the loop gain of interest over a sufficiently large frequency range. This command returns the dB value of the waveform when its phase crosses negative pi.

```
gainMargin( gain ) = 20 * log10( value( gain f0 ) )
```

The gain margin is calculated as the magnitude of the gain in dB at f0. The frequency f0 is the lowest frequency in which the phase of the gain provided is -180 degrees. For stability, the gain margin will be negative when g_stable is set to nil. If g_stable value is set to t, then a stable design will have a positive value.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)
g_stable	Boolean optional value that takes the value `nil` by default.

Value Returned

o_waveform	Returns a waveform representing the gain margin for a family of waveforms if the input argument is a family of waveforms.
n_value	Returns the value for the gain margin of the specified waveform.
`nil`	Returns `nil` and an error message otherwise.

Example

```
gainMargin( v( "/OUT" ) ) = -9.234
gainMargin( v( "/OUT" ) nil ) = -9.234
gainMargin( v( "/OUT" ) t ) =  9.234
```

gmax

```
gmax( o_s11 o_s12 o_s21 o_s22 )
     => o_waveform/nil
```

Description

Returns the maximum power gain in terms of the supplied parameters.

Arguments

o_s11	Waveform object representing s11.
o_s12	Waveform object representing s12.
o_s21	Waveform object representing s21.
o_s22	Waveform object representing s22.

Value Returned

o_waveform	Load reflection coefficient.
nil	Returns `nil` and an error message otherwise.

Example

```
s11 = sp(1 1)
s12 = sp(1 2)
s21 = sp(2 1)
s22 = sp(2 2)
plot(gmax(s11 s12 s21 s22))
```

gmin

```
gmin( o_Gopt o_Bopt f_zref )
     => o_gminWave/nil
```

Description

Returns the optimum noise reflection coefficient in terms of *o_Gopt*, *o_Bopt*, and *f_zref*.

`gmin` is returned as follows:

```
yOpt = o_Gopt + (complex 0 1) * o_Bopt
return ( 1 / f_zref(1) - yOpt ) / ( 1 / f_zref(1) + yOpt )
```

Arguments

o_Gopt	Waveform object representing the optimum source conductance.
o_Bopt	Waveform object representing the optimum source susceptance.
f_zref	Reference impedance.

Value Returned

o_gminWave	Waveform object representing the optimum noise reflection coefficient.
`nil`	Returns `nil` and an error message otherwise.

Example

```
Gopt = getData("Gopt")
Bopt = getData("Bopt")
Zref = zref(1 ?result "sp")
plot(gmin(Gopt Bopt Zref))
```

gmsg

```
gmsg( o_s11 o_s12 o_s21 o_s22 )
     => o_waveform/nil
```

Description

Returns the maximum stable power gain in terms of the supplied parameters.

Arguments

o_s11	Waveform object representing s11.
o_s12	Waveform object representing s12.
o_s21	Waveform object representing s21.
o_s22	Waveform object representing s22.

Value Returned

o_waveform	Waveform object representing the maximum stable power gain.
nil	Returns `nil` and an error message otherwise.

Example

```
s11 = sp(1 1)
s12 = sp(1 2)
s21 = sp(2 1)
s22 = sp(2 2)
plot(gmsg(s11 s12 s21 s22))
```

gmux

```
gmux( o_s11 o_s12 o_s21 o_s22 )
     => o_waveform/nil
```

Description

Returns the maximum unilateral power gain in terms of the supplied parameters.

Arguments

o_s11	Waveform object representing s11.
o_s12	Waveform object representing s12.
o_s21	Waveform object representing s21.
o_s22	Waveform object representing s22.

Value Returned

o_waveform	Waveform object representing the maximum unilateral power gain.
`nil`	Returns `nil` and an error message otherwise.

Example

```
s11 = sp(1 1)
s12 = sp(1 2)
s21 = sp(2 1)
s22 = sp(2 2)
plot(gmux(s11 s12 s21 s22))
```

gp

```
gp( o_s11 o_s12 o_s21 o_s22 [?gl n_gl] )
    => o_waveform/nil
```

Description

Computes the power gain in terms of the S-parameters.

Arguments

o_s11	Waveform object representing s11.
o_s12	Waveform object representing s12.
o_s21	Waveform object representing s21.
o_s22	Waveform object representing s22
n_gl	Load reflection coefficient. Default value: 0

Value Returned

o_waveform	Waveform object representing the power gain.
`nil`	Returns `nil` and an error message otherwise.

Example

```
s11 = sp(1 1)
s12 = sp(1 2)
s21 = sp(2 1)
s22 = sp(2 2)
plot(gp(s11 s12 s21 s22))
```

Note: gl is an imaginary number which should be input in the following format:

```
gp( s11 s12 s21 s22 ?gl complex(<realPart> <imagPart>))
```

gpc

```
gpc( o_s11 o_s12 o_s21 o_s22 g_level g_frequency )
     => o_waveform/nil
```

Description

Computes the power gain circles.

The *g* datatype on *g_level* and *g_frequency* allows either the level or the frequency to be swept while the other remains fixed.

Arguments

o_s11	Waveform object representing s11.
o_s12	Waveform object representing s12.
o_s21	Waveform object representing s21.
o_s22	Waveform object representing s22.
g_level	Level in dB. It can be specified as a scalar or a vector. If it is specified as a vector, the level is swept. The `linRg` function can be called to generate a linear range. For example, `linRg( -30 30 5 )` is the same as `list(-30 -25 -20 -15 -10 -5 0 5 10 15 20 25 30)` and the *g_level* argument can be specified as either. In that case, a power gain circle is calculated at each one of the 13 levels.
g_frequency	The frequency. It can be specified as a scalar or a linear range. If it is specified as a linear range, the frequency is swept. The linear range is specified as a list with three values: the start of the range, the end of the range, and the increment. For example, `list(100M 1G 100M)` specifies a linear range with the following values: { 100M, 200M, 300M, 400M, 500M, 600M, 700M, 800M, 900M, 1G } In that case, a power gain circle is calculated at each one of the 10 frequencies.

Value Returned

o_waveform	Waveform object representing the power gain circles.
nil	Returns nil and an error message otherwise.

groupDelay

```
groupDelay( o_waveform )
     => o_waveform/nil
```

Description

Computes the group delay of a waveform.

This command returns the derivative of the phase of `o_waveform` / 2pi. Group delay is defined as the derivative of the phase with respect to frequency. Group delay is expressed in seconds.

It is calculated using the `vp` function as shown below:

$$\text{Group Delay} = \frac{d\phi}{d\omega} = \frac{d}{df}\left[\frac{phase(\text{/netX})}{360}\right]$$

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)

Value Returned

o_waveform	Returns a waveform representing the group delay of the specified waveform.
`nil`	Returns `nil` and an error message otherwise.

Example

```
plot( groupDelay( v( "/net3" ) ) )
```

Plots the waveform representing the group delay of the voltage of `"/net3"`.

gt

```
gt( o_s11 o_s12 o_s21 o_s22 [ ?gs n_gs] [ ?gl n_gl] )
     => o_waveform/nil
```

Description

Returns the transducer gain in terms of the supplied parameters and the optional source reflection coefficient (Gs) and the input reflection coefficient (Gl).

Arguments

o_s11	Waveform object representing s11.
o_s12	Waveform object representing s12.
o_s21	Waveform object representing s21.
o_s22	Waveform object representing s22.
n_gs	Source reflection coefficient. Default value: 0
n_gl	Input reflection coefficient. Default value: 0

Value Returned

o_waveform	Waveform object representing the transducer gain.
`nil`	Returns `nil` and displays a message if there is an error.

Example

```
s11 = sp(1 1)
s12 = sp(1 2)
s21 = sp(2 1)
s22 = sp(2 2)
plot(gt(s11 s12 s21 s22))
```

Note: gl is an imaginary number which should be input in the following format:
```
gt( s11 s12 s21 s22 ?gl complex(<realPart> <imagPart>))
```

harmonic

```
harmonic( o_waveform h_index )
     => o_waveform/g_value/nil
```

Description

Returns the waveform for a given harmonic index.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*`XXXXX`*.)
h_index	The index number that designates the harmonic information to be returned. For the `'pss`, `'pac`, and `'pxf` analyses, the index is an integer number. For the `'pdisto` analysis, the index is a list of integers that correspond with the frequency names listed in the `funds` analysis parameter in the netlist. If more than one *h_index* is desired at one time, a list can be specified.

Value Returned

o_waveform	Returns a waveform (when a single *h_index* is specified) or family of waveforms (when more than one *h_index* is specified) if the input argument is a family of waveforms.
g_value	Returns the harmonic value if the input is a single waveform with the X values being harmonics
`nil`	Returns `nil` and displays a message if there is an error.

Example

For each of the following commands:

```
harmonic(v("/net49" ?result "pss-fd.pss") 1)
harmonic(v("/Pif" ?result "pdisto-fi.pdisto") list(1 -1))
```

Each result is a complex number.

For each of the following commands:

```
harmonic(v("/net54" ?result "pac-pac") 1)
harmonic(v("/net51" ?result "sweeppss_pss_fd-sweep") list(8))
harmonic(v("/Pif" ?result "sweeppss_pac-sweep") -8)
harmonic(v("/net36" ?result "sweeppdisto_pdisto_fi-sweep") '(1 -1))
```

Each result is a waveform.

For each of the following commands:

```
harmonic(v("/net54" ?result "pac-pac") list(1 5))
harmonic(v("/net51" ?result "sweeppss_pss_fd-sweep") '(1 8))
harmonic(v("/Pif" ?result "sweeppss_pac-sweep") list(-8 0))
harmonic(v("/net36" ?result "sweeppdisto_pdisto_fi-sweep") '((1 -1) (2 -2) (-1 2)))
```

Each result is a family of waveforms.

Neither of the following commands should be entered:

```
harmonic(v("/net49" ?result "pss-fd.pss") list(0 1))
harmonic(v("/Pif" ?result "pdisto-fi.pdisto") '((1 -1) (-1 2)))
```

Each resulting waveform is not in a useful format.

harmonicFreqList

```
harmonicFreqList( [?resultsDir t_resultsDir] [?result S_resultName])
     => n_list/nil
```

Description

Returns a list of lists, with each sublist containing a harmonic index and the minimum and maximum frequency values that the particular harmonic ranges between.

If both of these frequency values are the same, just one frequency value is returned.

Arguments

t_resultsDir	Directory containing the PSF files (results). If you supply this argument, you must also supply the *resultName* argument.
S_resultName	Results from an analysis.

Value Returned

n_list	Returns a list of lists. For the `'pss`, `'pac`, and `'pxf` analyses, the first element of each sublist is an integer number. For the `'pdisto` analysis, the first element of each sublist is a list of integers that correspond with the frequency names listed in the `funds` analysis parameter in the netlist. For all sublists, the remaining entries are the minimum and maximum frequency values that the particular harmonic ranges between. If both of these frequency values are the same, just one frequency value is returned.
`nil`	Returns `nil` if no harmonics are found in the data.

Example

For each of the following commands:

```
harmonicFreqList( ?result "pss-fd.pss" )
harmonicFreqList( ?result "pac-pac" )
harmonicFreqList( ?result "sweeppss_pss_fd-sweep" )
harmonicFreqList( ?result "sweeppss_pac-sweep" )
```

Each result is a list of integers.

For each of the following commands:

```
harmonicFreqList( ?result "pdisto-fi.pdisto" )
harmonicFreqList( ?result "sweeppdisto_pdisto_fi-sweep" )
```

Each result is a list of lists, with each sublist containing a combination of integer numbers that correspond with the frequency names listed in the `funds` analysis parameter in the netlist. These names can also be extracted from the PSF data by using the `resultParam` function to find the `'largefundname` and `'moderatefundnames` values. For example:

```
strcat(resultParam( 'largefundname ?result "pdisto-fi.pdisto" ) " "
resultParam( 'moderatefundnames ?result "pdisto-fi.pdisto" ))
```

Returns a string representing the order of the frequency names.

harmonicList

```
harmonicList( [?resultsDir t_resultsDir] [?result S_resultName] )
    => n_list
```

Description

Returns the list of harmonic indices available in the `resultName` or current result data.

Arguments

t_resultsDir	Directory containing the PSF files (results). If you supply this argument, you must also supply the resultName argument.
S_resultName	Results from an analysis.

Value Returned

n_list	Returns a list of harmonic indices. For the `'pss`, `'pac`, and `'pxf` analyses, the index is an integer number. For the `'pdisto` analysis, the index is a list of integers that correspond with the frequency names listed in the `'funds` analysis parameter in the netlist.
`nil`	Returns `nil` if no harmonics are found in the data.

Example

For each of the following commands:

```
harmonicList( ?result "pss-fd.pss" )
harmonicList( ?result "pac-pac" )
harmonicList( ?result "sweeppss_pss_fd-sweep" )
harmonicList( ?result "sweeppss_pac-sweep" )
```

Each result is a list of integers.

For each of the following commands:

```
harmonicList( ?result "pdisto-fi.pdisto" )
harmonicList( ?result "sweeppdisto_pdisto_fi-sweep" )
```

Each result is a list of lists, with each sublist containing a combination of integer numbers that correspond with the frequency names listed in the `'funds` analysis parameter in the netlist. These names can also be extracted from the PSF data by using the `'resultParam` function to find the `'largefundname` and `'moderatefundnames` values. For example:

```
strcat(resultParam( 'largefundname ?result "pdisto-fi.pdisto" ) " "
resultParam( 'moderatefundnames ?result "pdisto-fi.pdisto" ))
```

Returns a string representing the order of the frequency names.

histo

```
histo( o_waveform x_bins n_min n_max )
     => o_histoWaveform/nil
```

Description

Returns a waveform that represents the statistical distribution of input data in the form of a histogram. The height of the bars (or bins) in the histogram represents the frequency of the occurrence of values within a specific period. Using the `histo` function, the range for capturing these frequencies can be specified through the *n_min* and *n_max* values.

Arguments

o_waveform	Input waveform.
x_bins	Number of bins to represent the input data.
n_min	The first value on the horizontal axis of the histogram. By default, it assumes the minimum value of the input waveform.
n_max	The last value on the horizontal axis of the histogram. By default, it assumes the maximum value of the input waveform.

Value Returned

o_histoWaveform	Returns a waveform representing the statistical distribution of the input waveform *o_waveform*.
nil	Returns `nil` in case of an error.

Example

```
histo( VT("/vin") 3 1.5 3.5)
=> out_wave
plot( out_wave )
```

Plots the output waveform `out_wave` as a histogram, which represents the statistical distribution of the input waveform `VT("/vin")`.

histogram2D

```
histogram2D(o_waveform x_nbins s_type g_setAnnotation g_setDensityEstimator)
    => o_waveform/nil
```

Description

Returns a waveform that represents the statistical distribution of input data in the form of a histogram. The height of the bars (or bins) in the histogram represents the frequency of the occurrence of values within a specific period.

Arguments

o_waveform	Input waveform.
x_nbins	Number of bins (bars) to be plotted in the resulting histogram plot. Valid values: `1` to `50`. Default value:`10`.
s_type	Type of histogram to be plotted. Valid values: `Standard`, `Cumulative line`, and `Cumulative box`. Default value: `Standard`.
g_setAnnotation	Boolean specifying whether to display the standard deviation lines in the resulting histogram plot. Valid values: `t` or `nil` Default value: `nil`
g_setDensityEstimator	Boolean specifying whether the resulting histogram plot display a curve that estimates the distribution concentration. Valid values: `t` or `nil` Default value: `nil`

Value Returned

o_waveform	Returns a waveform representing the statistical distribution of the input waveform *o_waveform*.
`nil`	Returns `nil` in case of an error.

Example

```
histogram2D(i("/V2/PLUS" ?result "tran") 10 "standard" t t )
```

Plots the output waveform `out_wave` as a histogram, which represents the statistical distribution of the input waveform `/V2/PLUS`.

iinteg

```
iinteg( o_waveform )
     => o_waveform/nil
```

Description

Computes the indefinite integral of a waveform with respect to the X-axis variable.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)

Value Returned

o_waveform	Returns a waveform representing the indefinite integral of the input waveform.
`nil`	Returns `nil` and an error message otherwise.

Example

```
plot( iinteg( v( "/net8" )))
```

Computes the indefinite integral of the waveform representing the voltage of `"/net8"`.

imag

```
imag( {o_waveform | n_input} )
    => o_waveformImag/n_numberImag/nil
```

Description

Returns the imaginary part of a waveform representing a complex number or returns the imaginary part of a complex number.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*`XXXXX`*.)
n_input	Complex number.

Value Returned

o_waveformImag	Returns a waveform when the input argument is a waveform.
n_numberImag	Returns a number when the input argument is a number.
`nil`	Returns `nil` and an error message otherwise.

Example

```
imag( v( "/net8" ) )
```

Returns a waveform representing the imaginary part of the voltage of `"/net8"`. You also can use the `vim` alias to perform the same command, as in
`vim( "net8" ).`

```
x=complex( -1 -2 ) => complex(-1, -2)
imag( x ) => -2.0
```

Creates a variable `x` representing a complex number, and returns the real portion of that complex number.

inl

```
inl( o_dacSignal o_sample|o_pointList|n_interval [?mode t_mode] [?threshold
     n_threshold] [?crossType t_crossType] [?delay f_delay] [?units x_units]
     [?nbsamples n_nbsamples] )
     => n_inl/nil
```

Description

Computes the integral non-linearity of a transient simple or parametric waveform.

Arguments

o_dacSignal	Waveform for which the integral non-linearity is to be calculated.
o_sample	Waveform used to obtain the points for sampling the *dacSignal*. These are the points at which the waveform crosses the threshold while either `rising` or `falling` (defined by the *crossType* argument) with the *delay* added to them.
n_pointList	List of domain values at which the sample points are obtained from the *dacSignal*.
n_interval	The sampling interval.
t_mode	The mode for calculating the threshold. Valid values: `auto` and `user`. Default value: `auto`. If set to `user`, an `n_threshold` value needs to be provided. If set to `auto`, `n_threshold` is calculated internally.
n_threshold	The threshold value against which the integral non-linearity is to be calculated. It needs to be specified only when the *mode* is selected as `user`.
t_crossType	The points at which the curves of the waveform intersect with the threshold. While intersecting, the curve may be either rising or falling. Valid values: `rising` and `falling`, respectively. Default `crossType` is `rising`.

f_delay	The delay time after which the sampling begins. Valid values: Any valid time value. Default value: `0`.
x_units	Unit for expressing the output waveform. Valid values: `abs` (absolute) and `lsb` (multiples of least significant bit). Default value: `abs`.
n_nbsamples	Number of samples used for calculating the non-linearity. If not specified, the samples are taken against the entire data window.

Note: For each of the three ways in which the sample points can be specified, only a few of the other optional arguments are meaningful, as indicated below:

- For *o_sample*, the arguments *t_mode*, *n_threshold*, *t_crossType*, *f_delay*, and *x_units* are meaningful.
- For *n_pointList*, the arguments *x_units* are meaningful.
- For *n_interval*, the arguments *x_units*, and *n_nbsamples* are meaningful.

Value Returned

n_inl	Returns the integral non-linearity waveform.
`nil`	Returns `nil` and an error message otherwise.

Example

```
inl( wave1 wave2 ?crossType "rising" ?delay 0.4 )
=> srrWave:175051544
```

Returns the integral non-linearity for `wave1` by taking the points at which `wave2` crosses the internally calculated threshold while `rising` as the sample points and adding a delay of `0.4` to them.

integ

```
integ( o_waveform,[n_intial_limit,n_final_limit] )
     => o_waveform/n_value/nil
```

Description

Computes the definite integral of the waveform with respect to a range specified on the X-axis of the waveform. The result is the value of the area under the curve over the range specified on the X-axis.

You should specify either both the limits or neither. In case you do specify the limits, they become the end points of the range on the X-axis for definite integration. If you do not specify the limits, then the range for definite integration is the entire range of the sweep on the X-axis.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)
initial_limit_n	Initial limit for definite integration.
final_limit_n	Final limit for definite integration.

Value Returned

o_waveform	Returns a waveform representing the definite integral for a family of waveforms if the input argument is a family of waveforms.
n_value	Returns a numerical value representing the definite integral of the input waveform if the input argument is a single waveform.
`nil`	Returns `nil` and an error message otherwise.

Example

```
integ( v( "/out" ) )
```

Returns the definite integral of the waveform representing the voltage of `"/out"` over its entire range.

```
integ( VT( "/out" ),12.5n,18n)
```

Returns the definite integral of the waveform representing the voltage of `"/out"` within a specified range.

intersect

```
intersect( o_waveform1 o_waveform2 )
    => o_wave/nil
```

Description

Returns a waveform containing the points of intersection for two waveforms passed as arguments.

Arguments

o_waveform1	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)
o_waveform2	Additional waveform object.

Value Returned

o_wave	Returns a waveform containing the points of intersection for the two waveforms passed as arguments.
`nil`	Returns `nil` if the two waveforms are disjoint or overlap each other, and an error message, if the arguments to the function are not correct.

Example

```
intersect( VT("/inp1") VT("/inp2") )
```

ipn

```
ipn( o_spurious o_reference [ f_ordspur f_ordref f_epspur f_epref g_psweep
     s_measure ] )
     => o_waveform/f_number/nil
```

Description

Performs an intermodulation *n*th-order intercept measurement.

The data for this measurement can be either a single input power value or a parametric input power sweep.

From each of the spurious and reference power waveforms (or points), the `ipn` function extrapolates a line of constant slope (dB/dB) according to the specified order and input power level. These lines represent constant small-signal power gain (ideal gain). The `ipn` function calculates the intersection of these two lines and returns the value of either the X coordinate (input referred) or Y coordinate.

Arguments

o_spurious	Waveform or number representing the spurious output power (in dBm).
o_reference	Waveform or number representing the reference output power (in dBm).
f_ordspur	Order or slope of the spurious constant-slope power line. Default value: `3`
f_ordref	Order or slope of the reference constant-slope power line. Default value: `1`
f_epspur	Value (in dBm) used to indicate the point where the spurious constant-slope power line begins. If *g_psweep* `is t`, this value is the input power value of the point on the *o_spurious* waveform, otherwise this value is paired with the *o_spurious* value to define the point. This point should be in the linear region of operation. (If *g_psweep* `is t`, *f_spspur* defaults to the X coordinate of the first point of the *o_spurious* wave; if *s_measure* is `'input`, a number must be specified.)

f_epref — Value (in dBm) used to indicate the point where the reference constant-slope power line begins. If *g_psweep* is `t`, this value is the input power value of the point on the *o_reference* waveform, otherwise this value is paired with the *o_reference* value to define the point. This point should be in the linear region of operation. (If *g_psweep* is `t`, *f_epref* defaults to the X coordinate of the first point of the *o_reference* wave; if *s_measure* is `'input`, a number must be specified.)

g_psweep — Boolean indicating that the input power to the circuit was a parametric sweep. The power sweep must both be in dBm and be performed at the lowest parametric level.
Default value: `t`

s_measure — Name indicating if measurement is to be input referred (`'input`) or output referred (`'output`).
Default value: `'input`

Value Returned

o_waveform — Depending on setting of *g_psweep* and the dimension of the input waveforms, returns either a waveform or a family of waveforms.

f_number — If *o_spurious* and *o_reference* are numbers or they are waveforms when *g_psweep* is `t`, returns a number.

`nil` — Returns `nil` and an error message otherwise.

Example

```
spurWave = db20(harmonic(wave signalHarmonic))
refWave = db20(harmonic(wave referenceHarmonic))
xloc = ipn( spurWave refWave 3.0 1.0 -25 -25 )
yloc = ipn( spurWave refWave 3.0 1.0 -25 -25 t "Output")
```

Computes the IP3 point for the given wave.

Each of the following examples returns an ip3 measurement.

```
ipn(dB20(harmonic(v("/Pif" ?result "pss_fd") 9))
    dB20(harmonic(v("/Pif" ?result "pss_fd") 8)))
```

```
ipn(dbm(harmonic(spectralPower(v("/Pif" ?result "pss_fd")/50.0
    v("/Pif" ?result "pss_fd")) 9))
```

```
    dbm(harmonic(spectralPower(v("/Pif" ?result "pss_fd")/50.0
    v("/Pif" ?result "pss_fd")) 8)))

ipn(dbm(harmonic(spectralPower(v("/Pif" ?result "pss_fd")
    /resultParam("rif:r" ?result "pss_td")
    v("/Pif" ?result "pss_fd")) 9))
    dbm(harmonic(spectralPower(v("/Pif" ?result "pss_fd")
    /resultParam("rif:r" ?result "pss_td")
    v("/Pif" ?result "pss_fd")) 8)))

ipn(dbm(harmonic(spectralPower(i("/rif/PLUS" ?result "pss_fd")
    v("/Pif" ?result "pss_fd")) 9))
    dbm(harmonic(spectralPower(i("/rif/PLUS" ?result "pss_fd")
    v("/Pif" ?result "pss_fd")) 8))
    3. 1. -25 -25 t "Output")

ipn(dbm(harmonic(spectralPower(v("/Pif" ?result "pac")
    /resultParam("rif:r" ?result "pss_td")
    v("/Pif" ?result "pac")) -21))
    dbm(harmonic(spectralPower(v("/Pif" ?result "pac")
    /resultParam("rif:r" ?result "pss_td")
    v("/Pif" ?result "pac")) -25)))
```

ipnVRI

```
ipnVRI( o_vport x_harmspur x_harmref [?iport o_iport] [?rport f_rport]
    [?ordspur f_ordspur] [?epoint f_epoint] [?psweep g_psweep] [?epref f_epref]
    [?ordref f_ordref] [?measure s_measure] )
    => o_waveform/f_number/nil
```

Description

Performs an intermodulation *n*th-order intercept point measurement.

Use this function to simplify the declaration of an ipn measurement. This function extracts the spurious and reference harmonics from the input waveform(s), and uses `dBm(spectralPower((i or v/r),v))` to calculate the respective powers. The function passes these power curves or numbers and the remaining arguments to the `ipn` function to complete the measurement.

From each of the spurious and reference power waveforms (or points), the `ipn` function extrapolates a line of constant slope (dB/dB) according to the specified order and input power level. These lines represent constant small-signal power gain (ideal gain). The `ipn` function calculates the intersection of these two lines and returns the value of either the X coordinate (input referred) or the Y coordinate.

Arguments

o_vport	Voltage across the output port. This argument must be a family of spectrum waveforms (1 point per harmonic), with the option of containing a parametric input power sweep (in dBm).
x_harmspur	Harmonic number of the spurious voltage contained in *o_vport*. When *o_iport* is specified, also applies to a current waveform contained in *o_iport*.
x_harmref	Harmonic index of the reference voltage contained in *o_vport*. When *o_iport* is specified, also applies to a current waveform contained in *o_iport*.
o_iport	Current into the output port. This argument must be a family of spectrum waveforms (1 point per harmonic), with the option of containing a parametric input power sweep (in dBm). When specified, power is calculated using voltage and current.
f_rport	Resistance into the output port. When specified and *o_iport* is `nil`, the output power is calculated using voltage and

	resistance. Default value: `50`
f_ordspur	Order or slope of the spurious constant-slope power line. Default value: `3`
f_epoint	Value (in dBm) used to indicate the point where the spurious constant-slope power line begins. If *g_psweep* is `t`, this value is the input power value of the point on the *o_spurious* waveform, otherwise this value is paired with the *o_spurious* value to define the point. This point should be in the linear region of operation. Default value: If *g_psweep* is `t`, the lowest input power value; if *s_measure* is `'input`, a number must be specified.
g_psweep	Boolean indicating that the input power to the circuit was a parametric sweep. The power sweep must be in dBm and must be performed at the lowest parametric level. Default value: `t`
f_epref	Value (in dBm) used to indicate the point where the reference constant-slope power line begins. If *g_psweep* is `t`, this value is the input power value of the point on the *o_reference* waveform, otherwise this value is paired with the *o_reference* value to define the point. This point should be in the linear region of operation. Default value: If *f_epoint* is not `nil`, *f_epoint*; else if *g_psweep* is `t`, the X coordinate of the first point of the *o_reference* wave; else if *s_measure* is `'input`, a number must be specified.
f_ordref	Order or slope of the reference constant-slope power line. Default value: `1`
s_measure	Symbol indicating if measurement is to be input referred (`'input`) or output referred (`'output`). Default value: `'input`

Value Returned

o_waveform	Depending on the setting of *g_psweep* and the dimension of input waveform(s), the `ipnVRI` function returns either a waveform or a family of waveforms.

f_number	Depending on the setting of *g_psweep* and the dimension of input waveform(s), the `ipnVRI` function returns a number.
`nil`	Returns `nil` and an error message otherwise.

Example

Each of following examples returns an ip3 measurement:

```
ipnVRI(v("/Pif" ?result "pss_fd") 9 8)
```

```
ipnVRI(v("/Pif" ?result "pss_fd") 9 8
    ?rport resultParam("rif:r" ?result "pss_td"))
```

```
ipnVRI(v("/Pif" ?result "pss_fd") 9 8
    ?iport i("/rif/PLUS" ?result "pss_fd") ?epoint -25
    ?measure "Output")
```

```
ipnVRI(v("/Pif" ?result "pac") -21 -25
    ?rport resultParam("rif:r" ?result "pss_td"))
```

ipnVRICurves

```
ipnVRICurves( o_vport x_harmspur x_harmref [?iport o_iport] [?rport f_rport]
     [?ordspur f_ordspur] [?epoint f_epoint] [?psweep g_psweep] [?epref f_epref]
     [?ordref f_ordref] )
     => o_waveform/nil
```

Description

Constructs the waveforms associated with an ipn measurement.

Use this function to simplify the creation of waves associated with an ipn measurement. This function extracts the spurious and reference harmonics from the input waveform(s), and uses `dBm(spectralPower((i or v/r),v))` to calculate the respective powers.

From each of the spurious and reference power waveforms (or points), the `ipnVRICurves` function extrapolates a line of constant slope (dB/dB) according to the specified order and input power level. These lines represent constant small-signal power gain (ideal gain). The function returns these lines and power waveforms (when present) as a family of waveforms.

This function only creates waveforms and does not perform an ipn measurement or include labels with the waveforms. Use the `ipn` or `ipnVRI` function for making measurements.

Arguments

o_vport — Voltage across the output port. This argument must be a family of spectrum waveforms (1 point per harmonic), with the option of containing a parametric input power sweep (in dBm).

x_harmspur — Harmonic index of the spurious voltage contained in *o_vport*. When *o_iport* is specified, also applies to a current waveform contained in *o_iport*.

x_harmref — Harmonic index of the reference voltage contained in *o_vport*. When *o_iport* is specified, also applies to a current waveform contained in *o_iport*.

o_iport — Current into the output port. This argument must be a family of spectrum waveforms (1 point per harmonic), with the option of containing a parametric input power sweep (in dBm). When specified, power is calculated using voltage and current.

f_rport	Resistance into the output port. When specified and *o_iport* is `nil`, the output power is calculated using voltage and resistance. Default value: `50`
f_ordspur	Order or slope of the spurious constant-slope power line. Default value: `3`
f_epoint	Value (in dBm) used to indicate the point where the spurious constant-slope power line begins. If *g_psweep* is `t`, this value is the input power value of the point on the *o_spurious* waveform, otherwise this value is paired with the *o_spurious* value to define the point. This point should be in the linear region of operation. Default value: If *g_psweep* is `t`, the X coordinate of the first point of the *o_spurious* wave; otherwise a number must be specified.
g_psweep	Boolean indicating that the input power to the circuit was a parametric sweep. The power sweep must be in dBm and must be performed at the lowest parametric level. Default value: `t`
f_epref	Value (in dBm) used to indicate the point where the reference constant-slope power line begins. If *g_psweep* is `t`, this value is the input power value of the point on the *o_reference* waveform, otherwise this value is paired with the *o_reference* value to define the point. This point should be in the linear region of operation. Default value: If *f_epoint* is not `nil`, *f_epoint*; else if *g_psweep* is `t`, the X coordinate of the first point of the *o_reference* wave; else a number must be specified.
f_ordref	Order or slope of the reference constant-slope power line. Default value: `1`

Value Returned

o_waveform	A family of waveforms that contains the spurious and reference tangent lines, and when *g_psweep* is `t`, contains the spurious and reference waveforms.
`nil`	Returns `nil` and an error message otherwise.

Example

Each of following examples returns curves related to an ip3 measurement:

```
ipnVRICurves(v("/Pif" ?result "pss_fd") 9 8)
```

```
ipnVRICurves(v("/Pif" ?result "pss_fd") 9 8
    ?rport resultParam("rif:r" ?result "pss_td"))
```

```
ipnVRICurves(v("/Pif" ?result "pss_fd") 9 8
    ?iport i("/rif/PLUS" ?result "pss_fd") ?epoint -25)
```

```
ipnVRICurves(v("/Pif" ?result "pac") -21 -25
    ?rport resultParam("rif:r" ?result "pss_td"))
```

kf

```
kf( o_s11 o_s12 o_s21 o_s22 )
     => o_waveform/nil
```

Description

Returns the stability factor in terms of the supplied parameters.

Arguments

o_s11	Waveform object representing s11.
o_s12	Waveform object representing s12.
o_s21	Waveform object representing s21.
o_s22	Waveform object representing s22.

Value Returned

o_waveform	Waveform object representing the stability factor.
nil	Returns nil if there is an error.

Example

```
s11 = sp(1 1)
s12 = sp(1 2)
s21 = sp(2 1)
s22 = sp(2 2)
plot(kf(s11 s12 s21 s22))
```

ln

```
ln( {o_waveform | n_number} )
     => o_waveform/f_number/nil
```

Description

Gets the base-e (natural) logarithm of a waveform or number.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*`XXXXX`*.)
n_number	Number.

Value Returned

o_waveform	Returns a waveform object representing the base-e (natural) logarithm of the input waveform if the input argument is a waveform object, or returns a family of waveforms if the input argument is a family of waveforms
f_number	Returns a number if the input argument is a number.
`nil`	Returns `nil` and an error message otherwise.

Example

```
ln( v( "/net9" ) )
```

Gets a waveform that is calculated as the natural logarithm of the input waveform.

```
ln(ymax(v("/net9")))
```

Gets a waveform that is calculated as the natural logarithm of the following: `ymax(v("/net9"))`.

```
ln(100)
=> 4.60517
```

Gets the natural logarithm of 100.

log10

```
log10( {o_waveform | n_number} )
     => o_waveform/n_number/nil
```

Description

Gets the base-10 logarithm of a waveform or a number.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*`XXXXX`*.)
n_number	Number.

Value Returned

o_waveform	Returns a waveform object if the input argument is a waveform object or returns a family of waveforms if the input argument is a family of waveforms.
n_number	Returns a number that is calculated as the base-10 logarithm of the input number.
`nil`	Returns `nil` and an error message otherwise.

Example

```
log10( v( "/net9" ) )
```

Gets a waveform that is calculated as the base-10 logarithm of the input waveform.

```
log10( ymax( v( "/net9" ) ) )
```

Gets a waveform representing the base-10 logarithm of `ymax(v("/net9"))`.

```
log10( 100 )
=> 2.0
```

Gets the base-10 logarithm of 100, or 2.

lsb

```
lsb( o_s11 o_s12 o_s21 o_s22 g_frequency )
     => o_waveform/nil
```

Description

Computes the load stability circles.

Arguments

o_s11 Waveform object representing s11.

o_s12 Waveform object representing s12.

o_s21 Waveform object representing s21.

o_s22 Waveform object representing s22.

g_frequency Frequency. It can be specified as a scalar or a linear range. If it is specified as a linear range, the frequency is swept. The linear range is specified as a list with three values: the start of the range, the end of the range, and the increment. For example, `list(100M 1G 100M)` specifies a linear range with the following values:

```
{ 100M, 200M, 300M, 400M, 500M, 600M, 700M,
800M, 900M, 1G }
```

In that case, a load stability circle is calculated at each one of the 10 frequencies

Value Returned

o_waveform Waveform object representing the load stability circles.

`nil` Returns `nil` and an error message otherwise.

Example

```
plot(lsb(s11 s12 s21 s22 list(800M 1G 100M)))
```

lshift

```
lshift( o_waveform n_delta )
     => o_waveform/nil
```

Description

Shifts the waveform to the left by the delta value.

This command is the inverse of the `rshift` command.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)
n_delta	Value by which the waveform is to be shifted.

Value Returned

o_waveform	Returns a waveform object representing the input waveform shifted to the left. Returns a family of waveforms if the input argument is a family of waveforms.
`nil`	Returns `nil` and an error message otherwise.

Example

```
plot( lshift( v( "/net8" ) 30u ) )
```

Shifts the waveform representing the voltage of `"/net8"` to the left by `30u` and plots the resulting waveform.

mag

```
mag( {o_waveform | n_number} )
     => o_waveform/n_number/nil
```

Description

Gets the magnitude of a waveform or number.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)
n_number	Number.

Value Returned

o_waveform	Returns a waveform object if the input argument is a waveform object or returns a family of waveforms if the input argument is a family of waveforms.
n_number	Returns a number if the input argument is a number.
`nil`	Returns `nil` and an error message otherwise.

Example

```
mag( v( "5" ) )
```

Gets the magnitude of the waveform representing the voltage at net 5. You can also use the `vm` alias to perform the same command, as in `vm( "5" )`.

```
mag( i( "VFB" ) )
```

Gets the magnitude of the waveform representing current through the `VFB` component. You can also use the `im` alias to perform the same command, as in `im( "VFB" )`.

```
mag( -10 ) => 10
```

Returns the magnitude of `-10`.

nc

```
nc( o_Fmin o_Gmin o_rn g_level g_frequency )
    => o_waveform/nil
```

Description

Computes the noise circles.

Arguments

o_Fmin Waveform object representing the minimum noise factor.

o_Gmin Waveform object representing the optimum noise reflection.

o_rn Waveform object representing the normalized equivalent noise resistance.

g_level Level in dB. It can be specified as a scalar or a vector. The level is swept, if it is specified as a vector. The `linRg` function can be called to generate a linear range. For example, `linRg( -30 30 5 )` is the same as `list(-30 -25 -20 -15 -10 -5 0 5 10 15 20 25 30)` and the *g_level* argument can be specified as either of the above. In that case, a noise circle is calculated at each one of the 13 levels.

g_frequency Frequency. It can be specified as a scalar or a linear range. The frequency is swept if it is specified as a linear range. The linear range is specified as a list with three values: the start of the range, the end of the range, and the increment. For example, `list(100M 1G 100M)` specifies a linear range with the following values:

{ 100M, 200M, 300M, 400M, 500M, 600M, 700M, 800M, 900M, 1G }

In that case, a noise circle is calculated at each one of the 10 frequencies.

Value Returned

o_waveform Waveform object representing the noise circles.

`nil`	Returns `nil` and an error message otherwise.

Example

```
Gopt = getData("Gopt")
Bopt = getData("Bopt")
Zref = zref(1 ?result "sp")
Gmin = gmin(Gopt Bopt Zref)
Fmin = getData("Fmin")
rn = getData("NNR")
NC = nc(Fmin Gmin rn 10 list(100M 1G 100M))
displayMode("smith")
smithType("impedance")
plot(NC)
```

normalQQ

```
normalQQ( o_waveform )
     => o_waveform/nil
```

Description

Returns a quantile-quantile plot of the sample quantiles versus theoretical quantiles from a normal distribution. If the distribution is normal, the plot is close to a linear waveform.

Argument

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)

Values Returned

o_waveform	Returns the waveform representing the normal quantile plot.
`nil`	Returns `nil` and an error message otherwise.

Example

```
normalQQ(v("net10" ?result "tran"))
```

Returns the quantile plot for the `v("net10" ?result "tran")` signal.

overshoot

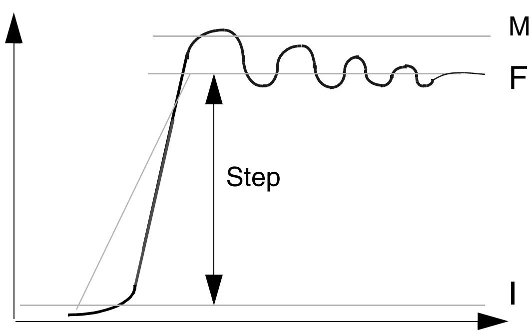
```
overshoot( o_waveform n_initVal g_initType n_finalVal g_finalType [g_multiple
     [s_Xname]][g_histoDisplay][x_noOfHistoBins] )
     => o_waveform/n_value/nil
```

Description

Computes the percentage by which an expression overshoots a step going from the initial value to the final value you enter.

This command returns the overshoot of *o_waveform* as a percentage of the difference between the initial value and the final value.

In the equation below, M represents Maximum Value of the peak wave, F represents Final Value of the settled wave, and I represents Initial Value of the wave.



$$\text{Overshoot} = \frac{(M-F)x100}{F-I}$$

Arguments

o_waveform — Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)

n_initVal — Initial X value at which to start the computation.

g_initType — Specifies how *initVal* functions.
Valid values: a non-nil value specifies that the initial value is taken to be the value of the waveform, interpolated at *initVal*, and the waveform is clipped from below, as follows:
o_waveform = clip(*o_waveform* initVal nil)

`nil` specifies that *`initVal`* is defined by the X value entered. (The command gets the Y value for the specified X value and uses that value for *`initVal`*.)

`n_finalVal` Final value at which to end the computation.

`g_finalType` Specifies how *`finalVal`* functions.
Valid values: a non-nil value specifies that the final value is taken to be the value of the waveform, interpolated at *`finalVal`*, and the waveform is clipped from above, as follows:

```
o_waveform = clip( o_waveform nil finalVal )
```

`nil` specifies that *`finalVal`* is defined by the X value entered. (The command gets the Y value for the specified X value and uses that value for *`finalVal`*.)

`g_multiple` An optional boolean argument that takes the value `nil` by default. If set to `t`, the function returns multiple occurrences of the overshoot event.

`s_xName` An optional argument that is used only when *`g_multiple`* is set to `t`. It takes the value `time` by default. It controls the contents of the x vector of the waveform object returned by the function.
Valid values: `'time`, `'cycle`

`g_histoDisplay` When set to t, returns a waveform that represents the statistical distribution of the riseTime data in the form of a histogram. The height of the bars (bins) in the histogram represents the frequency of the occurrence of values within the range of riseTime data.
Valid values: `t nil`
Default value: `nil`

`x_noOfHistoBins` Denotes the number of bins represented in the histogram representation.
Valid values: Any positive integer
Default value: `nil`

Note: *`g_histoDisplay`* and *`x_noOfHistoBins`* are added for backward compatibility only. It will be deprecated in future releases. Use the histo function for plotting the histogram of the resulting function.

Value Returned

o_waveform	Returns a waveform (or family of waveforms) representing the amount of overshoot in comparison to the whole signal if the input argument is a family of waveforms or if *g_multiple* is set to `t`.
n_value	Returns a value for the amount of overshoot in comparison to the whole signal if the input is a single waveform.
`nil`	Returns `nil` and an error message otherwise.

Example

```
overshoot( v( "/net8" ) 7n t 3.99u t )
```

Returns the value of the overshoot for the waveform representing the voltage of `"/net8"`.

```
overshoot(VT("/out") 0.5 nil 4.95 nil 5 t 'time)
```

Returns multiple occurrences of overshoot specified against time-points at which each overshoot event occurs.

```
overshoot(VT("/out") 0.5 nil 4.95 nil 5 t 'cycle)
```

Returns multiple occurrences of overshoot specified against cycle numbers, where a cycle number refers to the n'th occurrence of the overshoot event in the input waveform.

pavg

```
pavg( o_waveform n_from n_to [n_period [n_sfactor]])
     => o_waveform/nil
```

Description

Computes the periodic average of a family of signals for each time point.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like srrWave:XXXXX.).
n_from	Starting numeric value for the range on the X-axis.
n_to	Ending numeric value for the range on the X-axis.
n_period	Numeric value for the period of the input waveform.
n_sfactor	Sampling factor. This can be increased in order to increase the accuracy of the output. Default value: 1

Values Returned

o_waveform	Returns a waveform representing the periodic average of a family of signals.
`nil`	Returns `nil` and an error message otherwise.

Example

```
pavg( v("/net8") ?from 1n ?to 20n ?period 2n ?sfactor 1)
```

Returns the value of the periodic average for the family of waveforms representing the voltage of `"/net8"`.

peak

```
peak( o_waveform ?from f_from ?to f_to ?xtol f_xtol ?ytol f_ytol )
    => o_waveform/nil
```

Description

Detects the peaks in the input waveform and returns the X and Y coordinates of these peak points in the form of a waveform.

Note: The function will not work for waveforms that comprise of complex numbers.

Arguments

o_waveform	Input waveform.
f_from	The initial point on the given waveform to start determining the peaks. By default, the first point of the waveform is the starting point.
f_to	The final point on the given waveform up to which the peaks are to be determined. By default, the last point of the waveform is the end point.
f_xtol	The distance on the X axis within which all peaks are to be filtered. Default: 0.0
f_ytol	The distance on the Y axis within which all peaks are to be filtered. Default: 0.0

Note: If both *f_xtol* and *f_ytol* are specified, the filtering mechanism will operate as follows:

- The maximum peak is selected first.
- All adjacent peaks in the neighborhood of both *f_xtol* in the X-axis direction and *f_ytol*in the Y-axis direction are then filtered.
- Next, all the peaks in the rectangular window thus formed are filtered based on both *f_xtol* and *f_ytol*.

If only one of *f_xtol* or *f_ytol* is specified, the peaks are filtered only in either the X-axis direction or the Y-axis direction, respectively.

Value Returned

o_waveform	Returns a waveform whose X and Y coordinates of the peaks are determined from the input waveform and the peaks are filtered based on the *f_xtol* and *f_ytol* criteria.
nil	Returns nil and an error message otherwise.

Example

```
peak( vt("/out") ?from 1n ?to 20n ?xtol 2n ?ytol 0.5)
```

Out of all the peaks in the region starting from 1n to 20n, the function returns a waveform comprising of some of these peaks that satisfy the criteria of *x-tol* (2n) and *ytol* (0.5).

peakToPeak

```
peakToPeak( o_waveform )
     => o_waveform/n_value/nil
```

Description

Returns the difference between the maximum and minimum values of a waveform.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)

Value Returned

o_waveform	Returns a waveform or a family of waveforms if the input argument is a family of waveforms.
n_value	Returns the difference between the maximum and minimum values of a waveform if the input argument is a single waveform.
`nil`	Returns `nil` and an error message otherwise.

Example

```
peakToPeak( v( "/net2" ) )
```

Returns the difference between the maximum and minimum values of the waveform representing the voltage of the `"/net2"` net.

period_jitter

```
period_jitter( o_waveform t_crossType [?mode t_mode] [?threshold n_threshold]
     [binSize n_binSize] [?xName t_xName] [?outputType t_outputType] )
     => o_waveform/f_val/nil
```

Description

Computes the period jitter. It returns a waveform or a value representing deviation from the average period.

Arguments

o_waveform — Name of the signal, expression, or a family of waveforms.

t_crossType — The points at which the curves of the waveform intersect with the threshold. While intersecting, the curve may be either rising or falling.
Valid values: `rising` and `falling`.
Default value: `rising`.

t_mode — The mode for calculating the threshold.
Valid values: `auto` and `user`.
If set to `user`, an `n_threshold` value needs to be specified by you.
If set to `auto`, `n_threshold` is calculated as:

Auto Threshold Value = integral of the waveform divided by the X range

Default value: `auto`.

n_threshold — The threshold value against which the period is to be calculated. It needs to be specified only when the `mode` selected is `user`.

n_binSize — The width of the moving average window.The deviation of value at the particular point from the average of this window is the jitter.

If binsize=0, all periods are used to calculate the average.
If binsize=N, the last N periods are used to calculate the average.

t_xName — The X-axis of the output waveform. It specifies whether you want to retrieve the period jitter against time (or another X-axis

	parameter for non-transient data) or cycle. Cycle numbers refer to the n'th occurrence of the delay event in the input waveform. Valid values: `time` and `cycle`. Default value: `time`.
t_outputType	Type of output. Valid values: `sd` and `plot`. If set to `plot`, the output is a jitter waveform. If set to `sd`, the output is a standard deviation of the jitter waveform. Default value: `plot`.

Value Returned

o_waveform	Returns the period jitter values as a function of time or cycle when the *outputType* is set to `plot`.
f_val	Returns the standard deviation value when the *outputType* is set to `sd`.
`nil`	Returns `nil` otherwise.

Example

```
period_jitter( wave1 "rising" ?mode "user" ?threshold 1 ?binSize 2 ?xName "cycle"
?outputType "sd" )
=> 1.695467
```

Returns the standard deviation for the period jitter of `wave1` with the threshold of 1 against the cycle on the x-axis.

phase

```
phase( {o_waveform | n_number} )
     => o_waveform/n_number/nil
```

Description

Gets the phase of the waveform or number. The `phase` command is similar to the phaseDegUnwrapped command and returns the unwrapped phase in degrees.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)
n_number	Number.

Value Returned

o_waveform	Returns a waveform object if the input argument is a waveform object or returns a family of waveforms if the input argument is a family of waveforms.
n_number	Returns a number if the input argument is a number.
`nil`	Returns `nil` and an error message otherwise.

Example

```
phase( v( "5" ) )
```

Gets the phase of the waveform representing the voltage at net 5. You can also use the `vp` alias to perform the same command, as in `vp( "5" )`.

```
phase( i( "VFB" ) )
```

Gets the phase of the waveform representing the current through the `VFB` component. You can also use the `ip` alias to perform the same command, as in `ip( "VFB" )`.

```
phase( -2.0 ) => 180.0
```

Gets the phase of `-2`.

phaseDeg

```
phaseDeg( {o_waveform | n_number} )
     => o_waveform/n_number/nil
```

Description

Calculates the wrapped phase in degrees of a waveform and returns a waveform.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)
n_number	Number.

Value Returned

o_waveform	Returns a waveform object representing the wrapped phase in degrees of the input waveform. Returns a family of waveforms if the input argument is a family of waveforms.
n_number	Returns a number if the input argument is a number.
`nil`	Returns `nil` and an error message otherwise.

Example

```
phaseDeg( v( "vout" ) )
```

Takes the input waveform, representing the voltage of the `"vout"` net, and returns the waveform object representing the wrapped phase in degrees.

phaseDegUnwrapped

```
phaseDegUnwrapped( {o_waveform | n_number} )
     => o_waveform/n_number/nil
```

Description

Calculates the unwrapped phase in degrees of a waveform and returns a waveform.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)
n_number	Number.

Value Returned

o_waveform	Returns a waveform object representing the unwrapped phase in degrees of the input waveform. Returns a family of waveforms if the input argument is a family of waveforms.
n_number	Returns a number if the input argument is a number.
`nil`	Returns `nil` and an error message otherwise.

Example

```
phaseDegUnwrapped( v( "vout" ) )
```

Takes the input waveform, representing the voltage of the `"vout"` net, and returns the waveform object representing the unwrapped phase in degrees.

phaseMargin

```
phaseMargin( o_waveform )
     => o_waveform/n_value/nil
```

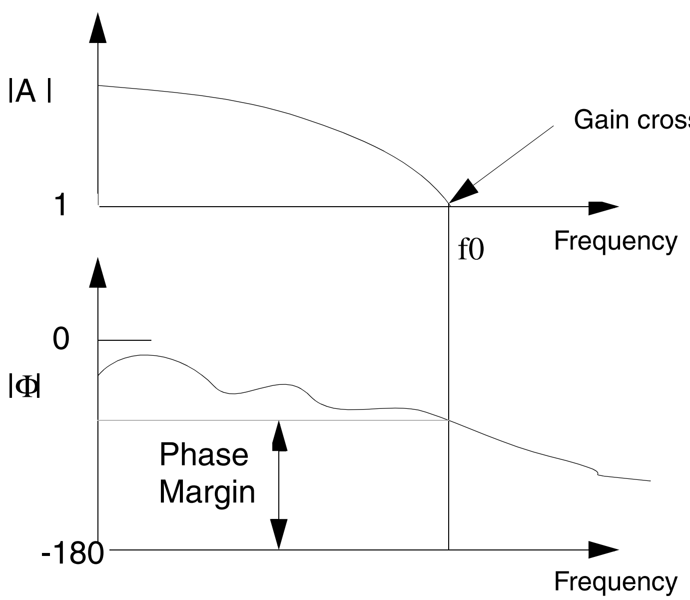
Description

Computes the phase margin of the loop gain of an amplifier.

You supply a waveform representing the loop gain of interest over a sufficiently large frequency range.

phaseMargin(gain) = 180 + phase(value(gain f0))

The phase margin is calculated as the difference between the phase of the gain in degrees at f0 and at -180 degrees. The frequency f0 is the lowest frequency where the gain is 1. For stability, the phase margin must be positive.



Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:XXXXX`.)

Value Returned

o_waveform	Returns a waveform representing the phase margin of the loop gain of an amplifier for a family of waveforms if the input argument is a family of waveforms.
n_value	Returns the value (in degrees) equivalent to the phase margin of the input waveform.
`nil`	Returns `nil` and an error message otherwise.

Example

```
phaseMargin( v( "/OUT" ) )
```

Returns the phase margin for the waveform representing the voltage of the `"/OUT"` net.

phaseRad

```
phaseRad( {o_waveform | n_number} )
     => o_waveform/n_number/nil
```

Description

Calculates the wrapped (discontinuous) phase in radians of a waveform.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)
n_number	Number.

Value Returned

o_waveform	Returns a waveform representing a discontinuous value (in radians) for the phase of the input waveform. Returns a family of waveforms if the input argument is a family of waveforms.
n_number	Returns a number when the input argument is a number.
`nil`	Returns `nil` and an error message otherwise.

Example

```
plot( phaseRad( v( "/OUT" ) ) )
```

Returns the wrapped phase of the waveform representing the voltage of the `"/OUT"` net.

phaseRadUnwrapped

```
phaseRadUnwrapped( o_waveform )
     => o_waveform/nil
```

Description

Calculates the unwrapped (continuous) phase in radians of a waveform and returns a waveform.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)

Value Returned

o_waveform	Returns a waveform representing the unwrapped (continuous) value for the phase of the input waveform in radians. Returns a family of waveforms if the input argument is a family of waveforms.
`nil`	Returns `nil` and an error message otherwise.

Example

```
plot( phaseRadUnwrapped( v( "/OUT" ) )
```

Returns the unwrapped phase of the waveform representing the voltage of the `"/OUT"` net.

PN

```
PN( o_waveform t_crossType n_threshold 1.0 ?windowName t_windowName
     ?smooth x_smooth ?windowsize x_windowsize ?detrending t_detrending)
     ?cohGain f_cohGain )
     => o_waveform/nil
```

Description

Calculates the transient phase noise of the input waveforms in decibels (`dBc/Hz`). Phase noise is defined as the power spectral density of the absolute jitter of an input waveform.

Arguments

o_waveform
: Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*`XXXXX`*.)

t_crossType
: The points at which the curves of the waveform intersect with the threshold. While intersecting, the curve may be either rising or falling.
Valid values: `rising` and `falling`, respectively.
Default `crossType` is `rising`.

t_windowName
: The window type.
Valid values: `'Blackman`, `'Cosine2`, `'Cosine4`, `'ExtCosBell`, `'HalfCycleSine`, `'HalfCycleSine3` or `'HalfCycleSine6`, `'Hamming`, `'Hanning`, `'Kaiser`, `'Parzen`, `'Rectangular`, or `'Triangular`.
Default value: `'Rectangular`

x_smooth
: The Kaiser window smoothing parameter. The `0` value requests no smoothing.
Valid values: 0 <= *x_smooth* <= 15.
Default value: `1`

x_windowsize
: The number of frequency domain points to use in the Fourier analysis. A larger window size results in an expectation operation over fewer samples, which leads to larger variations in the power spectral density. A small window size can smear out sharp steps in the power spectral density that might really be present.
Default value: `256`

t_detrending	The detrending mode to use. Valid values: `'None`, `'mean`, `'Linear` Default value: `'Mean`
f_cohGain	A scaling parameter. A non-zero value scales the power spectral density by 1/(*f_cohGain*). Valid values: *none, default, magnitude*, *dB20*, or *dB10* Default value: `db20`

Value Returned

o_waveform	The power spectral density waveform returned when the command is successful.
`nil`	Returns `nil` when the command fails.

Example

```
PN(v("net9") "rising" 1.0 ?windowName "Rectangular" ?smooth 1 ?windowSize 256
?detrending "Mean" ?cohGain (10**(/20)) )
```

Returns the Phase Noise waveform, `net9`, for the window type rectangular at threshold value `1.0`.

pow

```
pow( {o_waveformBase | n_numberBas} {o_waveformExpn | n_numberExpn} )
     => o_waveform/n_result/nil
```

Description

Takes the exponent of a given waveform or number.

Arguments

o_waveformBase	Waveform object to be used as the base for the expression.
o_waveformExpn	Waveform object to be used as the exponent for the expression.
n_numberBase	Number to be used as the base for the expression.
n_numberExpn	Number to used as the exponent for the expression.

Value Returned

o_waveform	Returns a family of waveforms if one of the input arguments is a family of waveforms or returns a waveform if one of the input arguments is a waveform (and none is a family).
n_result	Returns a number if both the input arguments are numbers.
nil	Returns `nil` and an error message otherwise.

Example

```
pow( average( v( "/net9" ) ) 0.5 )
```

Gets the square root of the average value of the voltage at `"/net9"`.

```
pow( 2 3 )
=> 8
```

Gets the value of 2 to the third power, or 8.

```
pow( -2 2 )
=> 4
```

Gets the value of -2 to the second power.

```
pow( 2.5 -1.2 )
=> 0.3330213
```

Gets the value of 2.5 to the power of -1.2.

prms

```
prms( o_waveform n_from n_to [n_period [n_sfactor]])
     => o_waveform/nil
```

Description

Computes the periodic root mean square of a family of signals for each time point, which is the square root of the periodic average of the square of the input waveform.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like srrWave:XXXXX.).
n_from	Starting numeric value for the range on the X-axis.
n_to	Ending numeric value for the range on the X-axis.
n_period	Numeric value for the period of the input waveform.
n_sfactor	Sampling factor. This can be increased in order to increase the accuracy of the output. Default value: 1

Values Returned

o_waveform	Returns a waveform representing the periodic root mean square of a family of signals.
nil	Returns `nil` and an error message otherwise.

Example

```
prms v("/net8") ?from 1n ?to 20n ?period 2n ?sfactor 1)
```

Returns the value of the periodic root mean square for the family of waveforms representing the voltage of `"/net8"`.

psd

```
psd( o_waveform f_timeStart f_timeEnd x_num ?windowName t_windowName
     ?smooth x_smooth ?cohGain f_cohGain ?windowsize x_windowsize
     ?detrending t_detrending)
     => o_waveformReal/nil
```

Description

Returns an estimate for the power spectral density of *o_waveform*. If *x_windowsize* is not a power of 2, it is forced to the next higher power of 2. If *x_num* is less than *x_windowsize*, *x_num* is forced to *x_windowsize*.

Arguments

o_waveform	Time domain waveform object with units of volts or amps.
f_timeStart	Starting time for the spectral analysis interval. Use this parameter and *f_timeEnd* to exclude part of the interval. For example, you might set these values to discard initial transient data.
f_timeEnd	Ending time for the spectral analysis interval.
x_num	The number of time domain points to use. The maximum frequency in the Fourier analysis is proportional to *x_num* and inversely proportional to the difference between *f_timeStart* and *f_timeEnd*. Default value: `512`
t_windowName	The window to be used for applying the moving window FFT. Valid values: `'Blackman`, `'Cosine2`, `'Cosine4`, `'ExtCosBell`, `'HalfCycleSine`, `'Half3CycleSine` or `'HalfCycleSine3`, `'Half6CycleSine` or `'HalfCycleSine6`,`'Hamming`, `'Hanning`, `'Kaiser`, `'Parzen`, `'Rectangular`, `'Triangle` or `'Triangular`. Default value: `'Hanning`
x_smooth	The Kaiser window smoothing parameter. The `0` value requests no smoothing. Valid values: 0 <= *x_smooth* <= 15. Default value: `1`

f_cohGain A scaling parameter. A non-zero value scales the power spectral density by 1/(*f_cohGain*).
Valid values: 0 < *f_cohGain* < 1 (You can use `1` if you do not want the scaling parameter to be used)
Default value: `1`

x_windowsize The number of frequency domain points to use in the Fourier analysis. A larger window size results in an expectation operation over fewer samples, which leads to larger variations in the power spectral density. A small window size can smear out sharp steps in the power spectral density that might really be present.
Default value: `256`

t_detrending The detrending mode to use.
Valid values: `'mean`, `'linear`, `'none`
Default value: `'none`

The `psd` function works by applying a moving windowed FFT to time-series data. If there is a deterministic trend to the underlying data, you might want to remove the trend before performing the spectral analysis. For example, consider analyzing phase noise in a VCO model. Without the noise, the phase increases more or less linearly with time, so it is appropriate to set the detrending mode to `'linear`. To subtract an average value, set the detrending mode to `'mean`. Where the spectrum of raw data is desired, set the detrending mode to `'none`.

Value Returned

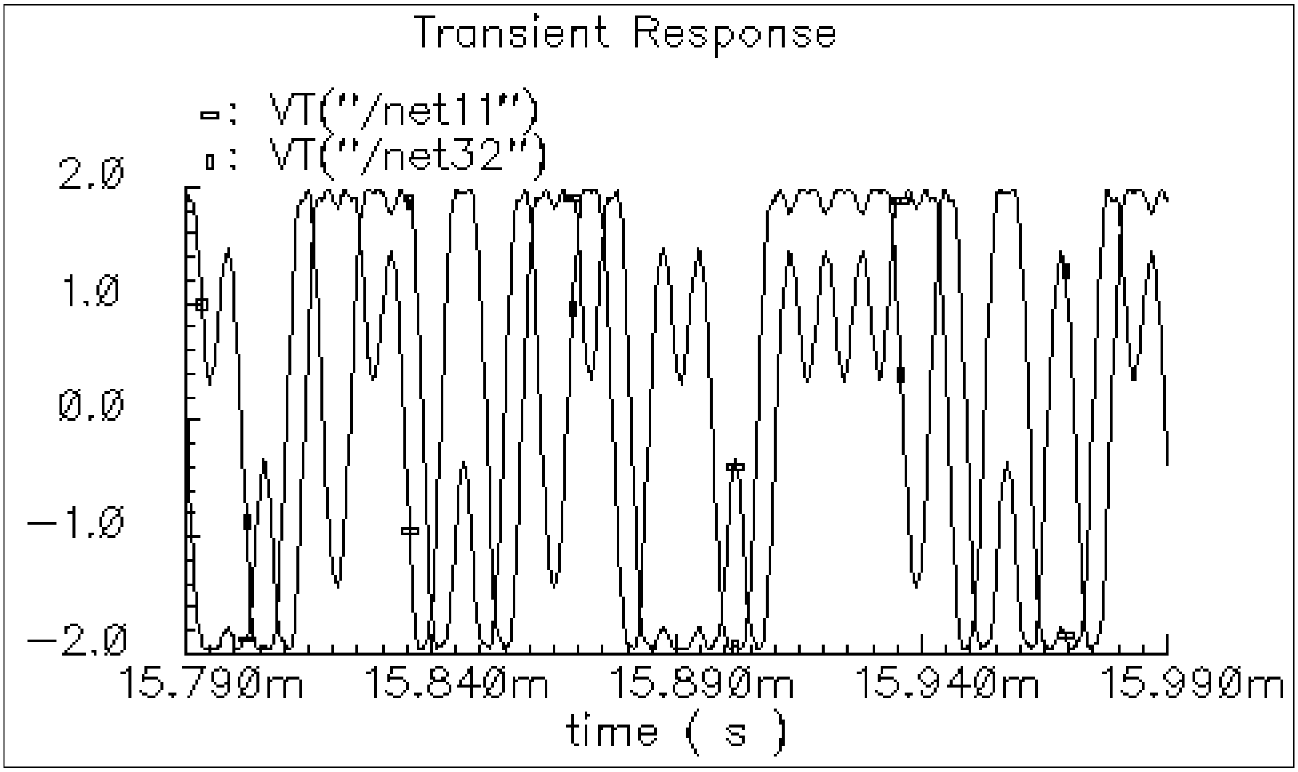
o_waveformReal The power spectral density waveform returned when the command is successful.

`nil` Returns `nil` when the command fails.

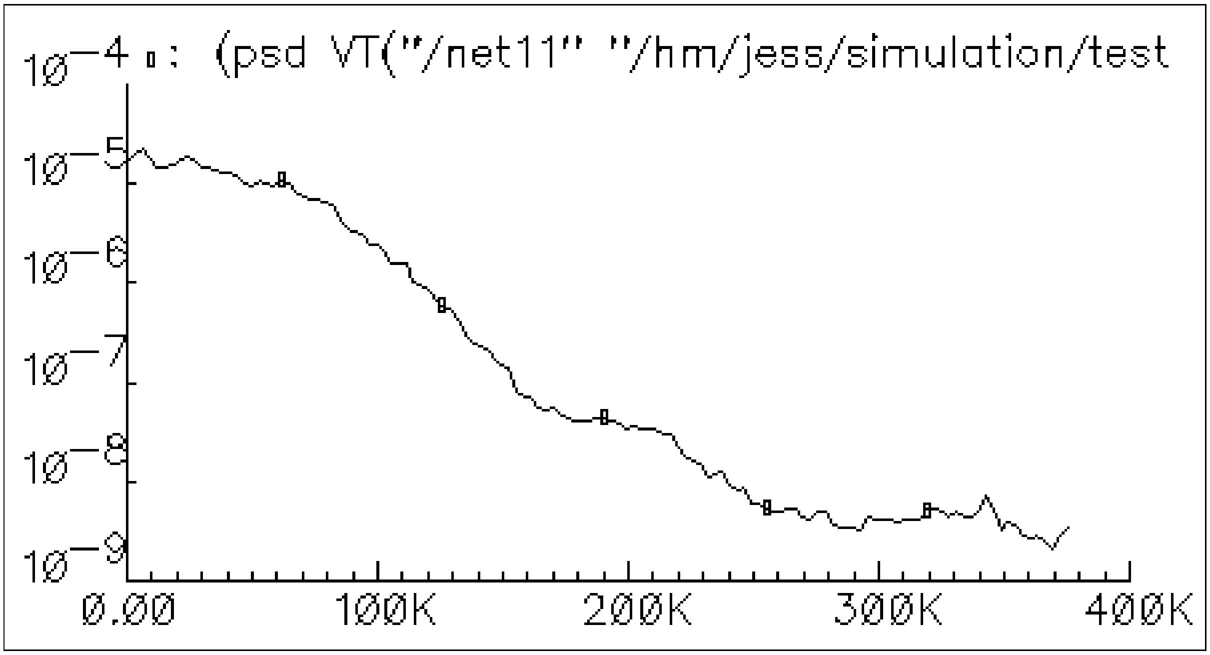
Example

```
psd(VT("/net32" "/hm/test_bench/spectre/schematic"), 0, 16m, 12000,
    ?windowName 'Hanning,?smooth 1, ?windowSize 256,
    ?detrending 'None, ?cohGain 1)
```

Consider applying this command to one of the waveforms in the following illustration.



The result is the following spectrum, which is displayed with a logarithmic vertical scale.



psdbb

```
psdbb( o_waveform1 o_waveform2 f_timeStart f_timeEnd x_num
     ?windowName t_windowName ?smooth x_smooth ?cohGain f_cohGain
     ?windowsize x_windowsize ?detrending t_detrending)
     => o_waveformReal/nil
```

Description

Returns an estimate for the power spectral density of *o_waveform1+j*o_waveform2*. If *x_windowsize* is not a power of 2, it is forced to the next higher power of 2. If *x_num* is less than *x_windowsize*, *x_num* is forced to *x_windowsize*.

Arguments

o_waveform1	Time domain waveform object with units of volts or amps.
o_waveform2	Time domain waveform object with units of volts or amps.
f_timeStart	Starting time for the spectral analysis interval. Use this parameter and *f_timeEnd* to exclude part of the interval. For example, you might set these values to discard initial transient data.
f_timeEnd	Ending time for the spectral analysis interval.
x_num	The number of time domain points to use. The maximum frequency in the Fourier analysis is proportional to *x_num* and inversely proportional to the difference between *f_timeStart* and *f_timeEnd*.
t_windowName	The window to be used for applying the moving window FFT. Valid values: `'Blackman`, `'Cosine2`, `'Cosine4`, `'ExtCosBell`, `'HalfCycleSine`, `'Half3CycleSine` or `'HalfCycleSine3`, `'Half6CycleSine` or `'HalfCycleSine6`,`'Hamming`, `'Hanning`, `'Kaiser`, `'Parzen`, `'Rectangular`, `'Triangle` or `'Triangular`. Default value: `'Hanning`
x_smooth	The Kaiser window smoothing parameter. `0` requests no smoothing. Valid values: 0 <= *x_smooth* <= 15. Default value: `1`

f_cohGain — A scaling parameter. A non-zero value scales the power spectral density by 1/(*f_cohGain*).
Valid values: 0 < *f_cohGain* < 1 (You can use `1` if you do not want the scaling parameter to be used)
Default value: `1`

x_windowsize — The number of frequency domain points to use in the Fourier analysis. A larger window size results in an expectation operation over fewer samples, which leads to larger variations in the power spectral density. A small window size can smear out sharp steps in the power spectral density that might really be present.

t_detrending — The detrending mode to use.
Valid values: `'mean`, `'linear`, `'none`
Default value: `'none`

The `psd` function works by applying a moving windowed FFT to time-series data. If there is a deterministic trend to the underlying data, you might want to remove the trend before performing the spectral analysis. For example, consider analyzing phase noise in a VCO model. Without the noise, the phase increases more or less linearly with time, so it is appropriate to set the detrending mode to `'linear`. To subtract an average value, set the detrending mode to `'mean`. Where the spectrum of raw data is desired, set the detrending mode to `'none`.

Value Returned

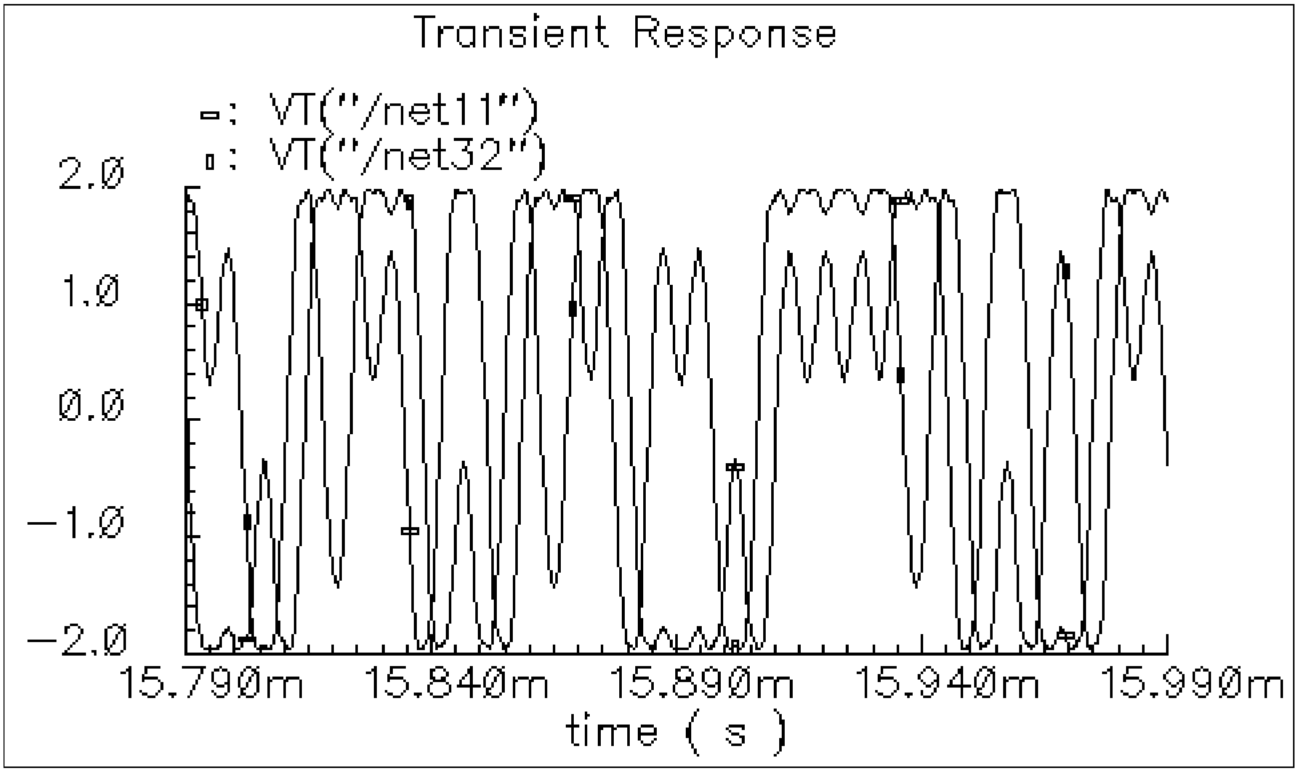
o_waveformReal — The power spectral density waveform returned when the command is successful.

`nil` — Returns `nil` when the command fails.

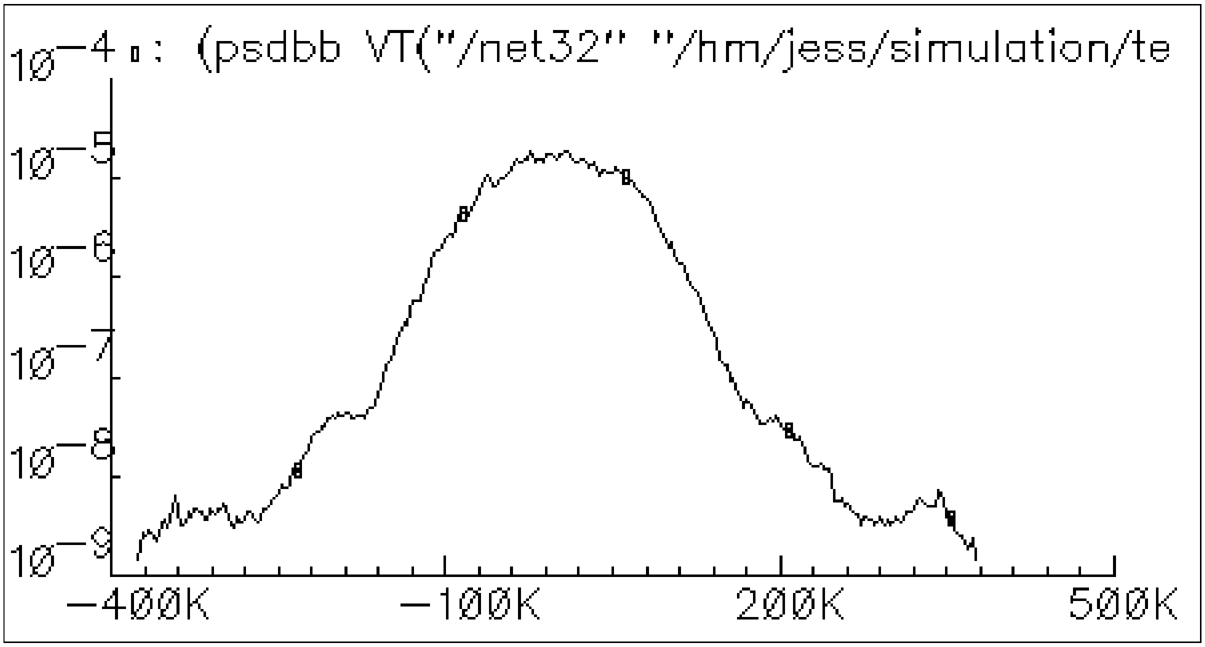
Example

```
psdbb(VT("/net32" "/hm/test_bench/spectre/schematic"),
    VT("/net11" "/hm/test_bench/spectre/schematic"), 0, 16m, 12000,
    ?windowName 'Hanning,?smooth 1, ?windowSize 256,
    ?detrending 'None, ?cohGain 1)
```

Consider applying this command to both of the waveforms in the following illustration.



The result is the following spectrum, which is displayed with a logarithmic vertical scale.



pstddev

```
pstddev( o_waveform n_from n_to [n_period [n_sfactor]])
     => o_waveform/nil
```

Definition

Computes the periodic standard deviation of a family of signals for each time point.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like srrWave:XXXXX.).
n_from	Starting numeric value for the range on the X-axis.
n_to	Ending numeric value for the range on the X-axis.
n_period	Numeric value for the period of the input waveform.
n_sfactor	Sampling factor. This can be increased in order to increase the accuracy of the output. Default value: 1

Values Returned

o_waveform	Returns a waveform representing the periodic standard deviation of a family of signals.
`nil`	Returns `nil` and an error message otherwise.

Example

```
pstddev( v("/net8") ?from 1n ?to 20n ?period 2n ?sfactor 1)
```

Returns the value of the periodic standard deviation for the family of waveforms representing the voltage of `"/net8"`

pzbode

```
pzbode(f_transferGain f_minfrequency f_maxfrequency x_nponits ?poles
     o_waveform1 ?zeros o_wavefoem2)
     => o_waveform/nil
```

Description

Calculates and plots the transfer function of a circuit from pole zero simulation data.

Note: This command also works for the parametric or sweep data.

Arguments

f_transferGain	The transfer gain constant.
f_minfrequency	The minimum frequency for the bode plot.
f_maxfrequency	The maximum frequency for the bode plot.
x_npoints	The frequency interval for the bode plot, in points per decade.
o_waveform1	Poles from the dumped simulation data. Default value: `all`
o_waveform2	Zeros from the dumped simulation data. Default value: `all`

Value Returned

o_waveform	Waveform containing the X and Y points of the transfer function. The scale of the Y-axis will be db20.
`nil`	Returns `nil` and error message otherwise.

Example

```
pzbode( 1.0 1M 1G 20 ?poles complexPoleWave ?zeros complexZeroWave )
```

pzfilter

```
pzfilter( [o_PoleWaveform] [o_ZeroWaveform] [?maxfreq t_maxfreq]
     [?reldist n_relDist] [?absdist n_absdist] [?minq n_minq] [?output_type
     o_output] )
     => o_waveform/nil
```

Description

Returns the filtered Pole and Zero waveforms.

Note: If you do not specify values for *o_PoleWaveform and o_ZeroWaveform* arguments, you should have run pz analysis prior to using this function. This command also works for the parametric or sweep data.

Arguments

o_PoleWaveform	Input Pole waveform (complex points). Default value: Poles of the simulator pz-analysis dump
o_ZeroWaveform	Input Zero waveform (complex points). Default value: Zeros of the simulator pz-analysis dump
t_maxfreq	Maximum frequency. Default value: `1e10`
n_reldist	Relative distance to be considered while filtering. Default value: `0.05`
n_absdist	Absolute distance to be considered while filtering. Default value: `1e-6`
n_minq	Minimum q factor to be allowed while filtering.
o_output	Specifies the type of the output. If this argument is not passed, the output is a family of waves with two child waveforms, representing poles and zeros respectively, with the real component of each waveform as the X values and the imaginary components as the Y values. Valid value: `complexwave`. The output is a family of waves with two child waves, both of which are complex and represent poles and zeros, respectively.

Value Returned

o_waveform	Returns a family (waveform) of Pole and Zero waveforms.
nil	Returns `nil` otherwise.

Example

```
pzfilter( complexPoleWave complexZeroWave )
=> srrWave:175051584
```

Returns a family of filtered Pole and Zero waveforms, which correspond to the sweep values of “Pole” and “Zero”, respectively.

rapidIPNCurves

```
rapidIPNCurves(
     o_result
     t_resultsDir
     n_resistance
     l_args
     )
     => o_waveformReal/nil
```

Description

Plots IPN curves.

Arguments

o_result	Object representing simulation results that can be displayed as a series of points on a grid.
t_resultsDir	Name of the directory where results are saved.
n_resistance	Value of resistance Default value: `50`
l_args	List of arguments to be used by the value function on the results data. Refer to the value function for more details.

Value Returned

o_waveformReal	Returns a waveform.
`nil`	Returns `nil` or an error message otherwise.

Example

```
w2 = rapidIPNCurves("ac-ip3" ?resultsDir "./simulation/amplifier/spectre/
schematic/psf" ?r 50)
```

rapidIIPN

```
rapidIIPN(
     o_result
     t_resultsDir
     n_resistance
     l_args
     )
     => o_waveform/nil
```

Description

Plots the input IPN curves.

Arguments

o_result	Object representing simulation results that can be displayed as a series of points on a grid.
t_resultsDir	Name of the directory where results are saved.
n_resistance	Value of resistance Default value: `50`
l_args	List of arguments to be used by the value function on the results data. Refer to the value function for more details.

Value Returned

o_waveform	Returns a waveform.
`nil`	Returns `nil` or an error message otherwise.

Example

```
rapidIIPN("hbac_ip3")
```

real

```
real( {o_waveform | n_input} )
    => o_waveformReal/n_numberReal/nil
```

Description

Returns the real part of a waveform representing a complex number, or returns the real part of a complex number.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)
n_input	Complex number.

Value Returned

o_waveformReal	Returns a waveform when the input argument is a waveform.
n_numberReal	Returns a number when the input argument is a number.
`nil`	Returns `nil` and an error message otherwise.

Example

```
real( v( "/net8" ) )
```

Returns a waveform representing the real part of the voltage of `"/net8"`. You also can use the `vr` alias to perform the same command, as in `vr( "net8")`.

```
x=complex( -1 -2 ) => complex(-1, -2)
real( x ) => -1.0
```

Creates a variable `x` representing a complex number, and returns the real portion of that complex number.

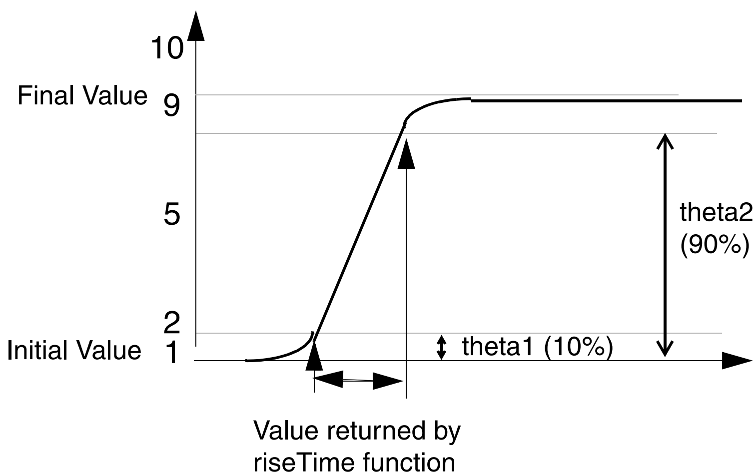
riseTime

```
riseTime( o_waveform n_initVal g_initType n_finalVal g_finalType n_theta1
     n_theta2 [g_multiple [s_Xname][g_histoDisplay][x_noOfHistoBins] ] )
     => o_waveform/n_value/nil
```

Description

Returns the rise time measured between *theta1* (percent low) to *theta2* (percent high) of the difference between the initial value and the final value.

The `riseTime` function can also be used to compute the fall time if *initVal* is higher than *finalVal*.



Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)
n_initVal	Initial value at which to start the computation.
g_initType	Specifies how *n_initVal* functions. Valid values: a non-nil value specifies that the initial value is taken to be the value of the waveform, interpolated at *n_initVal*, and the waveform is clipped from below as follows: *o_waveform* = `clip(` *o_waveform g_initVal* `nil )`

	where `nil` specifies that *n_initVal* is defined by the X value entered. (The command gets the Y value for the specified X value and uses that value for *n_initVal*.)
n_finalVal	Final value at which to end the computation.
g_finalType	Specifies how the *n_finalVal* argument functions. Valid values: a non-nil value specifies that the final value is taken to be the value of the waveform, interpolated at *n_finalVal*, and the waveform is clipped from above, as follows: *o_waveform* = `clip(`*o_waveform* `nil` *n_finalVal*`)` where `nil` specifies that the *n_finalVal* argument is defined by the X value entered. (The command gets the Y value for the specified X value and uses that value for *n_finalVal*.)
n_theta1	Percent low.
n_theta2	Percent high.
g_multiple	An optional boolean argument that takes the value `nil` by default. If set to `t`, the function returns multiple occurrences of the riseTime event.
s_xName	An optional argument that is used only when *g_multiple* is set to `t`. It takes the value `time` by default. It controls the contents of the x vector of the waveform object returned by the function. Valid values: `'time`, `'cycle`
g_histoDisplay	When set to t, returns a waveform that represents the statistical distribution of the riseTime data in the form of a histogram. The height of the bars (bins) in the histogram represents the frequency of the occurrence of values within the range of riseTime data. Valid values: `t nil` Default value: `nil`
x_noOfHistoBins	Denotes the number of bins represented in the histogram representation. Valid values: Any positive integer Default value: `nil`

Note: *g_histoDisplay* and *x_noOfHistoBins* are added for backward compatibility only. It will be deprecated in future releases. Use the histo function for plotting the histogram of the resulting function.

Value Returned

o_waveform	Returns a waveform representing the rise time for a family of waveforms if the input argument is a family of waveforms or if *g_multiple* is set to `t`.
n_value	Returns a value for the rise time if the input is a single waveform.
`nil`	Returns `nil` and an error message otherwise.

Example

```
riseTime( v( "/net8" ) 0 t 2 t 10 90 )
```

Computes the rise time for the waveform representing the voltage of `"/net8"` from `0` to `2`.

For the next example, assume that v is the following sinusoidal waveform:

```
sin( 2 * pi * time)
riseTime( v 0.25 t 0.5 t 10 90)
```

Computes the fall time of the first falling edge from `1` to `0`.

```
riseTime(VT("/out") 0.5 nil 4.5 nil 10 90 t "time") (s)
```

Returns multiple occurrences of riseTime specified against time-points at which each riseTime event occurs.

```
riseTime(VT("/out") 0.5 nil 4.5 nil 10 90 t "cycle") (s)
```

Returns multiple occurrences of riseTime specified against cycle numbers, where a cycle number refers to the n'th occurrence of the riseTime event in the input waveform.

rms

```
rms( o_waveform )
     => o_waveform/n_value/nil
```

Description

Returns the root-mean-square value of a waveform.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)

Value Returned

o_waveform	Returns a waveform representing the root-mean-square value for a family of waveforms if the input argument is a family of waveforms.
n_value	Returns a value for the root-mean-square value for the specified waveform if the input is a single waveform.
`nil`	Returns `nil` and an error message otherwise.

Example

```
rms( v( "/out" ) )
```

Returns the root-mean-square value of the waveform representing the voltage of the `"/out"` net.

rmsNoise

```
rmsNoise(
     n_from
     n_to
     )
     => o_waveform/n_value/nil
```

Description

Computes the integrated root-mean-square noise over the specified bandwidth.

Arguments

n_from	Frequency in hertz that specifies the minimum value for the bandwidth.
n_to	Frequency in hertz that specifies the maximum value for the bandwidth.

Value Returned

o_waveform	Returns a waveform (or a family of waveforms) representing the integrated root-mean-square noise if the data being analyzed is parametric.
n_value	Returns a value for the integrated root-mean-square noise if the data being analyzed is from a single simulation run.
nil	Returns `nil` and an error message otherwise.

Example

```
rmsNoise( 100 100M )
=> 250e-6
```

Computes the integrated root-mean-square noise from `100` to `100M`.

rmsVoltage

```
rmsVoltage(
      t_net
      [t_net1]
      )
      => f_voltage/nil
```

Description

Calculates the root-mean-square voltage between two nets for fast and regular envelop analysis.

Arguments

t_net	Name of the net selected in the schematic.
t_net1	Name of the second net selected in the schematic. This argument is optional. If not specified, the default value is assumed as `gnd`.

Value Returned

f_voltage	Returns a value in terms of voltage.
`nil`	Returns `nil` and an error message otherwise.

Example

```
rmsVoltage( "net1" "!gnd")
=> 120
```

Calculates the root-mean-square voltage between `net1` and `gnd`.

root

```
root( o_waveform n_rootVal x_n )
     => o_waveform/n_value/l_value/nil
```

Description

Returns the *n*th X value at which the Y value equals the specified Y value (*rootVal*).

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)
n_rootVal	Y value of interest.
x_n	Number that specifies which X value to return. If `n` equals `1`, the first X value that crosses over the Y *rootVal* is returned. If `n` equals `2`, the second X value that crosses over the Y *rootVal* is returned, and so on. If you specify a negative integer for `n`, the X values that cross the *rootVal* are counted from right to left (from maximum to minimum). If you specify `n` as `0`, the list of root values is returned.

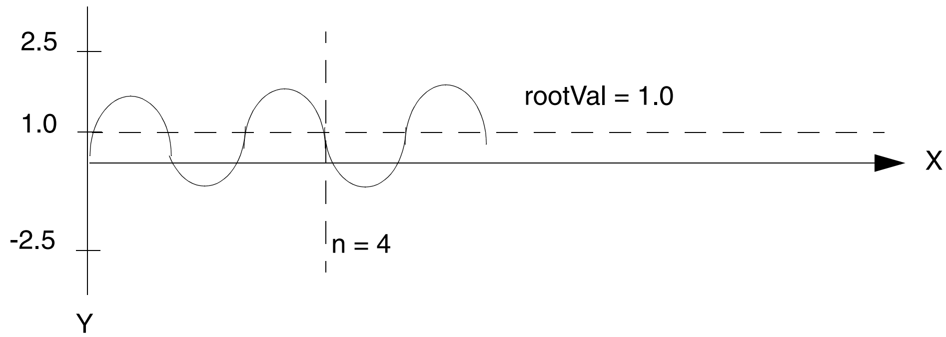
Value Returned

o_waveform	Returns a waveform if the input argument is a family of waveforms.
n_value	Returns an X value when the input argument is a single waveform.
l_value	Returns a list of all the root values when `n` is `0`.
`nil`	Returns `nil` and an error message otherwise.

Example

```
root( v( "vout" ), 1.0, 4 )
```

Returns the X value for the point at which the waveform curve crosses the 1.0 Y value for the fourth time.



rshift

```
rshift( o_waveform n_delta )
     => o_waveform/nil
```

Description

Shifts the waveform to the right by the *n_delta* value.

This command is the inverse of the `lshift` command.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)
n_delta	Value by which the waveform is to be shifted.

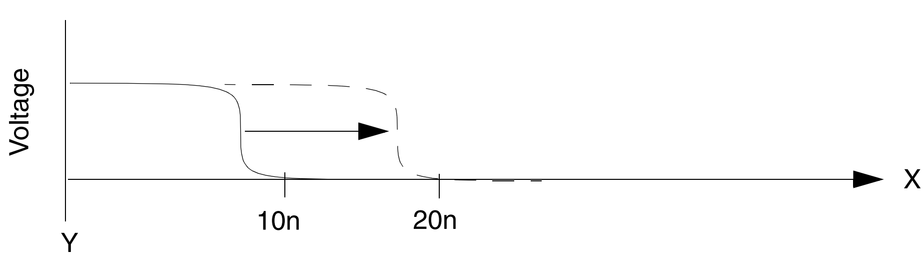
Value Returned

o_waveform	Returns a waveform object. Returns a family of waveforms if the input argument is a family of waveforms.
`nil`	Returns `nil` and an error message otherwise.

Example

```
rshift( v( "vout" ) ) 10n )
```

Shifts the waveform representing the voltage through the `"vout"` net to the right by `10n`.



sample

```
sample( o_waveform n_from n_to t_type n_by )
     => o_waveform/n_number/nil
```

Description

Samples a waveform at the specified interval.

You can use this function to reduce the time it takes to plot waveforms that have many data points. If you sample a waveform beyond its range, you get the final value of the waveform. You can use this function to demodulate a signal. Consider an AM modulated sine wave. Assume the carrier frequency is 1 GHz, and the modulation frequency is 1 MHz. If the waveform is sampled every 1 ns, the resulting signal is cleanly demodulated (the 1 GHz carrier is completely eliminated by the sampling).

Note: The function can be used to sample both a waveform object as well as a family of waveforms. If the family is of dimension *m*, the arguments *n_from*, *n_to*, and *n_by* would be of dimension *m-1*.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)
n_from	Starting value for the sampling.
n_to	Ending value for the sampling.
t_type	Type of the sampling. Valid values: `"linear"` or `"log"`
n_by	Interval at which to sample.

Value Returned

o_waveform	Returns a waveform representing the sampling you specified.
n_number	Returns a number if the output contains only one point.
`nil`	Returns `nil` and an error message otherwise.

Example

```
sample( v( "vout" ) 0 50n "linear" 0.1n )
```

Takes a linear sample of the waveform representing the voltage of the `"vout"` net.

```
sample( v( "vout" ) 0 100m "log" 10 )
```

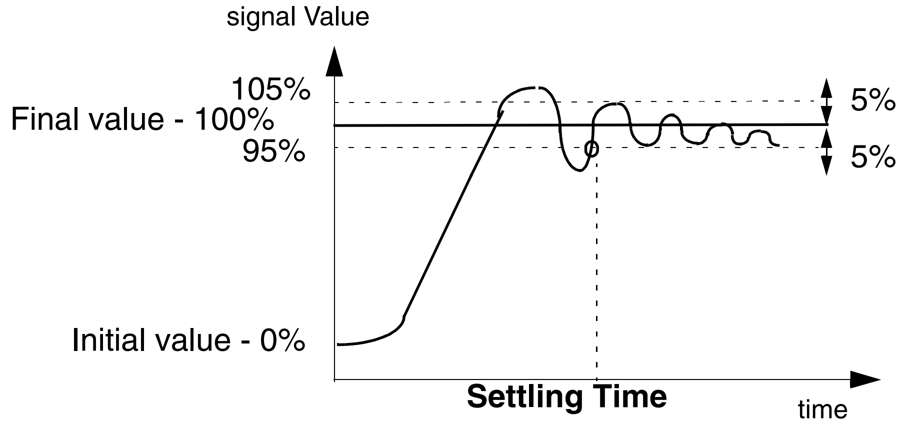
Takes a logarithmic sample of the waveform representing the voltage of the `"vout"` net.

settlingTime

```
settlingTime( o_waveform n_initVal g_initType n_finalVal g_finalType n_theta
     [g_multiple [s_Xname]] )
     => o_waveform/n_value/nil
```

Description

The settling time is the time by which the signal settles within the specified Percent of step (theta) of the difference between the Final Value and Initial Value from the Final Value.



Note: The above graph represents the `Initial value` of the signal as `0%` and `Final value` as `100%`. The `Percent of Step` is taken as `5%`.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)
n_initVal	Initial value at which to start the computation.
g_initType	Specifies whether the values entered are X values or Y values. Valid values: `t` specifies that *initVal* is defined by the X value entered; `nil` specifies that *initVal* is defined by the Y value entered

n_finalVal	Final value at which to start the computation.
g_finalType	Specifies whether the values entered are X values or Y values. Valid values: `t` specifies that *finalVal* is defined by the X value entered; `nil` specifies that *finalVal* is defined by the Y value entered
n_theta	Percent of the total step.
g_multiple	An optional boolean argument that takes the value `nil` by default. If set to `t`, the function returns multiple occurrences of the settlingTime event.
s_xName	An optional argument that is used only when *g_multiple* is set to `t`. It takes the value `time` by default. It controls the contents of the x vector of the waveform object returned by the function. Valid values: `'time`, `'cycle`

Value Returned

o_waveform	Returns a waveform representing the settling time for a family of waveforms if the input argument is a family of waveforms or if *g_multiple* is set to `t`.
n_value	Returns a value for the settling time for the specified waveform if the input is a single waveform.
`nil`	Returns `nil` and an error message otherwise.

Example

```
settlingTime( v("/out" ) 0 t 2 t 90 )
```

Computes the time required for the waveform representing the voltage of the `"/out"` net to settle within 90 percent of the step from 0 to 2.

```
settlingTime(VT("/out") 0.5 nil 4.95 nil 5 t "time") (s)
```

Returns multiple occurrences of settlingTime specified against time-points at which each settlingTime event occurs.

```
settlingTime(VT("/out") 0.5 nil 4.95 nil 5 t "cycle") (s)
```

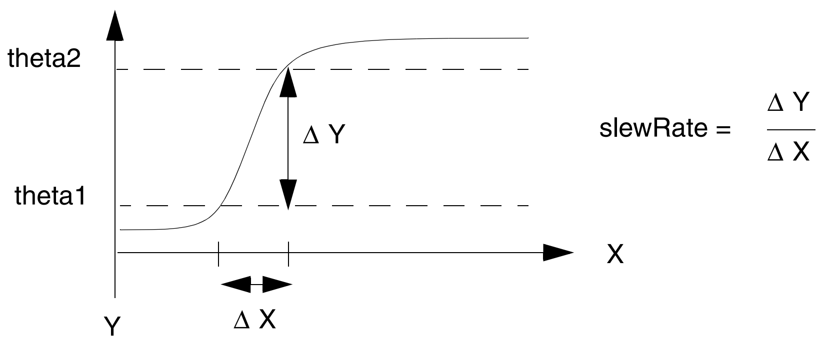
Returns multiple occurrences of settlingTime specified against cycle numbers, where a cycle number refers to the n'th occurrence of the settlingTime event in the input waveform.

slewRate

```
slewRate( o_waveform n_initVal g_initType n_finalVal g_finalType n_theta1
     n_theta2 [g_multiple [s_Xname]][g_histoDisplay][x_noOfHistoBins] )
     => o_waveform/n_value/nil
```

Description

Computes the average rate at which an expression changes from *theta1* (percent low) to *theta2* (percent high) of the difference between the initial value and final value.



Arguments

o_waveform — Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)

n_initVal — Initial X-axis value at which to start the computation.

g_initType — Specifies whether the values entered are X values or Y values. Valid values: `t` specifies that *initVal* is defined by the X value entered; `nil` specifies that *initVal* is defined by the Y value entered

n_finalVal — Final value at which to end the computation.

g_finalType — Specifies whether the values entered are X values or Y values. Valid values: `t` specifies that *finalVal* is defined by the X value entered; `nil` specifies that *finalVal* is defined by the Y value entered

n_theta1 — Percent low (percentage of the total step).

n_theta2	Percent high (percentage of the total step).
g_multiple	An optional boolean argument that takes the value `nil` by default. If set to `t`, the function returns multiple occurrences of the slewRate event.
s_xName	An optional argument that is used only when *g_multiple* is set to `t`. It takes the value `time` by default. It controls the contents of the x vector of the waveform object returned by the function. Valid values: `'time`, `'cycle`
g_histoDisplay	When set to t, returns a waveform that represents the statistical distribution of the riseTime data in the form of a histogram. The height of the bars (bins) in the histogram represents the frequency of the occurrence of values within the range of riseTime data. Valid values: `t nil` Default value: `nil`
x_noOfHistoBins	Denotes the number of bins represented in the histogram representation. Valid values: Any positive integer Default value: `nil`

Note: *g_histoDisplay* and *x_noOfHistoBins* are added for backward compatibility only. It will be deprecated in future releases. Use the histo function for plotting the histogram of the resulting function.

Value Returned

o_waveform	Returns a waveform representing the slew rate for a family of waveforms if the input argument is a family of waveforms or if *g_multiple* is set to `t`.
n_value	Returns a value for the slew rate for the specified waveform if the input is a single waveform.
`nil`	Returns `nil` or an error message otherwise.

Example

```
slewRate( v( "vout" ) 10n t 30n t 10 90 )
```

Computes the slew rate for the waveform representing the voltage of the `"vout"` net from `10n` to `30n`.

```
slewRate( v( "vout" ) 0 nil 10 nil 5 95 )
```

Computes the slew rate for the waveform representing the voltage of the `"vout"` net from 0 to 10. In this example, the initial value and final value are entered as Y values.

```
slewRate(VT("/out") 0.5 nil 4.5 nil 10 90 t 'time)
```

Return multiple occurrences of slewRate values, computed at different time-points.

```
slewRate(VT("/out") 0.5 nil 4.5 nil 10 90 t 'cycle)
```

Returns multiple occurrences of slewRate values specified against cycle numbers (where cycle number refers to the n'th occurrence of slewRate computation).

spectralPower

```
spectralPower( o_current o_voltage )
     => o_power/nil
```

Description

Returns the spectral power given the spectral current and voltage.

To obtain a list of the harmonic frequencies, use `harmonicList`.

Arguments

o_current	Waveform representing the current. The current can be obtained by calling the `i` data access function for the desired terminal.
o_voltage	Waveform representing the voltage. The voltage can be obtained by calling the `v` data access function for the desired net. To obtain meaningful results, the terminal used to obtain the current must be a member of the net used to obtain the voltage.

Value Returned

o_power	Waveform representing the power of the net.
`nil`	Returns `nil` if there is an error.

Example

```
plot(db10(spectralPower(i("/PORT0/PLUS") v("/net28"))))
```

Plots power of the output `"/net28"`. `"/PORT0/PLUS"` is a member of `"/net28"`.

spectrumMeas

```
spectrumMeas( o_waveform n_from n_to x_numSamples x_noiseBins n_startFreq
     n_endFreq t_windowName n_adcSpan t_measType )
     => o_spectrumWaveform/g_value/nil
```

Description

Calculates Signal-to-Noise-and-Distortion Ratio (SINAD), Spurious Free Dynamic Range (SFDR), Effective Number of Bits (ENOB), and Signal-to-Noise Ratio (without distortion) by using discrete fourier transform of any given input signal.

The spectrum measure is used for characterizing A-to-D converters and is typically supported for transient simulation data.

Arguments

o_waveform — Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.).

n_from — The X-axis start value of the portion of input *o_waveform* to be used for FFT and subsequent calculations.

n_to — The X-axis end value of the portion of input *o_waveform* to be used for FFT and subsequent calcu

x_numSamples — Optional number of sampled points used for the FFT.
Valid values: Any integer power of two greater than zero. For a value that is not a power of two, the function rounds it up to the next closest power of two.
Default value: Number of data points in the signal *o_waveform*.

x_noiseBins — Optional number of noise bins, where the size of one bin is the reciprocal of the data window width. For example, 1 ms of transient data creates a bin size of 1 kHz.
Valid values: Any integer power of two greater than or equal to zero.
Default value: 0, implying that no signal is spilling into the bins.
A frequency band of bin-size times the number of bins is calculated and adjusted as a function of the selected window.
Frequency components in each band to the left and right of the

	fundamental or the harmonics are set to zero and do not contribute to any output result.
n_startFreq	Optional lower limit of frequency range for the spectrum measures. Default value: First frequency point of the FFT.
n_endFreq	Optional upper limit of frequency range for the spectrum measures. Default value: Last frequency point of the FFT.
t_windowName	Optional windowing function applied to *o_waveform*. Valid values: `Blackman`, `Cosine2`, `Cosine4`, `ExtCosBell`, `HalfCycleSine`, `HalfCycleSine3`, `HalfCycleSine6`, `Hamming`, `Kaiser`, `Parzen`, `Rectangular`, and `Triangular`. Default value: `Rectangular`.
n_adcSpan	Optional full-scale span, ignoring any DC offsets. This is used in ENOB calculation. Valid values: Any floating point number. Default value: If *n_adcSpan* is not specified or is `nil`, it is assumed to be `0` and is taken to be the peak-to-peak value of the fundamental.
t_measType	Result specifier. Valid values: `sinad`, `sfdr(db)`, `enob`, and `snhr`.

Value Returned

o_spectrumWaveform	Returns a waveform of spectrum measures.
g_value	Returns the spectrum measure specified by the *t_measType* argument.
`nil`	Returns `nil` and an error message otherwise.

Example

```
spectrumMeas( VT("/vcoOut") 1K nil 1K 10G "Rectangular" nil "snhr")
=> -4.948
```

Returns the value of the spectrum measure `snhr`, as specified by the `spectrumMeas` function.

spectrumMeasurement

```
spectrumMeasurement( o_waveform g_isTimeWave n_from n_to x_numSamples
     n_startFreq n_endFreq x_signalBins t_windowName n_satLvl
     g_isNoiseAnalysis x_noOfHarmonics t_measType )
     => g_value/nil
```

Description

Calculates Signal-to-Noise-and-Distortion Ratio (SINAD), Spurious Free Dynamic Range (SFDR), Effective Number of Bits (ENOB), and Signal-to-Noise Ratio (without distortion) by using Fast Fourier Transform (FFT) of any given input signal.

The spectrum measure is used for characterizing A-to-D converters and is typically supported for transient simulation data.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)
g_isTimeWave	Boolean that specifies whether the input wave type is time domain waveform or frequency domain waveform.
n_from	The X-axis start value of the portion of input *o_waveform* to be used for FFT and subsequent calculations.
n_to	The X-axis end value of the portion of input *o_waveform* to be used for FFT and subsequent calculations.
x_numSamples	Number of sampled points used for the FFT. Valid values: Any integer power of two greater than zero. For a value that is not a power of two, the function rounds it up to the next closest power of two. Default value: Number of data points in the *Signal*.
n_startFreq	Lower limit of frequency range for the spectrum measures. Default value: First frequency point of the FFT.
n_endFreq	Upper limit of frequency range for the spectrum measures. Default value: Last frequency point of the FFT.

x_signalBins — Number of signal bins. When you select a window type, this field displays the default number of bins for the selected window type. For example, if you select the *Window Type* as `Kaiser` that has two signal bins, this field displays `2`. You can increase the number of signal bins to up to half the value of the sample count. For example, if the sample count is `16` for the window type `Kaiser`, you can increase the signal bin count in the *Signal Bins* field up to `8`. You cannot decrease the displayed signal bin value.
Valid values: `0` to `99`.
Default value: `0`.

t_windowName — Windowing function applied to *o_wave* while applying the FFT for measurement calculations.
Valid values: `Blackman`, `Cosine2`, `Cosine4`, `ExtCosBell`, `HalfCycleSine`, `HalfCycleSine3`, `HalfCycleSine6`, `Hanning`, `Hamming`, `Kaiser`, `Parzen`, `Rectangular`, and `Triangular`.
Default value: `Rectangular`.

n_satLvl — Peak saturation level of the FFT waveform. Magnitude of the FFT wave is divided by the Peak Sat Level before using it in calculations. Peak sat level is the full-scale span ignoring any DC offsets and used in ENOB calculation.
Valid values: Any floating point number.
Default value: `0`

g_isNoiseAnalysis — Boolean that specifies whether the analysis type is `Signal Analysis` or `Noise Analysis`.

x_noOfHarmonics — Number of harmonics for the waveform that you want to plot. For example, If this variable is `n`, where `n` should be greater than 1 and the fundamental frequency is harmonic `1`, the `n` harmonics are considered for the harmonic power calculation. The signal bins are used for calculating the harmonic power. For example, to calculate the total harmonic distortion (THD), if you set the *Harmonics* value to `n`, where `n` is greater than `1`, and the fundamental frequency is harmonic 1, the number of harmonics used to calculate THD is `2`,...,`n`. If `n=3`, the `2`nd and `3`rd harmonics are used to calculate THD.

t_measType	Result specifier. Valid values: `sinad`, `sfdr(db)`, `enob`, and `snhr`. Default value: `sinad`

Value Returned

g_value	Returns the spectrum measure specified by the *t_measType* argument.
`nil`	Returns `nil` and an error message otherwise.

Example

```
spectrumMeasurement(v("out" ?result "tran") 0 10s 10 0 0 nil "Rectangular" 0
"sinad")
```

Returns the value of the spectrum measure `sinad`, as specified by the `spectrumMeasurement` function.

Additional Information

When you send the computed measurement values from the Spectrum toolbox to ADE Outputs and create an expression for them using ADE, the `spectrumMeasurement` function is used in the expression. For more information about Spectrum toolbox, see Spectrum in *Virtuoso Visualization and Analysis XL User Guide*.

The `spectrumMeas` function uses the same algorithm to calculate measurement values as that of the `spectrumMeasurement` SKILL function. The following table displays the mapping in the arguments for `spectrumMeas` and `spectrumMeasurement` functions:

spectrumMeas	spectrumMeasurement	Description
waveform	*waveform*	Specifies the waveform object.
NA	*isTimeWave*	This argument is available only in `spectrumMeasurement` function. The value of this argument is `nil` if the waveform sweep vector is of frequency domain, and the value is `t` if it is of time domain. In `spectrumMeas` function, internally the unit of X-Vector is checked for `Hz` to know whether it is frequency domain or not.

spectrumMeas	spectrumMeasurement	Description
from	*from*	The X-axis start value of the portion of input *o_waveform* to be used for FFT and subsequent calculations.
to	*to*	The X-axis end value of the portion of input *o_waveform* to be used for FFT and subsequent calculations.
numSamples	*numSamples*	Number of sampled points used for the FFT. Valid values: Any integer power of two greater than zero. For a value that is not a power of two, the function rounds it up to the next closest power of two. Default value: Number of data points in the *Signal*.
noiseBins	*signalBins*	In `spectrumMeas`, *Number of Noise bins* is the number of noise bins where the size of one bin is the reciprocal of the data window width. For example, 1 ms of transient data creates a bin size of 1 kHz. Valid values: Any integer power of two greater than or equal to zero. Default value: `0`, implying that no signal is spilling into the bins In `spectrumMeasurement`, *signalBins* specifies the number of signal bins. When you select a window type, this field displays the default number of bins for the selected window type. Default value: `0` to indicate the rectangular window type.
startFreq	*startFreq*	Lower limit of frequency range for the spectrum measures. Default value: First frequency point of the FFT.
endFreq	*endFreq*	Upper limit of frequency range for the spectrum measures. Default value: Last frequency point of the FFT.

spectrumMeas	spectrumMeasurement	Description
windowName	*windowName*	Windowing function applied to *o_wave* while applying the FFT for measurement calculations. Valid values: Blackman, Cosine2, Cosine4, ExtCosBell, HalfCycleSine, HalfCycleSine3, HalfCycleSine6, Hanning, Hamming, Kaiser, Parzen, Rectangular, and Triangular. Default value: Rectangular
adcSpan	*satLvl*	In spectrumMeas, *ADC Span* is the full-scale span ignoring any DC offsets. This is used in ENOB calculation. Valid values: Any floating point number. In spectrumMeasurement, *satLvl* specifies the peak saturation level of the FFT waveform. Magnitude of the FFT wave is divided by the Peak Sat Level before using it in calculations. Peak sat level is the full-scale span ignoring any DC offsets and used in ENOB calculation. Valid values: Any floating point number.
NA	*isNoiseAnalysis*	This argument is present only in the spectrumMeasurement function. It specifies whether the analysis type is Noise Analysis.
NA	*noOfHarmonics*	This argument is available only in spectrumMeasurement function. This specifies the number of harmonics for the waveform that you want to plot. Default value: 1

spectrumMeas	spectrumMeasurement	Description
measType	*measType*	Result specifier. This argument is common for both the functions, but includes the following differences: ■ `sfdr(db)` of `spectrumMeas` is same as `sfdr` of `spectrumMeasurement` or Spectrum assistant ■ `snhr` of `spectrumMeas` is same as `snr` of `spectrumMeasurement` or Spectrum assistant. ■ `spectrumMeas` supports the following measurements—`sinad`, `sfdr(db)`, v, `enob`, `thd`. However, `spectrumMeasurement` supports more measurements in addition to the measurements supported by `spectrumMeas`.

ssb

```
ssb( o_s11 o_s12 o_s21 o_s22 g_frequency )
     => o_waveform/nil
```

Description

Computes the source stability circles.

Arguments

o_s11	Waveform object representing s11.
o_s12	Waveform object representing s12.
o_s21	Waveform object representing s21.
o_s22	Waveform object representing s22.
g_frequency	Frequency. It can be specified as a scalar or a linear range. The frequency is swept if it is specified as a linear range. The linear range is specified as a list with three values: the start of the range, the end of the range, and the increment. For example, `list(100M 1G 100M)` specifies a linear range with the following values: `{ 100M, 200M, 300M, 400M, 500M, 600M, 700M, 800M, 900M, 1G }` In that case, a source stability circle is calculated at each one of the 10 frequencies.

Value Returned

o_waveform	Waveform object representing the source stability circles.
`nil`	Returns `nil` and an error message otherwise.

Example

```
plot(ssb(s11 s12 s21 s22 list(800M 1G 100M)))
```

stddev

```
stddev( o_waveform )
     => n_stddev/o_waveformStddev/nil
```

Description

Computes the standard deviation of a waveform (or a family of waveforms) over its entire range. Standard deviation (stddev) is defined as the square-root of the variance where variance is the integral of the square of the difference of the expression f(x) from average (f(x)), divided by the range of x.

For example, if `y=f(x)`

$$stddev(\mathrm{y}) = \sqrt{\frac{\int_{from}^{to} (y - average(y))^2}{to - from}}$$

Arguments

o_waveform	Waveform object or family of waveforms representing simulation results that can be displayed as a series of points. (A waveform object identifier looks like this: srrWave:XXXXX)

Value Returned

n_stddev	Returns a number representing the standard deviation value of the input waveform.
o_waveformStddev	Returns a waveform representing the average value if the input is a family of waveforms.
`nil`	Returns `nil` or an error message.

Example

```
stddev( v( "/net9" ) )
```

Gets the standard deviation of the voltage (Y-axis value) of `/net9` over the entire time range specified in the simulation analysis.

tangent

```
tangent( o_waveform [ ?x n_x ] [ ?y n_y ] [ ?slope n_slope ] )
     => o_waveform/nil
```

Description

Returns the tangent to a waveform through the point (*n_x*, *n_y*) with the given slope.

Arguments

o_waveform	Waveform object representing the wave.
n_x	X coordinate of the point. The default value is the X coordinate of the first point on the wave.
n_y	Y coordinate of the point. The default value is the Y coordinate at the given or default X coordinate.
n_slope	Slope of the line. Default value: `1.0`

Value Returned

o_waveform	Wave object representing the line.
`nil`	Returns `nil` if there is an error.

Example

```
refLine
=> tangent(refWave ?x -25 ?slope 1.0)
```

thd

```
thd( o_waveform n_from n_to x_num n_fund)
     => o_waveform/n_thdValue/nil
```

Description

The thd function computes the percentage of total harmonic content of a signal with respect to the fundamental frequency expressed as a voltage percentage.

The computation uses the dft function. Assume that the *dft* function returns complex coefficients A_0, A_1..., A_f , ... Please note that fundamental frequency ***f is the frequency contributing to the largest power in the signal.*** A_0 is the complex coefficient for the DC component and A_i is the complex coefficient for the ***i***th harmonic where $i \neq 0,\ f$. Then, total harmonic distortion is computed as:

$$\frac{\sqrt{\sum_{i = 1,\ i \neq 0,\ f} |A_i|^2}}{|A_f|} \times 100\%$$

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)
n_from	Starting time for the DFT sample window.
n_to	Ending time for the DFT sample window.
x_num	Number of timepoints. If *x_num* is not a power of 2, it is forced to be the next higher power of 2.
n_fund	Fundamental Frequency of the signal. If it is nil or zero then the non-zero frequency contributing to the largest power in the signal is used as the fundamental frequency. Otherwise, the harmonic

frequency nearest to its value is used as the fundamental frequency.

Value Returned

o_waveform	Returns a waveform representing the absolute value of the total harmonic distortion if the input argument is a family of waveforms.
n_thdValue	Returns the absolute value of the total harmonic distortion of the input waveform.
`nil`	Returns `nil` and an error message otherwise.

Example

```
plot( thd( v( "/net8" ) 10u 20m 64 0 ) )
```

Computes the absolute value of the total harmonic distortion for the waveform representing the voltage of "/net8". The computation is done from 10u to 20m with 64 time points using the non-zero frequency contributing to the largest power in the signal as the fundamental frequency. The resulting waveform is plotted.

```
plot( thd( v( "/net8" ) 10u 20m 64 90 ) )
```

Computes the absolute value of the total harmonic distortion for the waveform representing the voltage of "/net8". The computation is done from 10u to 20m with 64 timepoints using a harmonic frequency, whose absolute difference w.r.t 90 is minimum, as the fundamental frequency. The resulting waveform is plotted.

unityGainFreq

```
unityGainFreq( o_gainFreqWaveform )
     => n_frequency/nil
```

Description

Computes and reports the frequency at which the gain is unity.

Arguments

o_gainFreqWaveform	Gain frequency waveform.

Value Returned

n_frequency	Returns a scalar value representing the frequency at which the gain of the input waveform is unity.
nil	Returns nil otherwise.

Example

```
unityGainFrequency( VF("/out") )
```

value

```
value( o_waveform [s_name] g_value ?period n_period [g_multiple [s_Xname]]
     [g_histoDisplay][x_noOfHistoBins])
     => o_waveform/g_value/nil
```

Description

Returns the Y value of a waveform for a given X value.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)
s_name	The name of the innermost or outermost sweep variable. If the sweep variable name is not supplied, the innermost sweep variable is used.
g_value	Value (X value) at which to provide the Y value. If a string has been defined for a value or set of values, the string may be used instead of the value.
n_period	The interval or period after which the value needs to be computed.
g_multiple	An optional boolean argument that takes the value `nil` by default. If set to `t`, the function returns multiple occurrences of the interpolated value.
s_xName	An optional argument that is used only when *g_multiple* is set to `t`. It takes the value `time` by default. It controls the contents of the x vector of the waveform object returned by the function. Valid values: `'time`, `'cycle`
g_histoDisplay	When set to t, returns a waveform that represents the statistical distribution of the riseTime data in the form of a histogram. The height of the bars (bins) in the histogram represents the frequency of the occurrence of values within the range of riseTime data.

	Valid values: `t` `nil` Default value: `nil`
`x_noOfHistoBins`	Denotes the number of bins represented in the histogram representation. Valid values: Any positive integer Default value: `nil`

Note: *`g_histoDisplay`* and *`x_noOfHistoBins`* are added for backward compatibility only. It will be deprecated in future releases. Use the histo function for plotting the histogram of the resulting function.

For the simplest calls to the function, which specify only the given waveform (*`o_waveform`*) and the X value (*`g_value`*), the given waveform can be a family of waveforms. If the family is of dimension *`m`*, *`g_value`* can be either of dimension *`m-1`* or a scalar. If *`g_value`* is scalar, the function returns the Y value of all the components of the family at the specified *`g_value`*.

Value Returned

`o_waveform`	Returns a waveform or a family of waveforms if the input argument is a family of waveforms or if values are expected at multiple points.
`g_value`	Returns the Y value if the input argument is a single waveform. For parametric sweeps, the value might be a waveform that can be printed with the `ocnPrint` command.
`nil`	Returns `nil` and an error message if the value cannot be printed.

Example

```
value( v( "/net18" ) 4.428e-05 )
```

Prints the value of `"/net18"` at `time=4.428e-05`. This is a parametric sweep of temperature over time.

```
value( v( "/OUT" )'TEMPDC 20.0 )
```

Returns `srrWave:`*`XXXXX`*, indicating that the result is a waveform.

```
print( value( v( "/OUT" )'TEMPDC 20.0 ) )
```

Prints the value of `v( "/OUT" )` at every time point for `TEMPDC=20`.

```
print( value( v( "/OUT" ) 200n ?period 100n) )
```

Prints the value of `v( "/OUT" )` at `200n`, `300n` and so on at intervals of `100n` until the end of the waveform.

```
value(VT("/out") 2e-07 ?period 2e-07 ?xName "time") (V)
```

Returns multiple occurrences of the value specified against time-points at which each interpolated value occurs.

```
value(VT("/out") 2e-07 ?period 2e-07 ?xName "cycle") (V)
```

Returns multiple occurrences of value specified against cycle numbers, where a cycle number refers to the n'th occurrence of the value event in the input waveform.

xmax

```
xmax( o_waveform x_numberOfPeaks )
     => o_waveform/g_value/l_value/nil
```

Description

Computes the value of the independent variable (X) at which the Y value attains its maximum value.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)
x_numberOfPeaks	Specifies the *n*th X value corresponding to the maximum Y value. For example, if *x_numberOfPeaks* is `3`, the X value corresponding to the third maximum Y value is returned. If you specify a negative integer for *x_numberOfPeaks*, the X values are counted from right to left (from maximum to minimum). If *x_numberOfPeaks* is `0`, `xmax` returns a list of X locations.

Value Returned

o_waveform	Returns a waveform (or a family of waveforms) if the input argument is a family of waveforms.
g_value	Returns the X value corresponding to the peak specified with *x_numberOfPeaks* if the input argument is a single waveform.
l_value	Returns a list of X locations when *x_numberOfPeaks* is `0` and the input argument is a single waveform.
`nil`	Returns `nil` and an error message otherwise.

Example

```
xmax( v( "/net9" ) 1 )
```

Gets the time value (X-axis value) at which the voltage of `"/net9"` attains its first peak value.

```
xmax( v( "/net9" ) 0 )
```

Gets the list of time values (X-axis values) at which the voltage of `"/net9"` attains each of its peak values.

xmin

```
xmin( o_waveform x_numberOfValleys )
    => o_waveform/g_value/l_value/nil
```

Description

Computes the value of the independent variable (X) at which the Y value attains its minimum value.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)
x_numberOfValleys	Specifies the *n*th X value corresponding to the minimum Y value. For example, if *x_numberOfValleys* is `3`, the X value corresponding to the third minimum Y value is returned. If you specify a negative integer for *x_numberOfValleys*, the X-values are counted from right to left (from maximum to minimum). If *x_numberOfValleys* is `0`, `xmin` returns a list of X locations.

Value Returned

o_waveform	Returns a waveform (or a family of waveforms) if the input argument is a family of waveforms.
g_value	Returns the X value corresponding to the valley specified with *x_numberOfValleys* if the input argument is a single waveform.
l_value	Returns a list of X locations when *x_numberOfValleys* is `0` and the input argument is a single waveform.
`nil`	Returns `nil` and an error message otherwise.

Example

```
xmin( v( "/net9" ) 1 )
```

Gets the time value (X axis) at which the voltage of `"/net9"` has its first low point or valley.

```
xmin( v( "/net9" ) 0 )
```

Gets the list of time values (X axis) at which the voltage of `"/net9"` has low points or valleys.

xval

```
xval( o_waveform )
     => o_waveform/nil
```

Description

Returns a waveform whose X vector and Y vector are equal to the input waveform's X vector.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)

Value Returned

o_waveform	Returns a waveform if the input argument is a single waveform. Returns a family of waveforms if the input argument is a family of waveforms.
`nil`	Returns `nil` and an error message otherwise.

Example

```
xval( v( "/net8" ))
```

Returns a waveform in which the X vector for the voltage of `"/net8"` is also used for the Y vector.

ymax

```
ymax( o_waveform )
     => n_max/o_waveformMax/nil
```

Description

Computes the maximum value of the waveform's Y vector.

A waveform consists of an independent-variable X vector and a corresponding Y vector.

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)

Value Returned

n_max	Returns a number representing the maximum value of Y if the input argument is a single waveform.
o_waveformMax	Returns a waveform (or family of waveforms) representing the maximum value of Y if the input argument is a family of waveforms.
`nil`	Returns `nil` and an error message otherwise.

Example

```
ymax( v( "/net9" ) )
```

Gets the maximum voltage (Y value) of `"/net9"`.

ymin

```
ymin( o_waveform )
    => n_min/o_waveformMin/nil
```

Description

Computes the minimum value of a waveform's Y vector.

(A waveform consists of an independent-variable X vector and a corresponding Y vector.)

Arguments

o_waveform	Waveform object representing simulation results that can be displayed as a series of points on a grid. (A waveform object identifier looks like this: `srrWave:`*XXXXX*.)

Value Returned

n_min	Returns a number representing the minimum value of Y if the input argument is a single waveform.
o_waveformMin	Returns a waveform (or family of waveforms) representing the minimum value of Y if the input argument is a family of waveforms.
`nil`	Returns `nil` and an error message otherwise.

Example

```
ymin( v( "/net9" ) )
```

Gets the minimum voltage (Y value) of `"/net9"`.

Spectre RF Calculator Functions

This section describes the following calculator functions used for Spectre RF data analysis:

ifreq

```
ifreq( s_ana t_terminal [freq n_freq])
     => o_waveform/nil
```

Description

Returns the current of the terminal at a specified frequency or at all frequencies in the frequency domain.

Arguments

s_ana	Analysis type or analysis name. The available analyses are `hb`, `pss`, `qpss`, `pac`, `hbac`, `qpac`, and `ac`. Default value: `hb`
t_terminal	Terminal name on the schematic or signal name from the Results Browser.
n_freq	Frequency for which you want to plot the results. It is an optional field. Valid values: Any integer or floating point number. Default value: `nil` When you specify `nil`, current on all the frequency points are returned.

Value Returned

o_waveform	Returns a waveform representing current at a specified frequency or at all frequency points.
`nil`	Returns `nil` and an error message otherwise.

Example

```
ifreq("hb" "/load/PLUS" 50 )
```

Returns the current for `/load/PLUS` signal, which is obtained from `hb` analysis, at frequency=`50`.

ih

```
ih( s_ana t_terminal [harmonic x_hlist])
    => o_waveform/nil
```

Description

Returns the current of the terminal at a specified harmonic or at all harmonics in the frequency domain.

Arguments

s_ana	Analysis type or analysis name. The available analyses are `hb`, `pss`, `qpss`, `pac`, `hbac`, and `qpac`. Default value: `hb`
t_terminal	Terminal name on the schematic or signal name from the Results Browser.
x_hlist	Harmonics for which you want to plot the results. It is an optional field. For analyses, such as `hb`, `pss`, `pac`, and `hbac`, you can add either single harmonic value or an available list of harmonic values in this field. Valid values: Any integer or a list from the available list of harmonics. You can find the available harmonics by using the harmonicList function. Default value: `nil`.

Value Returned

o_waveform	Returns a waveform representing current at a specified harmonic or at all harmonic points.
`nil`	Returns `nil` and an error message otherwise.

Example

```
ih("hb" "/rf/PLUS" 2 )
```

Returns the current for `/rf/PLUS` signal, which is obtained from `hb` analysis, at harmonic= `2`.

itime

```
itime( s_ana t_terminal [time n_time])
     => o_waveform/nil
```

Description

Returns the current of the terminal at a specified time point or at all time points in the time domain.

Arguments

s_ana	Analysis type or analysis name. The available analyses are `hb`, `pss`, and `tran`. Default value: `hb`
t_terminal	Terminal name on the schematic or signal name from the Results Browser.
n_time	Time points for which you want to plot the results. If you specify a time point in this field, the result of the specified time is returned. It is an optional field. Valid values: Any integer or floating point number. Default value: `nil`.

Value Returned

o_waveform	Returns a waveform representing current at a specified time point or at all time points.
`nil`	Returns `nil` and an error message otherwise.

Example

```
itime("hb" "/load/PLUS" 4 )
```

Returns the current for `/load/PLUS` signal, which is obtained from `hb` analysis, at time=4s.

pir

```
pir( s_ana t_branch1 t_branch2 n_resistance [harmonic x_hlist])
     => o_waveform/nil
```

Description

Returns the spectral power from current and resistance for a specified harmonic list or for all harmonic points.

Arguments

s_ana	Analysis type or analysis name. The available analyses are `hb`, `pss`, `qpss`, `pac`, `hbac` and `qpac`. Default value: `hb`
t_branch1	First branch name on the schematic or signal name from the Results Browser.
t_branch2	Second branch name on the schematic or signal name from the Results Browser.
n_resistance	The resistance value. Valid values: Any integer or floating point number.
x_hlist	Harmonics for which you want to plot the results. It is an optional field. For analyses, such as `hb`, `pss`, `pac`, and `hbac`, you can add either single harmonic value or an available list of harmonic values in this field. Valid values: Any integer or a list from the available list of harmonics. You can find the available harmonics by using the harmonicList function. Default value: `nil`.

Value Returned

o_waveform	Returns a waveform representing spectral power from current and resistance for a specified harmonic list.
`nil`	Returns `nil` and an error message otherwise.

Example

```
pir("hb" "/V1/PLUS" "/rf/PLUS" 2 5 )
```

This example returns the spectral power for `/V1/PLUS` and `/rf/PLUS`, which are obtained from the `hb` analysis, at resistance=`2` ohms and harmonic=`5` .

pmNoise

```
pmNoise( s_ana [freq n_freq]s_modifier g_dsb )
     => o_waveform/n_pnoise/nil
```

Description

Returns the modulated phase noise at a specified frequency or for the entire spectrum.

Arguments

s_ana	Analysis type or analysis name. Valid values: `pnoise`, and `hbnoise`. Default value: `pnoise`
n_freq	Frequency for which you want to calculate the modulated phase noise. Valid values: Any integer or floatng point number Default value: `nil`, which means the frequency at all points are calculated.
s_modifier	Modifier to be used. Valid values: `dBc`, `normalized`, `Power`, `Magnitude`, and `dBV` Default value: `dBc`.
g_dsb	Specifies whether you want to include the double side band. Valid values: `t` and `nil` Default value: `t`

Value Returned

n_pnoise	Returns the modulated phase noise at the specified frequency point.
o_waveform	Returns a waveform representing the modulated phase noise at all frequency points.
`nil`	Returns `nil` and an error message otherwise.

Example

```
pmNoise("hbnoise" 50 "dBc" t )
```

This example returns the modulated phase noise for `hbnoise` analysis at frequency=`50` and modifier=`dBc` and double side bands included.

pn

```
pn( s_ana [freq n_freq])
     => o_waveform/n_pn/nil
```

Description

Returns the phase noise at a specified frequency or at all frequency points.

Arguments

s_ana	Analysis type or analysis name. Valid values: `pnoise`, `hbnoise`, and `qpnoise`. Default value: `pnoise`
n_freq	Frequency for which you want to calculate the phase noise. Valid values: Any integer or floating point number Default value: `nil`, which means the frequency at all points are calculated.

Value Returned

n_pn	Returns the phase noise at a specified frequency point.
o_waveform	Returns a waveform representing the phase noise at all frequency points.
`nil`	Returns `nil` and an error message otherwise.

Example

```
pn("hbnoise" 50 )
```

This example returns the phase noise for `hbnoise` analysis at frequency=`50`.

pvi

```
pvi( s_ana t_pos t_neg t_branch1 t_branch2 [harmonic x_hlist])
     => o_waveform/nil
```

Description

Returns the spectral power from voltage and current for a specified harmonic list or for all harmonics.

Arguments

s_ana	Analysis type or analysis name. The available analyses are `hb`, `pss`, `qpss`, `pac`, `hbac` and `qpac`. Default value: `hb`
t_pos	Positive node or net from the schematic or from the Results Browser. This field can also contain an explicit voltage value.
t_neg	Negative node or net from the schematic or from the Results Browser. This field can also contain an explicit voltage value.
t_branch1	First branch name on the schematic or signal name from the Results Browser.
t_branch2	Second branch name on the schematic or signal name from the Results Browser.
x_hlist	Harmonics for which you want to plot the results. It is an optional field. For analyses, such as `hb`, `pss`, `pac`, and `hbac`, you can add either single harmonic value or available list of harmonic values in this field. Valid values: Any integer or a list from the available list of harmonics. You can find the available harmonics by using the harmonicList function. Default value: `nil`

Value Returned

o_waveform	Returns a waveform representing the spectral power from voltage and current for a specified harmonic list or for all harmonics.

`nil`	Returns `nil` and an error message otherwise.

Example

```
pvi("hb" "/RFin" "/RFout" "/V1/PLUS" "/V2/PLUS" 2)
```

This example returns the spectral power for the following values:

- *Analysis Type* is `hb`
- *Positive node* is `/RFin`
- *Negative node* is `/RFout`
- *Branch name* 1 `/V1/PLUS`
- *Branch name 2* `/V2/PLUS`
- *Harmonic List* is `2`

pvr

```
pvr( s_ana t_pos t_neg n_resistance [harmonic x_hlist])
     => o_waveform/nil
```

Description

Returns the spectral power at a specified harmonic list or at all harmonics with resistor and voltage on the positive and negative nodes.

Arguments

s_ana	Analysis type or analysis name. The available analyses are `hb`, `pss`, `qpss`, `pac`, `hbac` and `qpac`. Default value: `hb`
t_pos	Positive node or net from the schematic or from the Results Browser. This field can also contain an explicit voltage value.
t_neg	Negative node or net from the schematic or from the Results Browser. This field can also contain an explicit voltage value.
n_resistance	The resistance value. Valid values: Any integer or floating point number
x_hlist	Specify the harmonics for which you want to plot the results. It is an optional field. For analyses, such as `hb`, `pss`, `pac`, and `hbac`, you can add either single harmonic value or available list of harmonic values in this field. Valid values: Any integer or a list from the available list of harmonics. You can find the available harmonics by using the harmonicList function. Default value: `ni`.

Value Returned

o_waveform	Returns a waveform representing the spectral power on specified harmonic list or on all harmonics with resistor and voltage on the positive and negative nodes
`nil`	Returns `nil` and an error message otherwise.

Example

```
pvr("hb" "/RFin" "/RFout" 2 2 )
```

This example returns the spectral power for the following values:

- *Analysis Type* is `hb`
- *Positive node* is `/RFin`
- *Negative node* is `/RFout`
- *Resistance* is `2`
- *Harmonic List* is `2`

spm

```
spm( s_ana x_index1 x_index2 [port1 x_port1] [port2 x_port2])
     => o_waveform/nil
```

Description

Returns the waveform for s-parameters.

Arguments

s_ana	Analysis type or analysis name. Valid values: `sp`, `psp`, `qpsp`, and `hbsp` Default value: `sp`
x_index1	Port index for sp simulation. By default, this field is set to blank. Valid values: Available port index, such as `1`, `2`.
x_index1	Port index for sp simulation. By default, this field is set to blank. Valid values: Available port index, such as `1`, `2`.
x_port1	Port instance. The port instance can be specified only for the differential s-parameter analysis and not applicable for `psp`, `qpsp` and `hbsp` analyses. Valid values: Predefined values “`c`” and “`d`” for Spectre simulator.
x_port2	Port instance. The port instance can be specified only for the differential s-parameter analysis and not applicable for `psp`, `qpsp` and `hbsp` analyses. Valid values: Predefined values “`c`” and “`d`” for Spectre simulator.

Value Returned

o_waveform	Returns a waveform representing the s-parameters.
`nil`	Returns `nil` and an error message otherwise.

Example

```
spm("sp" 1 1 ?port1 nil ?port2 nil)
```

This example plots the s-parameter waveform for `sp` analysis with index1=`1` and index 2=`1`.

totalNoise

```
totalNoise( s_ana n_sfreq n_efreq [instances l_instances] )
     => n_totalNoise/nil
```

Description

Returns the total noise in a specified frequency limit.

Arguments

s_ana	Analysis type or analysis name. The available analyses are `noise`, `pnoise`, `qpnoise`, and `hbnoise`. Default value: `noise`.
n_sfreq	The start frequency. Valid values: Any integer or floating point number
n_efreq	The end frequency. Valid values: Any integer or floating point number
l_instances	List of instances or instance names. The noise contributed by the instances specified in this field is ignored while calculating the total noise. This is an optional field.

Value Returned

n_totalNoise	Returns the total noise in a specified frequency limit.
`nil`	Returns `nil` and an error message otherwise.

Example

```
totalNoise("hbnoise" 1k 100k out )
```

This example returns the total noise for `hbnoise` analysis in the frequency range `1k` to `100k` with instance `out` being excluded.

vfreq

```
vfreq( s_ana t_net [freq x_freq] )
     => o_waveform/nil
```

Description

Returns the voltage of net at a specified frequency or at all frequencies in the frequency domain.

Arguments

s_ana	Analysis type or analysis name. The available analyses are `hb`, `pss`, `qpss`, `pac`, `hbac`, `qpac`, and `ac`. Default value: `hb`
t_net	Net name from the schematic or signal name from the Results Browser.
x_freq	Frequency for which you want to plot the results. It is an optional field. Valid values: Any integer value. Default Value: `nil`

Value Returned

o_waveform	Returns a waveform representing the voltage of net at a specified frequency.
`nil`	Returns `nil` and an error message otherwise

Example

```
vfreq("hb" "/outp" 50 )
```

This example returns the voltage of `/outp` net from `hb` analysis at frequency=`50`.

vh

```
vh( s_ana t_net [harmonic x_hlist] )
    => o_waveform/nil
```

Description

Returns the voltage on net at a specified harmonic or at all harmonics in the frequency domain.

Arguments

s_ana	Analysis type or analysis name. The available analyses are `hb`, `pss`, `qpss`, `pac`, `hbac`, and `qpac`. Default value: `hb`
t_net	Net name on the schematic or signal name from the Results Browser.
x_hlist	Harmonics for which you want to plot the results. It is an optional field. For analyses, such as `hb`, `pss`, `pac`, and `hbac`, you can add either single harmonic value or available list of harmonic values in this field. Valid values: Any integer or a list from the available list of harmonics. You can find the available harmonics by using the harmonicList function. Default value: `nil`.

Value Returned

o_waveform	Returns a waveform representing the voltage on net on the specified harmonic.
`nil`	Returns `nil` and an error message otherwise

Example

```
vh("hb" "/outp" 5 )
```

This example returns the voltage of `/outp` net from `hb` analysis at harmonic=`5`.

vtime

```
vtime( s_ana t_net [time n_time] )
     => o_waveform/nil
```

Description

Returns the voltage of net at a specified time point or at all time points in the time domain.

Arguments

s_ana	Analysis type or analysis name. The available analyses are `hb`, `pss`, and `tran` Default value: `hb`
t_net	Net name from the schematic or signal name from the Results Browser.
n_time	Time points for which you want to plot the results. If you specify a time point in this field, the result of the specified time is returned. Otherwise, It is an optional field. Valid values: Any integer or floating point number. Default value: `nil`.

Value Returned

o_waveform	Returns a waveform representing the voltage of net at a specified time point.
`nil`	Returns `nil` and an error message otherwise

Example

```
vtime("hb" "/outm" 20)
```

This example returns the voltage of `/outp` net from `hb` analysis at time=`20s`.

ypm

```
ypm( s_ana x_index1 x_index2 )
     => o_waveform/nil
```

Description

Returns the waveform for y-parameters.

Arguments

s_ana	Analysis type or analysis name. Valid values: `sp`, `psp`, `qpsp`, and `hbsp` Default value: `sp`
x_index1	Port index for sp simulation. By default, this field is set to blank. Valid values: Available port index, such as `1, 2`
x_index1	Port index for sp simulation. By default, this field is set to blank. Valid values: Available port index, such as `1, 2`

Value Returned

o_waveform	Returns a waveform representing the y-parameters.
`nil`	Returns `nil` and an error message otherwise.

Example

```
ypm("sp" 1 1)
```

This example returns the waveform for y-parameters when index1=`1` and index2=`1`.

zpm

```
zpm( s_ana x_index1 x_index2 )
     => o_waveform/nil
```

Description

Returns the waveform for z-parameters.

Arguments

s_ana	Analysis type or analysis name. Valid values: `sp`, `psp`, `qpsp`, and `hbsp` Default value: `sp`
x_index1	Port index for sp simulation. By default, this field is set to blank. Valid values: Available port index, such as `1`, `2`
x_index1	Port index for sp simulation. By default, this field is set to blank. Valid values: Available port index, such as `1`, `2`

Value Returned

o_waveform	Returns a waveform representing the z-parameters.
`nil`	Returns `nil` and an error message otherwise.

Example

```
zpm("sp" 1 1)
```

This example returns the waveform for z-parameters when index1=`1` and index2=`1`.

11

Parametric Analysis Commands

These commands set up a parametric analysis. When you run a parametric analysis, you can plot the resulting data as a family of curves.

This chapter contains information on the following commands:

paramAnalysis

```
paramAnalysis(
     t_desVar
     [?start n_start]
     [?stop n_stop]
     [?center n_center]
     [?span n_span]
     [?step f_step]
     [?lin n_lin]
     [?log n_log]
     [?dec n_dec]
     [?oct n_oct]
     [?times n_times]
     [?spanPercent n_spanPercent]
     [?sweepType t_sweepType]
     [?values l_values]
     [o_paramAnalysis])
     => undefined/nil
```

Description

Sets up a parametric analysis.

Groups the PSF data so that it can be plotted as a family of curves when the analysis is finished. The commands can be nested as shown in the syntax of the command.

If you specify more than one range, the OCEAN environment uses the following precedence to select a single range to use.

n_start, *n_stop*	highest precedence
n_center, *n_span*	
n_center, *n_spanPercent*	lowest precedence

Similarly, if you specify more than one step control, the OCEAN environment uses the following precedence.

f_step	highest precedence
n_lin	
n_dec	
n_log	
n_oct	

n_times	lowest precedence

To run the analysis, use the `paramRun` command described in "paramRun" on page 537.

Arguments

t_desVar	Name of the design variable to be swept.
n_start	Beginning value for the design variable.
n_stop	Final value for the design variable.
n_center	Center point for a range of values that you want to sweep.
n_span	Range of values that you want to sweep around the center point. For example, if *n_center* is 100 and *n_span* is 20 then the sweep range extends from 90 to 110.
f_step	Increment by which the value of the design variable changes. For example, if *n_start* is `1.0`, *n_stop* is `2.1`, and *f_step* is `0.2`, the parametric analyzer simulates at values 1.0, 1.2, 1.4, 1.6, 1.8, and 2.0.
n_lin	The number of steps in the analysis. The parametric analyzer automatically assigns equal intervals between the steps. With this option, there is always a simulation at both *n_start* and *n_stop*. The value for the *n_lin* argument must be an integer greater than 0. For example, if *n_start* is `0.5`, *n_stop* is `2.0`, and *n_lin* is `4`, the parametric analyzer simulates at values 0.5, 1.0, 1.5, and 2.0.
n_log	The number of steps between the starting and stopping points at equal-ratio intervals using the following formula: $log\ multiplier = (n\_stop/n\_start)^{(n\_log\ -1)}$ The number of steps can be any positive number, such as 0.5, 2, or 6.25. Default value: `5`

For example, if *n_start* is 3, *n_stop* is 15, and *n_log* is 5, the parametric analyzer simulates at values 3, 4.48605, 6.7082, 10.0311, and 15.

The ratios of consecutive values are equal, as shown below.

3/4.48605 = 4.48605/6.7082 = 6.7082/10.0311 = 10.0311/15 = .67

n_dec The number of steps between the starting and stopping points calculated using the following formula:

$$decade\ multiplier = 10^{1/n\_dec}$$

The number of steps can be any positive number, such as 0.5, 2, or 6.25.
Default value: 5

For example, if *n_start* is 1, *n_stop* is 10, and *n_dec* is 5, the parametric analyzer simulates at values 1, 1.58489, 2.51189, 3.98107, 6.30957, and 10.

The values are 10^0, $10^{.2}$, $10^{.4}$, $10^{.6}$, $10^{.8}$, and 10^1.

n_oct The number of steps between the starting and stopping points using the following formula:
The number of steps can be any positive number, such as 0.5, 2, or 6.25.
Default value: 5

For example, if *n_start* is 2, *n_stop* is 4, and *n_oct* is 5, the parametric analyzer simulates at values 2, 2.2974, 2.63902, 3.03143, 3.4822, and 4.

These values are 2^1, $2^{1.2}$, $2^{1.4}$, $2^{1.6}$, $2^{1.8}$, and 2^2.

$$octave?multiplier = 2^{1/(n\_oct)}$$

n_times A multiplier. The parametric analyzer simulates at the points between *n_start* and *n_stop* that are consecutive multiples of *n_times*.

For example, if *n_start* is 1, *n_stop* is 1000, and *n_times* is 2, the parametric analyzer simulates at values 1, 2, 4, 8, 16, 32, 64, 128, 256, and 512.

n_spanPercent

Range specified as a percentage of the center value. For example, if *n_center* is 100 and *n_spanPercent* is 40, the sweep range extends from 80 to 120.

t_sweepType

Type of parametric analysis.
Valid values are:
- paramset - Runs Parametric Set analysis, specific to Spectre.
- nil - Runs Sweeps & Ranges type parametric analysis.
Default value: nil

l_values

List of values to be swept. You can use *l_values* by itself or in conjunction with *n_start*, *n_stop*, and *f_step* to specify the set of values to sweep.

o_paramAnalysis

Value returned from another paramAnalysis call used to achieve multidimensional parametric analysis.

Value Returned

undefined

The return value for this command is undefined.

nil

Returns nil and prints an error message if there are problems setting the option.

Example

```
paramAnalysis( "rs" ?start 200 ?stop 1000 ?step 200
    ?values '(1030 1050 1090) )
```

Sets up a parametric analysis for the rs design variable. The swept values are 200, 400, 600, 800, 1000, 1030, 1050, and 1090.

```
paramAnalysis( "rl" ?start 200 ?stop 600 ?step 200
    paramAnalysis( "rs" ?start 300 ?stop 700 ?step 200
    )
)
```

Sets up a nested parametric analysis for the rl design variable.

```
paramAnalysis("temp" ?start -50 ?stop 100 ?step 50)
```

Sets up a parametric analysis for temperature.

paramRun

```
paramRun( [s_paramAnalysis] )
     => t / nil

paramRun( [?jobName t_jobName] [?drmsCmd t_drmsCmd] )
     => s_jobName/nil

paramRun( [?jobName t_jobName] [?host t_hostName] [?queue t_queueName]
     [?startTime t_startTime] [?termTime t_termTime] [?dependentOn
     t_dependentOn] [?mail t_mailingList] [?block s_block] [?notify
     s_notifyFlag] [?lsfResourceStr s_lsfResourceStr] )
     => s_jobName/nil
```

Description

Runs the specified parametric analysis.

If you do not specify a parametric analysis, all specified analyses are run. Distributed processing must be enabled using the `hostmode` command before parametric analyses can be run in distributed mode.

When the `paramRun` command finishes, the PSF directory contains a file named `runObjFile` that points to a family of data. To plot the family, use a normal `plot` command. For example, you might use `plot(v("/out"))`.

For information about specifying a parametric analysis, see the `paramAnalysis` command described in "paramAnalysis" on page 532.

Arguments

t_jobName	Used as the basis of the job name. The value entered for *t_jobName* is used as the job name and return value if the run command is successful. If the name given is not unique, a number is appended to create a unique job name.
t_hostName	Name of the host on which to run the analysis. If no host is specified, the system assigns the analysis to an available host.
t_queueName	Name of the queue. If no queue is defined, the analysis is placed in the default queue (your home machine).
t_startTime	Desired start time for the job. If dependencies are specified, the job does not start until all dependencies are satisfied.

t_termTime	Termination time for job. If the job is not completed by *t_termTime*, the job is terminated.
t_dependentOn	List of jobs on which the specified analysis is dependent. The analysis is not started until after dependent jobs are complete.
t_mailingList	List of users to be notified by e-mail when the analysis is complete.
s_block	When *s_block* is not `nil`, the OCEAN script halts until the job is complete. Default value: `nil`
s_notifyFlag	When *notifyFlag* is not `nil`, a job completion message is echoed to the OCEAN interactive window. Default value: `t`
s_paramAnalysis	Parametric analysis.
t_drmsCmd	A DRMS (Distributed Resource Management System) command, such as a bsub command for LSF or a qsub command for SGE (Sun Grid Engine) used to submit a job. When this argument is used, all other arguments, except `?jobName` will be ignored. Moreover, it will not be possible to call the OCEAN function `wait` on the jobs submitted using this argument. To know more about the command option, refer to the section Submitting a Job in the chapter *Using the Distributed Processing Option in the Analog Design Environment* of the *Virtuoso Analog Distributed Processing OptionUser Guide*.
s_lsfResourceStr	Specifies an LSF Resource Requirement string to submit a job. It is effective only in the LSF mode.

Value Returned

`t`	Returned if successful.
`nil`	Returns `nil` and prints an error message if unsuccessful.

Example

```
paramRun() => t
```

Runs all specified parametric analyses.

```
rsAnalysis = paramAnalysis("CAP" ?values '(10 20))
paramRun('rsAnalysis)
```

OR

```
rsAnalysis = paramAnalysis("CAP" ?values '(10 20) paramAnalysis("RES" ?values '(10
20 )))
paramRun('rsAnalysis)
```

Runs the `rs` parametric analysis.

```
paramRun(?queue "background" ?lsfResourceStr "mem>500")
```

Runs the analysis in the queue named `background` on a machine, if it has at least `500` MB of RAM memory.

12

OCEAN Distributed Processing Commands

The Open Command Environment for Analysis (OCEAN) distributed processing commands let you run OCEAN jobs across a collection of computer systems.

This chapter contains information on the following commands:

This chapter also provides sample OCEAN scripts that optimally use these commands. See the section Sample Scripts on page 553.

For detailed information on distributed processing, refer to Virtuoso Analog Distributed Processing Option User Guide.

deleteJob

```
deleteJob( t_jobName [t_jobName2 t_jobName3 … t_jobNameN] )
     => t / nil
```

Description

Removes a job or series of jobs from the text-based job monitor.

Deleted jobs are no longer listed in the job monitor. The `deleteJob` command applies only to ended jobs.

Arguments

t_jobName Name used to identify the job.

t_jobname2…t_jobnameN
Additional jobs that you want to delete.

Value Returned

`t` Returns `t` if successful.

`nil` Returns `nil` and prints an error message if unsuccessful.

Example

```
deleteJob( 'myckt)
=> t
```

Deletes the `myckt` job.

digitalHostMode

```
digitalHostMode( {'local | 'remote} )
     => t / nil
```

Description

For mixed-signal simulation, specifies whether the digital simulator will run locally or on a remote host.

Arguments

`'local`	Sets the simulation to run locally on the user's machine.
`'remote`	Sets the simulation to run on a remote host. If you use this argument, you must specify the host name by using the `digitalHostName` command.

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` and prints an error message if unsuccessful.

Example

```
digitalHostMode( 'local )
```

Sets the digital simulator to run locally on the user's host.

digitalHostName

```
digitalHostName( t_name )
     => t / nil
```

Description

For mixed-signal simulation, specifies the name of the remote host for the digital simulator.

When you use the `digitalHostMode('remote)` command, use this command to specify the name of the remote host.

Arguments

t_name	Name used to identify the host for the digital simulator.

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` and prints an error message if unsuccessful.

Example

```
digitalHostName( "digitalhost" )
```

Indicates that the digital simulator runs on the host called `digitalhost`.

hostMode

```
hostMode( { 'local | 'remote | 'distributed } )
     => t / nil
```

Description

Sets the simulation host mode.

The default value for `hostMode` is specified in the `asimenv.startup` file with the `hostMode` environment variable.

Arguments

`'local`	Sets the simulation to run locally on the user's machine.
`'remote`	Sets the simulation to run on a remote host queue. For this release, the remote host is specified in the `.cdsenv` file.
`'distributed`	Sets the simulation to run using the distributed processing software.

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` and prints an error message if unsuccessful.

Example

```
hostMode( 'distributed )
=> t
```

Enables distributed processing on the current host.

hostName

```
hostName( t_name )
     => t / nil
```

Description

Specifies the name of the remote host.

When you use the `hostMode('remote)` command, use this command to specify the name of the remote host.

Arguments

t_name	Name used to identify the remote host.

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` and prints an error message if unsuccessful.

Example

```
hostName( "remotehost" )
```

Specifies that the host called `remotehost` is to be used for remote simulation.

killJob

```
killJob( t_jobName [t_jobName2 t_jobName3 … t_jobNameN] )
     => t / nil
```

Description

Stops processing of a job or a series of jobs.

The job might still show up in the job monitor, but it cannot be restarted. Use the `deleteJob` command to remove the job name from the job server and job monitor.

Arguments

t_jobName Name used to identify the job.

t_jobname2…t_jobnameN
Additional jobs that you want to stop.

Value Returned

`t` Returns `t` if successful.

`nil` Returns `nil` and prints an error message if unsuccessful.

Example

```
killJob( 'myckt )
=> t
```

Aborts the job called `myckt`. If the job is in the queue and has not started running yet, it is deleted from the queue.

monitor

```
monitor( [?taskMode s_taskMode] )
     => t / nil
```

Description

Monitors the jobs submitted to the distributed system.

Arguments

s_taskMode	When not `nil`, multitask jobs are expanded to show individual jobs. A multitask job is one that contains several related jobs.

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` and prints an error message if unsuccessful.

Example

```
monitor( ?taskMode t )
```

Displays the name, host, and queue for all pending tasks sorted on a queue name.

remoteDir

```
remoteDir( t_path )
     => t / nil
```

Description

Specifies the project directory on the remote host to be used for remote simulation.

When you use the `hostMode('remote)` command, use this command to specify the project directory on the remote host.

Arguments

t_path	Specifies the path to the project directory on the remote host to be used for remote simulation.

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` and prints an error message if unsuccessful.

Example

```
remoteDir( "~/simulation" )
```

Specifies that the project directory is `~/simulation`.

resumeJob

```
resumeJob( t_jobName [t_jobName2 t_jobName3 … t_jobNameN] )
     => t / nil
```

Description

Resumes the processing of a previously suspended job or series of jobs. The `resumeJob` command applies only to jobs that are suspended.

Arguments

t_jobName	Name used to identify the job.
t_jobName2…t_jobNameN	Additional jobs that you want to resume

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` and prints an error message if unsuccessful.

Example

```
resumeJob( 'myckt )
=> t
```

Resumes the `myckt` job that was halted with the `suspendJob` command.

suspendJob

```
suspendJob( t_jobName [t_jobName2 t_jobName3 … t_jobNameN] )
     => t / nil
```

Description

Suspends the processing of a job or series of jobs. The `suspendJob` command applies only to jobs that are pending or running.

Arguments

t_jobName	Name used to identify the job.
t_jobName2…t_jobnameN	Additional jobs that you want to suspend.

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` and prints an error message if unsuccessful.

Example

```
suspendJob( 'myckt )
=> t
```

Suspends the job called `myckt`.

wait

```
wait( [?queue t_queueName] jobName [jobName2 jobName3 … jobNameN] )
     => t / nil
```

Description

Postpones processing of a script until the specified jobs complete. This command is ignored if distributed processing is not available.

The `wait` command is very useful when you use the non-blocking mode of distributed processing and you want to do some post-processing, such as selecting and viewing results after a job is completed. The `wait` command is not required when you use the blocking mode of distributed processing. To know more about blocking and non-blocking modes of DP, refer to Virtuoso Analog Distributed Processing Option User Guide.

Arguments

t_queueName	The name of queue on which job launched by `wait` is submitted.
t_jobName	Name used to identify the job. The job name is user defined or system generated, depending on how the user submitted the job.
t_jobName2…t_jobnameN	Additional jobs that you want to postpone.

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` and prints an error message if unsuccessful.

Examples

```
wait( 'myckt1 )
=> t
```

Postpones execution of all subsequent OCEAN commands until the job `myckt1` completes.

```
wait( ?queue "lnx64" 'job0 )
=> t
```

Job launched by `wait` is submitted on `lnx64` queue that postpones the execution of all subsequent OCEAN commands until the job `job0` completes.

Sample Scripts

This section provides sample scripts for the following:

- To submit multiple jobs and show the use of the dependentOn argument in one job
- To set up and run a simple analysis in blocking mode and select results
- To set up and run a parametric analysis in blocking mode and select results
- To submit multiple jobs without using wait or selecting results
- To submit multiple jobs using wait and selection of results

To submit multiple jobs and show the use of the dependentOn argument in one job

This script can be used to submit multiple jobs while using the `dependentOn` argument in one of these jobs.

```
; set up the environment for the jobs
simulator( ’spectre )
hostMode( ’distributed )
design(  "/home/simulation/test2/spectre/schematic/netlist/netlist")
resultsDir( "/home/simulation/test2/spectre/schematic" )
analysis(’tran ?stop "5u"  )
temp( 27 )

jobList = nil

; starting first job
jobList = append1( jobList run( ?queue "test" ?host "menaka" ) )

analysis(’tran ?stop "50u")

; starting second job
jobList = append1( jobList run(?jobName "job_2" ?queue "test" ?host "menaka"))

analysis(’tran ?stop "10u")

; starting third job, which is dependent on job_2
jobList= append1(jobList run(?jobName "disable" ?queue "test" ?dependentOn
                symbolToString(car(last(jobList)))))

; wait for all the jobs to complete
```

```
wait((append1 last(jobList) nil))

; open and plot the results of the jobs
openResults( car(last(jobList)))
selectResult( 'tran )
newWindow()
plot(getData("/net61") )

openResults( nth(1 jobList))
selectResult('tran)
newWindow()
plot(getData("/net61") )
```

To set up and run a simple analysis in blocking mode and select results

```
; set up the environment for Simple Analysis
simulator( 'spectre )
hostMode( 'distributed )
design(
"/home/amit/Artist446/simulation/ampTest/spectre/schematic/netlist/netlist" )
resultsDir( "/home/Artist446/simulation/ampTest/spectre/schematic" )
modelFile(
    '("/home/Artist446/Models/myModels.scs" "")
)
analysis( 'tran ?stop "3u" )
desVar( "CAP" 0.8p )
temp( 27 )

; submit the job in blocking mode, to the queue test and machine menaka
run(?queue "test" ?host "menaka" ?block t)

; select and plot the results
selectResult( 'tran )
plot(getData("/out"))
```

To set up and run a parametric analysis in blocking mode and select results

```
; set up the environment for parametric analysis.
simulator( 'spectre )
hostMode( 'distributed )
design(
```

```
"/home/amit/Artist446/simulation/ampTest/spectre/schematic/netlist/netlist")
resultsDir( "/home/amit/Artist446/simulation/ampTest/spectre/schematic"
)
modelFile(
    '("/home/amit/Artist446/Models/myModels.scs" "")
)
analysis('tran ?stop "3u"  )
desVar(   "CAP" 0.8p    )
temp( 27 )
paramAnalysis("CAP" ?values '(1e-13 2.5e-13 4e-13 ))

; submit the job in blocking mode, to the queue test and machine menaka
paramRun(?queue "fast" ?host "menaka" ?block t)

; select and plot the results
selectResult( 'tran )
plot(getData("/out") )
```

To submit multiple jobs without using wait or selecting results

```
; set up the environment for the jobs
simulator( 'spectre )
hostMode( 'distributed )
design(
"/home/Artist446/simulation/ampTest/spectre/schematic/netlist/netlist")
resultsDir( "/home/Artist446/simulation/ampTest/spectre/schematic" )
modelFile(
    '("/home/Artist446/Models/myModels.scs" "")
)

; setup and submit first job
analysis('tran ?stop "3u"  )
desVar(   "CAP" 0.8p    )
temp( 27 )
run(?queue "SUN5_5032" ?host "menaka")

; setup and submit second job
analysis('ac ?start "1M"  ?stop "2M"  )
analysis('tran ?stop "3u"  )
desVar(   "CAP" 0.8p    )
temp( 27 )
```

```
run(?queue "SUN5_5032" ?host "menaka")
```

To submit multiple jobs using wait and selection of results

```
; set up the environment for the jobs
simulator( 'spectre )
hostMode( 'distributed )
design(
"/home/Artist446/simulation/ampTest/spectre/schematic/netlist/netlist")
resultsDir( "/home/Artist446/simulation/ampTest/spectre/schematic" )
modelFile(
    '("/home/Artist446/Models/myModels.scs" "")
)

; initialize jobList to nil
jobList = nil

; setup and submit first job
analysis('tran ?stop "3u"  )
desVar(   "CAP" 0.8p    )
temp( 27 )
jobList = append1( jobList run(?queue "SUN5_5032" ?host "menaka") )

; setup and submit second job
analysis('ac ?start "1M"  ?stop "2M"  )
analysis('tran ?stop "3u"  )
desVar(   "CAP" 0.8p    )
temp( 27 )
jobList = append1( jobList run(?queue "SUN5_5032" ?host "menaka"))

; wait for both the jobs to finish
wait( (append1 jobList nil) )

; open and plot the result of first job
openResults( (car jobList))
selectResult( 'tran )
plot(getData("/out") )

; open and plot the result of second job
openResults( (cadr jobList))
selectResult( 'tran )
```

```
plot(getData("/out") )
selectResult( 'ac )
plot(getData("/out") )

; delete the jobs
foreach( x jobList deleteJob( x ) )
```

13

Language Constructs

There are three types of SKILL language constructs:

- Conditional statements

  Conditional statements test for a condition and perform operations when that condition is found. These statements are `if`, `unless`, and `when`.

- Selection statements

  A selection statement allows a list of elements, each with a corresponding operation. A variable can then be compared to the list of elements. If the variable matches one of the elements, the corresponding operation is performed. These statements include `for`, `foreach`, and `while`.

- Iterative statements

  Iterative statements repeat an operation as long as a certain condition is met. These statements include `case` and `cond`.

  This chapter contains information on the following statements

if

```
if( g_condition g_thenExpression [g_elseExpression] )
     => g_result
```

Description

Evaluates *g_condition*, typically a relational expression, and runs *g_thenExpression* if the condition is true (that is, its value is non-nil); otherwise, runs *g_elseExpression*.

The value returned by `if` is the value of the corresponding expression evaluated.

Arguments

g_condition	Any Virtuoso® SKILL language expression.
g_thenExpression	Any SKILL expression.
g_elseExpression	Any SKILL expression.

Value Returned

g_result	Returns the value of *g_thenExpression* if *g_condition* has a non-nil value. The value of *g_elseExpression* is returned otherwise.

Example

```
x = 2
if( x > 5 1 0 )
=> 0
```

Returns `0` because `x` is less than `5`.

```
a ="npn"
if(( a == "npn" ) print( a ) ) "npn"
=> nil
```

Prints the string `npn` and returns the result of `print`.

```
x = 5
if( x "non-nil" "nil" )
=> "non-nil"
```

Returns `"non-nil"` because `x` was not `nil`. If `x` was `nil`, `"nil"` would be returned.

```
x = 7
if( x > 5 1 0 )
=> 1
```

Returns `1` because `x` is greater than `5`.

unless

```
unless( g_condition g_expr1 … )
     => g_result/nil
```

Description

Evaluates a condition. If the result is true (non-nil), it returns `nil`; otherwise it evaluates the body expressions in sequence and returns the value of the last expression.

The semantics of this function can be read literally as "unless the condition is true, evaluate the body expressions in sequence."

Arguments

g_condition	Any SKILL expression.
g_expr1…	Any SKILL expression.

Value Returned

g_result	Returns the value of the last expression of the sequence *g_expr1* … if *g_condition* evaluates to `nil`.
`nil`	Returns `nil` if *g_condition* evaluates to non-nil.

Example

```
x = -123
unless( x >= 0 println( "x is negative" ) -x )
=> 123
```

Prints `"x is negative"` as a side effect.

```
unless( x < 0 println( "x is positive ") x)
=> nil
```

Returns `nil`.

when

```
when( g_condition g_expr1 … )
     => g_result/nil
```

Description

Evaluates a condition.

If the result is non-nil, evaluates the sequence of expressions and returns the value of the last expression. Otherwise, returns `nil`.

Arguments

g_condition	Any SKILL expression.
g_expr1...	Any SKILL expression.

Value Returned

g_result	Returns the value of the last expression of the sequence *g_expr1* ... if *g_condition* evaluates to non-nil.
`nil`	returns `nil` if the *g_condition* expression evaluates to `nil`.

Example

```
x = -123
when( x < 0 println( "x is negative" ) -x )
=> 123
```

Prints `"x is negative"` as a side effect.

```
when( x >= 0 println( "x is positive" ) x)
=> nil
```

Returns `nil`.

for

```
for( s_loopVar x_initialValue x_finalValue g_expr1 [g_expr2 …] )
     => t
```

Description

Evaluates the sequence *g_expr1 g_expr2* … for each loop variable value, beginning with *x_initialValue* and ending with *x_finalValue*.

First evaluates the initial and final values, which set the initial value and final limit for the local loop variable named *s_loopVar*. Both *x_initialValue* and *x_finalValue* must be integer expressions. During each iteration, the sequence of expressions *g_expr1 g_expr2* … is evaluated and the loop variable is then incremented by one. If the loop variable is still less than or equal to the final limit, another iteration is performed. The loop ends when the loop variable reaches a value greater than the limit. The loop variable must not be changed inside the loop. It is local to the `for` loop and would not retain any meaningful value upon exit from the `for` loop.

Note: Everything that can be done with a `for` loop can also be done with a `while` loop.

Arguments

s_loopVar	Name of the local loop variable that must not be changed inside the loop.
x_initialValue	Integer expression setting the initial value for the local loop variable.
x_finalValue	Integer expression giving final limit value for the loop.
g_expr1	Expression to evaluate inside loop.
g_expr2 …	Additional expressions to evaluate inside loop.

Value Returned

`t`	This construct always returns `t`.

Example

```
sum = 0

for( i 1 10
    sum = sum + i
    printf( "%d" sum ))
=> t
```

Prints 10 numbers and returns `t`.

```
sum = 0

for( i 1 5
    sum = sum + i
    println( sum )
    )
=> t
```

Prints the value of `sum` with a carriage return for each pass through the loop:

```
1
3
6
10
15
```

foreach

```
foreach( s_formalVar g_exprList g_expr1 [g_expr2 …] )
     => l_valueList

foreach( (s_formalVar1…s_formalVarN) g_exprList1… g_exprListN g_expr1
     [g_expr2 …] )
     => l_valueList

foreach( s_formalVar g_exprTable g_expr1 [g_expr2 …])
     => o_valueTable
```

Description

Evaluates one or more expressions for each element of a list of values.

The first syntax form,

`foreach(` *s_formalVar g_exprList g_expr1* `[`*g_expr2* `…] )`
`=>` *l_valueList*

evaluates *g_exprList*, which returns a list *l_valueList*. It then assigns the first element from *l_valueList* to the formal variable *s_formalVar* and processes the expressions *g_expr1 g_expr2* … in sequence. The function then assigns the second element from *l_valueList* and repeats the process until *l_valueList* is exhausted.

The second syntax form,

`foreach( (`*s_formalVar1...s_formalVarN*`)` *g_exprList1… g_exprListN g_expr1* `[`*g_expr2* `…] )=>`
l_valueList

can iterate over multiple lists to perform vector operations. Instead of a single formal variable, the first argument is a list of formal variables followed by a corresponding number of expressions for value lists and the expressions to be evaluated.

The third syntax form,

`foreach(` *s_formalVar g_exprTable g_expr1* `[`*g_expr2* `…])`
`=>` *o_valueTable*

can be used to process the elements of an association table. In this case, *s_formalVar* is assigned each key of the association table one by one, and the body expressions are evaluated each iteration. The syntax for association table processing is provided in this syntax statement.

Arguments

s_formalVar	Name of the variable.

g_exprList	Expression whose value is a list of elements to assign to the formal variable *s_formalVar*.
g_expr1 g_expr2 …	Expressions to execute.
g_exprTable	Association table whose elements are to be processed.

Value Returned

l_valueList	Returns the value of the second argument, *g_exprList*.
o_valueTable	Returns the value of *g_exprTable*.

Example

```
foreach( x '( 1 2 3 4 ) println( x ) )
1
2
3
4
=> ( 1 2 3 4 )
```

Prints the numbers 1 through 4 and returns the second argument to `foreach`.

```
foreach( key myTable printf( "%L : %L" key myTable[key] ) )
```

Accesses an association table and prints each key and its associated data.

```
( foreach ( x y ) '( 1 2 3 ) '( 4 5 6 ) ( println x+y ) )
5
7
9
=> ( 1 2 3 )
```

Uses `foreach` with more than one loop variable.

Errors and Warnings

The error messages from `foreach` might at times appear cryptic because some `foreach` forms get expanded to call the mapping functions `mapc`, `mapcar`, `mapcan`, and so forth.

while

```
while( g_condition g_expr1 … )
=> t
```

Description

Repeatedly evaluates *g_condition* and the sequence of expressions *g_expr1* ... if the condition is true.

This process is repeated until *g_condition* evaluates to false (`nil`). Note that because this form always returns `t`, it is principally used for its side effects.

Note: Everything that can be done with a `for` loop can also be done with a `while` loop.

Arguments

g_condition	Any SKILL expression.
g_expr1	Any SKILL expression.

Value Returned

`t`	Always returns `t`.

Example

```
i = 0
while( (i <= 10) printf("%d" i++) )
=> t
```

Prints the digits 0 through 10.

case

```
case( g_selectionExpr l_clause1 [l_clause2 …] )
    => g_result/nil
```

Description

Evaluates the selection expression, matches the resulting selector values sequentially against comparators defined in clauses, and runs the expressions in the matching clause.

Each `l_clause` is a list of the form (`g_comparator` `g_expr1` [`g_expr2`…]), where a comparator is either an atom (that is, a scalar) of any data type or a list of atoms. Comparators are always treated as constants and are never evaluated. The `g_selectionExpr` expression is evaluated and the resulting selector value is matched sequentially against comparators defined in `l_clause1` `l_clause2`…. A match occurs when either the selector is equal to the comparator or the selector is equal to one of the elements in the list given as the comparator. If a match is found, the expressions in that clause and that clause only (that is, the first match) are run. The value of `case` is then the value of the last expression evaluated (that is, the last expression in the clause selected). If there is no match, `case` returns `nil`.

The symbol `t` has special meaning as a comparator: it matches anything. It is typically used in the last clause to serve as a default case when no match is found with other clauses.

Arguments

g_selectionExpr

An expression whose value is evaluated and tested for equality against the comparators in each clause. When a match is found, the rest of the clause is evaluated.

l_clause1 An expression whose first element is an atom or list of atoms to be compared against the value of *g_selectionExpr*. The remainder of the *l_clause* is evaluated if a match is found.

l_clause2… Zero or more clauses of the same form as *l_clause1*.

Value Returned

g_result Returns the value of the last expression evaluated in the matched clause.

`nil` Returns `nil` if there is no match.

Example

```
cornersType = "min"

type = case( cornersType
             ("min" path("./min"))
             ("typ" path("./typ"))
             ("max" path("./max"))
             (t println("you have not chosen an appropriate
                 corner")))
             => path is set to "./min"
```

Sets `path` to `./min`.

cond

```
cond( l_clause1 … )
     => g_result/nil
```

Description

Examines conditional clauses from left to right until either a clause is satisfied or there are no more clauses remaining.

This command is useful when there is more than one test condition, but only the statements of one test are to be carried out. Each clause is of the form `( g_condition g_expr1… )`. The `cond` function examines a clause by evaluating the condition associated with the clause. The clause is satisfied if *g_condition* evaluates to non-nil, in which case expressions in the rest of the clause are evaluated from left to right, and the value returned by the last expression in the clause is returned as the value of the `cond` form. If *g_condition* evaluates to `nil`, however, `cond` skips the rest of the clause and moves on to the next clause.

Arguments

l_clause1	Each clause must be of the form `(g_condition g_expr1…)`. When *g_condition* evaluates to non-nil, all the succeeding expressions are evaluated.

Value Returned

g_result	Returns the value of the last expression of the satisfied clause.
`nil`	Returns `nil` if no clause is satisfied.

Example

```
procedure( test(x)
    cond(((null x) (println "Arg is null"))
        ((numberp x) (println "Arg is a number"))
        ((stringp x) (println "Arg is a string"))
        (t (println "Arg is an unknown type")))
)
test( nil )
=> nil; Prints "Arg is null".
test( 5 )
=> nil; Prints "Arg is a number".
test( 'sym )
=> nil; Prints "Arg is an unknown type".
```

Tests each of the arguments according to the conditions specified with `cond`.

14

File Commands and Functions

This chapter contains information on the following commands:

close

```
close( p_port )
     => t
```

Description

Drains, closes, and frees a port.

When a file is closed, it frees the `FILE*` associated with *`p_port`*. Do not use this function on `piport`, `stdin`, `poport`, `stdout`, or `stderr`.

Arguments

`p_port`	Name of port to close.

Value Returned

`t`	The port closed successfully.

Example

```
p = outfile( "~/test/myFile" ) => port:"~/test/myFile"
close( p )
=> t
```

Drains, closes, and frees the `/test/myFile` port.

fscanf

```
fscanf( p_inputPort t_formatString [s_var1 ...]  )
     => x_items/nil
```

Description

Reads input from a port according to format specifications and returns the number of items read in.

The results are stored into corresponding variables in the call. The `fscanf` function can be considered the inverse function of the `fprintf` output function. The `fscanf` function returns the number of input items it successfully matched with its format string. It returns `nil` if it encounters an end of file.

The maximum size of any input string being read as a string variable for `fscanf` is currently limited to 8 K. Also, the function `lineread` is a faster alternative to `fscanf` for reading Virtuoso® SKILL objects.

The common input formats accepted by `fscanf` are summarized below.

Common Input Format Specifications

Format Specification	Types of Argument	Scans for
%d	fixnum	An integer
%f	flonum	A floating-point number
%s	string	A string (delimited by spaces) in the input

Arguments

p_inputPort Input port to read from.

t_formatString Format string to match against in the reading.

s_var1… Name of the variable in which to store results.

Value Returned

x_items	Returns the number of input items it successfully read in. As a side effect, the items read in are assigned to the corresponding variables specified in the call.
`nil`	Returns `nil` if an end of file is encountered

Example

```
fscanf( p "%d %f" i d )
```

Scans for an integer and a floating-point number from the input port `p` and stores the values read in the variables `i` and `d`, respectively.

Assume a file `testcase` with one line:

```
hello 2 3 world
```

```
x = infile("testcase")
=> port:"testcase"
```

```
fscanf( x "%s %d %d %s" a b c d )
=> 4
```

```
(list a b c d) => ("hello" 2 3 "world")
```

gets

`gets( s_variableName [p_inputPort] ) => t_string/nil`

Description

Reads a line from the input port and stores the line as a string in the variable. This is a macro.

The string is also returned as the value of `gets`. The terminating newline character of the line becomes the last character in the string.

Arguments

s_variableName	Variable in which to store the input string.
p_inputPort	Name of input port. Default value: `piport`

Value Returned

t_string	Returns the input string when successful.
`nil`	Returns `nil` when the end of file is reached. (*s_variableName* maintains its last value.)

Example

Assume the `test1.data` file has the following first two lines:

```
#This is the data for test1
0001 1100 1011 0111
p = infile("test1.data") => port:"test1.data"
gets(s p) => "#This is the data for test1"
gets(s p) => "0001 1100 1011 0111"
s => "0001 1100 1011 0111"
```

Gets a line from the `test1.data` file and stores it in the variable `s`. The `s` variable contains the last string stored in it by the `gets` function.

infile

```
infile( S_fileName )
     => p_inport/nil
```

Description

Opens an input port ready to read a file.

Always remember to close the port when you are done. The file name can be specified with either an absolute path or a relative path. In the latter case, the current SKILL path is used if it is not `nil`.

Arguments

S_fileName	Name of the file to be read; it can be either a string or a symbol.

Value Returned

p_inport	Returns the port opened for reading the named file.
`nil`	Returns `nil` if the file does not exist or cannot be opened for reading.

Example

```
in = infile( "~/test/input.il" ) => port:"~/test/input.il"
close( in )
=> t
```

Closes the `/test/input.il` port.

Opens the input port `/test/input.il`.

```
infile("myFile") => nil
```

Returns `nil` if `myFile` does not exist according to the current setting of the SKILL path or exists but is not readable.

load

```
load( t_fileName [t_password])
     => t
```

Description

Opens a file and repeatedly calls `lineread` to read in the file, immediately evaluating each form after it is read in.

This function uses the file extension to determine the language mode (`.il` for SKILL, `.ils` for SKILL++, and `.ocn` for a file containing OCEAN commands) for processing the language expressions contained in the file. For a SKILL++ file, the loaded code is always evaluated in the top-level environment.

`load` closes the file when the end of file is reached. Unless errors are discovered, the file is read in quietly. If `load` is interrupted by pressing `Control-c`, the function skips the rest of the file being loaded.

SKILL has an autoload feature that allows applications to load functions into SKILL on demand. If a function being run is undefined, SKILL checks to see if the name of the function (a symbol) has a property called `autoload` attached to it. If the property exists, its value, which must be either a string or an expression that evaluates to a string, is used as the name of a file to be loaded. The file should contain a definition for the function that triggered the autoload. Processing proceeds normally after the function is defined.

Arguments

t_fileName — File to be loaded. Uses the file name extension to determine the language mode to use.
Valid values:

`.ils`	Means the file contains SKILL++ code.
`.il`	Means the file contains SKILL code.
`.ocn`	Means the file contains OCEAN commands (with SKILL or SKILL++ commands)

t_password — Password, if *t_fileName* is an encrypted file.

Value Returned

`t`	Returns `t` if the file is successfully loaded.

Example

```
load( "test.ocn" )
```

Loads the `test.ocn` file.

```
procedure( trLoadSystem()
    load( "test.il" ) ;;; SKILL code
    load( "test.ils" );;; SKILL++ code
    ) ; procedure
```

You might have an application partitioned into two files. Assume that `test.il` contains SKILL code and `test.ils` contains SKILL/SKILL++ code. This example loads both files.

newline

```
newline( [p_outputPort] )
     => nil
```

Description

Prints a newline (backslash `n`) character and then flushes the output port.

Arguments

p_outputPort	Output port. Defaults value: `poport`

Value Returned

`nil`	Prints a newline and then returns `nil`.

Example

```
print( "Hello" ) newline() print( "World!" )
"Hello"
"World!"
=> nil
```

Prints a newline character after `Hello`.

outfile

```
outfile( S_fileName [t_mode] )
     => p_outport/nil
```

Description

Opens an output port ready to write to a file.

Various print commands can write to this file. Commands write first to a character buffer, which writes to the file when the character buffer is full. If the character buffer is not full, the contents are not written to the file until the output port is closed or the `drain` command is entered. Use the close or `drain` command to write the contents of the character buffer to the file. The file can be specified with either an absolute path or a relative path. If a relative path is given and the current SKILL path setting is not `nil`, all directory paths from SKILL path are checked in order, for that file. If found, the system overwrites the first updatable file in the list. If no updatable file is found, it places a new file of that name in the first writable directory.

Arguments

S_fileName	Name of the file to open or create.
t_mode	Mode in which to open the file. If `a`, the file is opened in append mode; If `w`, a new file is created for writing (any existing file is overwritten). Default value: `w`

Value Returned

p_outport	An output port ready to write to the specified file.
`nil`	returns `nil` if the named file cannot be opened for writing. An error is signaled if an illegal mode string is supplied.

Example

```
p = outfile( "/tmp/out.il" "w" )
=> port:"/tmp/out.il"
```

Opens the `/tmp/out.il` port.

```
outfile( "/bin/ls" )
=> nil
```

Returns `nil`, indicating that the specified port could not be opened.

pfile

```
pfile( [S_fileName | p_port] )
     => p_port/nil
```

Description

Opens an output port ready to write to a file or returns the name of an existing port indicating that it is available.

This command is similar to the outfile command when a valid `S_fileName` is specified. When `p_port` is specified, it returns the file port that is currently being used by `p_port`. When no argument is specified, it opens the `stdout` port.

Arguments

S_fileName	Name of the file to open or create.
p_port	Retrieves the name of the file port that is being used.

Value Returned

p_port	The ID of the port that was opened, or `stdout`.
`nil`	Returns `nil` if the named file cannot be opened for writing.

Example

```
p = pfile( "/tmp/out.il" "w" )
=> port:"/tmp/out.il"
```

Opens the `/tmp/out.il` port.

```
pfile( "/bin/ls" )
=> nil
```

Returns `nil`, indicating that the specified port could not be opened.

```
p = pfile()
=> port:"*stdout*"
```

Returns stdout as the file port indicating that stdout has been opened.

```
pfile( p )
=> port:"/tmp/out.il"
```

Returns the file port.

printf

```
printf( t_formatString [g_arg1 ...] )
     => t
```

Description

Writes formatted output to *poport*, which is the standard output port.

The optional arguments following the format string are printed according to their corresponding format specifications. Refer to the "Common Output Format Specifications" table for `fprintf` in the *Cadence SKILL Language User Guide.*

`printf` is identical to `fprintf` except that it does not take the *`p_port`* argument and the output is written to `poport`.

Arguments

`t_formatString`	Characters to be printed verbatim, intermixed with format specifications prefixed by the "%" sign.
`g_arg1`…	Arguments following the format string are printed according to their corresponding format specifications.

Value Returned

`t`	Prints the formatted output and returns `t`.

Example

```
x = 197.9687 => 197.9687
printf( "The test measures %10.2f." x )
```

Prints the following line to `poport` and returns `t`.

```
The test measures 197.97. => t
```

println

```
println( g_value [p_outputPort] )
     => nil
```

Description

Prints a SKILL object using the default format for the data type of the value, and then prints a newline character.

A newline character is automatically printed after printing *g_value*. The `println` function flushes the output port after printing each newline character.

Arguments

g_value	Any SKILL value.
p_outputPort	Port to be used for output. Default value: `poport`

Value Returned

`nil`	Prints the given object and returns `nil`.

Example

```
for( i 1 3 println( "hello" ))
"hello"
"hello"
"hello"
=> t
```

Prints hello three times. `for` always returns `t`.

15

OCEAN Commands in XL Mode

The following OCEAN XL commands provide for multi-test ADE XL support in OCEAN.

ocnSetXLMode

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

Description

Sets OCEAN mode to XL. In this mode, multi-tests can be created and run. Also Corners, MonteCarlo commands can be given. Once mode is set to XL, it cannot be reverted.

Arguments

None

Value Returned

`t`	Returns `t` if the mode is set to XL.
`nil`	Returns `nil` otherwise.

Example

```
ocnSetXLMode()
```

ocnxlBeginTest

```
ocnxlBeginTest(t_testName)
    => t / nil
```

Description

This command indicates the beginning of the test specified by `testName`. Subsequent commands populate this test. The test specification ends when `ocnxlEndTest()` command is given. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

t_testName	The name of the test.

Value Returned

`t`	Returns `t` if its able to begin the test.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlBeginTest("test_one")
```

ocnxlEndTest

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

Description

This command indicates the end of the current test's specification. See help on *`ocnxlBeginTest()`*. This command works only in XL mode. See help on *`ocnSetXLMode()`*.

Arguments

None.

Value Returned

`t`	Returns `t` if the test setup completes.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlBeginTest("test_one")
design("solutions" "ampTest" "schematic")
simulator('spectre)
ocnxlEndTest()
```

ocnxlEndXLMode

```
ocnxlEndXLMode()
    => t
```

Description

This command ends the XL mode. Also see help on `ocnxlSetXLMode()`.

Arguments

None.

Value Returned

`t`	Returns `t` if it exits the XL mode.

Example

```
ocnxlEndXLMode()
```

ocnxlFeasibilityAnalysisOptions

```
ocnxlFeasibilityAnalysisOptions(
     t_refPoint
     t_meetAllGoals
     t_effort)
     => t | nil
```

Description

Specifies options for the Feasibility Analysis run mode. See help on ocnxlRun for help on run modes. This command works only in XL mode.

Arguments

t_refPoint	Specifies whether to use a reference point that you have created as a starting place for sizing. It is optional to set this argument when the algorithm specified with the `t_effort` argument is `neocircuitGlobal`. For other values of `t_effort`, set this as `1`. Default value is `0`.
t_meetAllGoals	Specifies the stopping criteria for the analysis. By default, it is set to `1` and all operating region specifications are to be met. **Note:** Currently, you cannot set this argument to any value other then `1`.
t_effort	Specifies the name of algorithm for optimizing the design to meet the operating region specifications. Possible values are: `neocircuitGlobal`, `conjugateGradient`, `brentPowell`, `hookeJeeves`. The default algorithm is `neocircuitGlobal`.

Value Returned

t	Returns t when successful
nil	Otherwise, returns nil

Example

```
ocnxlFeasibilityAnalysisOptions(?effort "conjugateGradient")
t
```

ocnxlSelectTest

```
ocnxlSelectTest(t_testName)
     => t / nil
```

Description

Lets you select a test. List of test names can be obtained by `ocnxlGetTests()` command. See help on `ocnxlGetTests()`. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

`t_testName`	The name of the test.

Value Returned

`t`	Returns `t` if the test is selected.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlSelectTest("test_two") => t
Sets "test_two" as the currently selected test.
```

ocnxlSensitivityOptions

```
ocnxlSensitivityOptions(
     t_haveDesignParams
     t_haveProcessParams
     t_haveDcOp
     t_haveDesignParamSteps
     t_designParamPercentage
     t_processSigmaSteps
     t_processConfidenceIntervalUniform
     t_processMethod
     t_dcOpInfo

     )
     => nil
```

Description

Specifies options for the Sensitivity Analysis run mode. See help on ocnxlRun for help on run modes. This command works only in XL mode.

Arguments

t_haveDesignParams Specifies if you want to vary global variables and device parameters. Default value is 0. If you set this to 1, ensure that you have specified at least one sweep variable or parameter and also created a reference point. You also need to set either *t_haveDesignParamSteps* or *t_designParamPercentage*.

t_haveProcessParams Specifies if you want to vary the statistical process and mismatch parameters. Default value is 0.

t_haveDcOp Specifies if you want to save the sensitivity data for specific DC operating point parameters. Default value is 0.

t_haveDesignParamSteps
Specifies that you want to vary global variable and device parameter values by a single step from the reference values specified for global variables and parameters in the reference point.

t_designParamPercentage
Specifies the percentage of the range of a variable or

parameter's value from the reference values by which the process parameters need to be varied. Value range is between 0 and 100.

t_processSigmaSteps

Specifies the number of standard deviations for statistical parameters with normal or log normal distribution. Default value is `1`.

t_processConfidenceIntervalUniform

Specifies the percentage range by which statistical parameters with uniform distribution need to be varied. Value range is between `0` and `50`.

t_processMethod — Specifies whether the statistical parameters to be used are process, mismatch or both. Default value is `process`. Possible values are `process`, `mismatch` and `all`.

t_dcOpInfo — Specifies DC operating point parameters as input parameters for the sensitivity analysis run.

Value Returned

nil — Returns nil

Example

```
ocnxlSensitivityOptions( ?haveDesignParams "1" ?haveProcessParams "0" ?haveDcOp
"1" ?haveDesignParamSteps "1" ?designParamPercentage "10" ?processSigmaSteps "1"
?processConfidenceIntervalUniform "20" ?processMethod "Process" ?dcOpInfo
"voltage_divider:voltage_divider:1%/R0%i%analogLib/res/spectre%Master%analo gLib/
res/spectre#"
nil
```

ocnxlSensitivityVars

```
ocnxlSensitivityVars(
     l_varValueList

     )
     => t | nil
```

Description

Specifies a list of sensitivity variables for the given setup.

This command works only in XL mode.

Arguments

l_varValueList	List of variable and values combination. Each list item contains the name of sensitivity variable, a list of sweep values for that variable, and a reference value.

Value Returned

t	Returns `t` if the list of sensitivity variables is set.
nil	Returns `nil` otherwise.

Example

```
ocnxlSensitivityVars(list '("CAP" "100f 200f 300f" "200f") '("RES" "1K 1.5K 2K"
"1.5K"))
```

ocnxlSweepVar

```
ocnxlSweepVar(t_varName t_varValue)
     => t / nil
```

Description

Lets you define a sweep variable along with its value. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

t_varName	Name of the variable.
t_varValue	Value of the variable and the specification for the sweep.

Value Returned

`t`	Returns `t` if the sweep is set.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlSweepVar("CAP" "5p") =>t
```

ocnxlSweepParam

```
ocnxlSweepParam(t_paramName t_paramValue)
     => t / nil
```

Description

Lets you define a sweep parameter along with its value. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

t_paramName	Name of the parameter.
t_paramValue	Value of the parameter.

Value Returned

`t`	Returns `t` if sweep for the parameter is set.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlSweepParam("solutions/ampTest/schematic/R1/r" "10K") =>t
```

ocnxlSweepsAndCornersOptions

```
ocnxlSweepsAndCornersOptions
     (t_submitpointenabled )
     => t/nil
```

Description

Lets you specify options for Single Run, Sweeps and Corners runs.

Arguments

`t_submitpointenabled`	Specify `t` to override the active setup with the submit point information.

Value Returned

`t`	Returns `t` if the options are set successfully.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlSweepsAndCornersOptions(t) => t
```

ocnxlCorner

```
ocnxlCorner(t_cornerName l_cornerDetails)
     => t / nil
```

Description

Lets you define a corner.`cornerDetails` is a list of elements where each element is (*t_type t_varName t_value*). Available types are `variable`, `parameter`, and `model`. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

t_cornerName	Name of the corner.
l_cornerDetails	Details of the corner. Details is a list of items where each item has a tag, name, and a value. The tag can be of 3 types — `variable`, `parameter`, and `model`.

Value Returned

`t`	Returns `t` if the corner is defined.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlCorner("C0" '( ("variable" "CAP" "2p") ("variable" "T" "78"))) => t
```

ocnxlCornerVars

```
ocnxlCornerVars(l_varValueList )
     => t | nil
```

Description

Specifies a list of corner variables (along with their minimum, maximum, and reference values) to be used to run the Create Worst Care Corner simulation run mode. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

l_varValueList	List of corner variable-value combination list. Each list item contains a list of variable name, minimum and maximum values for the variable, and a reference value.

Value Returned

`t`	Returns `t` if the corner variables for Create Worst Care Corner simulation run mode are set successfully.
`nil`	Returns `nil` otherwise.

ocnxlWorstCaseCornersOptions

```
ocnxlWorstCaseCornersOptions(
     ?algorithm t_algorithm
     grouprun g_grouprun
     )
     => t | nil
```

Description

Sets the algorithm to be used while running the worst case corner simulation.

Arguments

t_algorithm	Specifies the algorithm based on which you want to create the worst case corners. Possible values: `OFAT 3-level`, `OFAT Sweep`, `2^K Factorial`, `Central Composite Design`, and `Full Factorial`.
g_grouprun	Specifies whether to create a group run with child histories. Possible values: `0` or `1`. The value `1` is used for OFAT and Central Composite Design algorithms.

Value Returned

`t`	Returns `t` if the option is set successfully.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlWorstCaseCornersOptions( ?algorithm "OFAT 3-level" ?grouprun "1" )
```

ocnxlDisableTest

```
ocnxlDisableTest(t_testName)
     => t / nil
```

Description

Lets you disable a test. A disabled test will not be run when `ocnxlRun()` command is fired. See help on `ocnxlRun()`. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

`t_testName`	Name of the test.

Value Returned

`t`	Returns `t` if the test is disabled.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlDisableTest("test_two") => t
```

ocnxlDisableSweepVar

```
ocnxlDisableSweepVar(t_varName)
     => t / nil
```

Description

Lets you disable a sweep variable. A disabled sweep is not run when `ocnxlRun()` command is fired. See help on `ocnxlRun()`. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

t_varName	Name of the variable.

Value Returned

`t`	Returns `t` if the sweep variable is disabled.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlDisableSweepVar("CAP") => t
```

ocnxlDisableSweepParam

```
ocnxlDisableSweepParam(t_paramName)
     => t / nil
```

Description

Lets you disable a sweep parameter. A disabled parameter is not run when `ocnxlRun()` command is fired. See help on `ocnxlRun()`. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

`t_paramName`	Name of the parameter.

Value Returned

`t`	Returns `t` if the sweep parameter is disabled.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlDisableSweepParam("solutions/ampTest/schematic/R1/r") => t
```

ocnxlDisableCornerForTest

```
ocnxlDisableCornerForTest(t_cornerName t_testName)
     => t / nil
```

Description

Lets you disable a corner for a test. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

t_cornerName	Name of the corner.
t_testName	Name of the test.

Value Returned

`t`	Returns `t` if the corner of the test is disabled.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlDisableCornerForTest("C0" "test_one")
```

ocnxlGlobalOptimizationOptions

```
ocnxlGlobalOptimizationOptions(
     [ ?runFullEvaluation t_runFullEvaluation ] [ ?refPoint t_refPoint ]
     [ ?meetAllGoals t_meetAllGoals] [ ?timeLimit t_timeLimit ]
     [ ?numPoints t_numPoints ] [ ?noImprvPoints t_noImprvPoints ]
     [ ?pointsAfterAllSpecsSatisfied t_pointsAfterAllSpecsSatisfied]
     [ ?startingstateorpoint t_startingstateorpoint]
     [ ?startingstatename t_startingstatename]
     )
     => t / nil
```

Description

Lets you specify options for global optimization run. See help on ocnxlRun() for run modes. This command works only in XL mode. See help on `ocnSetXLMode()`

Arguments

t_runFullEvaluation Sets the program to run full optimization. Default for `runFullEvaluation` is `0`. Possible values are `0` and `1`.

t_refPoint Sets the current point as a starting place for sizing. Default for `refPoint` is `0`. Possible values are `0` and `1`.

t_meetAllGoals Sets the program to run optimization only until all specifications are met. Default for `meetAllGloals` is `0`. Possible values are `0` and `1`.

t_timeLimit Sets the time limit (in minutes) for the optimization run. Default for `timeLimit` is "".

t_numPoints the maximum number of points for the optimization run. Default for `numPoints` is "".

t_noImprvPoints Default for `noImprvPoints` is "".

t_pointsAfterAllSpecsSatisfied Sets the number of points to be run after all specifications are satisfied. Default for `t_pointsAfterAllSpecsSatisfied` is "".

t_startingstateorpoint Sets the starting point for the simulation run as reference point or starting state. Default for *t_startingstateorpoint* is "". Possible values are `Starting State` or `Reference Point`.

t_startingstatename Sets the starting state if the *startingstateorpoint* argument is set to `Starting State`.

Value Returned

`t` Returns `t` if options are specified for global optimization run.

`nil` Returns `nil` otherwise.

Example

```
ocnxlGlobalOptimizationOptions(?runFullEvaluation "1" )
# Sets global optimization to be run only until all specifications are met.
ocnxlGlobalOptimizationOptions(?startingstateorpoint "Starting State"
?startingstatename "Optimization.PointID.46")
# Sets the optimization to be run with starting point as "Optimization.PointID.46"
state.
```

ocnxlJobSetup

```
ocnxlJobSetup (l_setupOptions)
     => t / nil
```

Description

Specifies various job setup details for a simulation job.

This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

`l_setupOptions`	The list of setup options. You can set the following options using this argument: - `distributionmethod`: Specifies the location where the job has to run. The possible values for distribution method are: `Local`, `Remote-Host`, `Command`, `LBS`. Default value: `Local` **Note:** When you choose `LBS` distribution method, by default OCEAN uses the LBS Distributed Resource Management Systems (DRMS). You can choose to use LSF or SGE DRMS by setting the `LBS_CLUSTER_MASTER` and `LBS_BASE_SYSTEM` environment variables in your `.cshrc` file. For more details, refer to Setting Up to Use Distributed Processing Option in *Virtuoso Analog Distributed Processing Options User Guide*. - `maxjobs`: Specifies the maximum number of jobs that can run at any time during the given session. Default value: `1` - `starttimeout`: Specifies the time (number of seconds) to wait for the icrp process (a process that runs the specific job) to report back that it has started the job. The wait time starts as the job is submitted. Default value: `300` - `startmaxjobsimmed`: Immediately submits all the specified maximum number of jobs. Default value: `1` - `configuretimeout`: Specifies the time (number of seconds) to wait for the icrp process to report back that it has configured the job. The wait time starts as soon as a job configure command is sent. Default value: `300`

- `lingertimeout`: Specifies the time (number of seconds) after which you want the program to kill the icrp process after the simulations finish. Default value: `300`

- `runtimeout`: Specifies the time (number of seconds) to wait for the icrp process to report back that it has run the job. The wait time starts as soon as the run command for the job is sent. Default value: `-1`, which means ADE XL keeps waiting for infinite time for the icrp process to report back.

- `showoutputlogerror`: Displays the output log files of all error points in the test. Default value: `0`

- `showerrorwhenretrying`: Displays the output log file on the occurrence of an error for a test, even if the ADE XL distribution system is retrying the test. Default value: `1`

- `reconfigureimmediately`: When running multiple runs in the same ADE XL session, specifies that a completed job be reassigned from the current run to a new run. Default value: `1`

- `jobqueue`: Specifies the name of the queue. If no queue is defined, the job is placed in the default queue. This option is used only when LBS or LSF DRMS is used.

- `jobhostname`: Specifies the name of the host on which the job will run. If no host is specified, the system assigns the job to any available host. This option is used when `distributedmethod` is set as `Remote-Host` or when `distributedmethod` is set as LBS and either LBS or LSF DRMS is used.

- `parallelnumprocs`: Specifies the number of parallel processors to be used. This option is used only when LSF DRMS is used.

- `jobresourcerequirements`: Specifies a string describing the resources required to run the job when LSF DRMS is used.

To know more about the format of the resource requirements string, refer to <u>LSF Resource Requirement String Format</u> in *Virtuoso Analog Distributed Processing Options User Guide*.

- `sgehardresource`: Specifies requirements for hardware resources for the job to be run when SGE DRMS is used.

- `sgesoftresource`: Specifies requirements for software resources for the job to be run when SGE DRMS is used.

- `sgepriority`: Specifies priority for the job being submitted when SGE DRMS is used.

- `sgeparallelenv`: Specifies name of a parallel environment for the job to be run when SGE DRMS is used.

- `jobsubmitcommand`: Specifies the command you want to use to start jobs. This option is used when `distributionmethod` is set as `Command`.

Value Returned

`t`	Returns `t` if the job setup information is set successfully.
`nil`	Returns `nil` otherwise.

Example 1

The following command sets the job policy for a local job:

```
ocnxlJobSetup( '("configuretimeout" "300" "distributionmethod" "Local"
"lingertimeout" "300" "maxjobs" "1" "preemptivestart" "1" "reconfigureimmediately"
"1" "runtimeout" "-1" "showerrorwhenretrying" "1" "starttimeout" "300" ) )
```

Example 2

The following command sets the job policy for a bsub command for LSF (command):

```
ocnxlJobSetup ( '( "distributionmethod" "command" "bsub -I -q queue1") )
```

Example 3

The following command sets the job policy for LSF (LBS mode):

```
ocnxlJobSetup ( '( "distributionmethod" "LBS" configuretimeout" "300"
"lingertimeout" "300" "maxjobs" "1" "jobqueue" "fast" "jobhostname" "sun15") )
```

ocnxlLocalOptimizationOptions

```
ocnxlLocalOptimizationOptions( [ ?effort t_effort ]
     [ ?runFullEvaluation t_runFullEvaluation ] [ ?meetAllGoals t_meetAllGoals ] [
     ?timeLimit t_timeLimit ] [ ?numPoints t_numPoints ]
     [ ?startingstateorpoint t_startingstateorpoint]
     [ ?startingstatename t_startingstatename] )
     => t / nil
```

Description

Lets you specify options for local optimization run. See help on `ocnxlRun()` for run modes.

Arguments

t_effort	Value for effort. Default for effort is `coarse`. Possible values are `fine` and `coarse`.
t_runFullEvaluation	Sets the program to run full optimization. Default for `runFullEvaluation` is `0`. Possible values are `0` and `1`.
t_meetAllGoals	Sets the program to run optimization only until all specifications are met. Default for `meetAllGloals` is `0`. Possible values are `0` and `1`.
t_timeLimit	Sets the time limit (in minutes) for the optimization run. Default for `timeLimit` is "".
t_numPoints	Sets the maximum number of points for the optimization run. Default for `numPoints` is "".
t_startingstateorpoint	Sets the starting point for the simulation run as reference point or starting state. Default for *t_startingstateorpoint* is "". Possible values are `Starting State` or `Reference Point`.
t_startingstatename	Sets the starting state if the *startingstateorpoint* argument is set to `Starting State`.

Value Returned

`t`	Returns `t` if options are specified for local optimization run.

`nil` Returns `nil` otherwise.

Example

```
ocnxlLocalOptimizationOptions(?effort "coarse" )
# Sets coarse as the effort for local optimization run.
```

```
ocnxlLocalOptimizationOptions(?startingstateorpoint "Starting State"
?startingstatename "Optimization.PointID.46")
```

```
# Sets the optimization to be run with starting point as “Optimization.PointID.46”
state.
```

ocnxlModelGroup

```
ocnxlModelGroup( t_modelGroupName l_modelFileSetup)
    => t/nil
```

Description

Lets you add and define a new model group. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

t_modelGroupName	Name of the new model group.
l_modelFileSetup	List of model file spec. Model file spec: *(t_modelFilePath* *[?section t_section]* *[?enabled g_enabled]* *[?test t_test]* *[?block t_block]* *)*

Value Returned

`t`	Returns `t` if a new model is defined.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlModelGroup( "F2"

'(

( "/myModels/Models/model1.scs" ?enabled nil  ?section "")

( "/myModels/Models/model2.scs" ?section "")

( "/myModels/Models/model3.scs" ?enabled nil  ?section "")

))
```

ocnxlOutputOceanScript

```
ocnxlOutputOceanScript( t_script [ ?name t_outputName ] [ ?plot plot ] [ ?save save
    ] [ ?evalType t_evaltype ])
    => t/nil
```

Description

Adds an OCEAN script based output in the current test being specified. A test's specification begins with `ocnxlBeginTest()`. See help on `ocnxlBeginTest()`.

This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

t_script	Name and location of the script file.
t_outputName	Name of the output file.
plot	Specifies if the values are to be plotted.
save	Specifies if the output are to be saved.
t_evaltype	Whether to evaluate the OCEAN script for a design point or across all corners for a design points. This is an optional argument. Valid Values: `corners`, `point` Default Value: `point`

Value Returned

`t`	Returns `t` if the output is generated.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlOutputOceanScript( "/tmp/my_measure.ocn" ?name "MAX" ?plot t ?save t ) => t
Adds "MAX" in the outputs.
```

ocnxlOutputMatlabScript

```
ocnxlOutputMatlabScript( t_script [ ?name t_outputName ] [ ?plot plot ] [ ?save
     save ] )
     => t/nil
```

Description

Adds a MATLAB script based output in the current test being specified. A test's specification begins with `ocnxlBeginTest()`. See help on `ocnxlBeginTest()`.

This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

t_script	Name and location of the script file.
t_outputName	Name of the output file.
plot	Specifies if the values are to be plotted.
save	Specifies if the output are to be saved.

Value Returned

t	Returns `t` if the output is generated.
nil	Returns `nil` otherwise.

Example

```
ocnxlOutputMatlabScript( "/tmp/my_measure.m" ?name "MAX" ?plot t ?save t ) => t
Adds "MAX" in the outputs.
```

ocnxlOutputOpRegion

```
ocnxlOutputOpRegion
     ( t_expr
     [ ?plot plot ]
     [ ?save save ]
     )
     => t | nil
```

Description

Adds an operating region specification in the current test being specified. Note that there can be only one operating region for a test. A test specification begins with ocnxlBeginTest. This command works only in XL mode. See help on ocnSetXLMode.

Arguments

t_expr	Operating region expression
t_name	Name of the operating region output
t_plot	Specifies if the results for the output are to be plotted Specify: ■ `t` to plot the results ■ `nil` to disable plotting of the results
t_save	Specifies if the results for the output are to be saved Specify: ■ `t` to save the results ■ `nil` to disable saving of the results

Return Values

t	Returns `t` if the output is added
nil	Returns `nil` otherwise

Example

```
ocnxlOutputOpRegion( "((\"enabled\" \"/amp/M1\" \"\" \"Schematic\" \"vgs-vth\"
\">\" \"25m\" \"amp.M1\"))" ?name "Op_Region" ?plot t ?save t)
```

ocnxlMonteCarloOptions

```
ocnxlMonteCarloOptions( [ ?mcmethod t_mcmethod ] [ ?mcNumPoints t_mcNumPoints ]
     [ ?samplingMode t_samplingMode ] [ ?saveAllPlots t_saveAllPlots ]
     [ ?saveProcess t_saveProcess ] [?saveMismatch t_saveMismatch ]
     [ ?useReference t_useReference ] [ ?donominal t_donominal ]
     [ ?monteCarloSeed t_monteCarloSeed ]
     [ ?mcStartingRunNumber t_mcStartingRunNumber ]
     [ ?dutSummary t_dutSummary ]
     [ ?ignoreFlag t_ignoreFlag ]
     [ ?mcNumBins t_mcNumBins ]
     )
     => t / nil
```

Description

Lets you specify options for Monte Carlo runs. See help on `ocnxlRun()` for run modes.

Arguments

t_mcmethod	Sets the statistical variation method for Monte Carlo runs. Default for `mcmethod` is `all`. Possible values are `global`, `mismatch` and `all`.
t_mcNumPoints	Sets the number of points you want to simulate for Monte Carlo runs. Default for `mcNumPoints` is `100`.
t_samplingMode	Sets the default statistical sampling method for Monte Carlo runs. Default for `samplingMode` is `random`. Possible values are `random`, `orthogonal`, and `lhs` (Latin Hypercube).
t_saveAllPlots	Saves raw data (`psf` files) for every Monte Carlo iteration so that you can plot a family of curves. Default for `saveAllPlots` is 0. Possible values are `0` and `1`.
t_saveProcess	Controls whether ‘process’ parameters need to be saved to the results database. Default value is `1`. Possible values are `0` and `1`.
t_saveMismatch	Controls whether ‘mismatch’ parameters need to be saved to the results database. Default value is `0`. Possible values are `0` and `1`.

t_useReference	Specifies whether to use a schematic point or a reference point that you have created as a starting place for sizing. Possible values are `0` and `1`. The default value is 0.
t_donominal	Specifies whether to run a simulation at the reference point prior to beginning the Monte Carlo process. Possible values are `0` and `1`. If set to `1`, Spectre will run a simulation at the reference point, and, if this fails, then the sampling process is not initiated and the simulation stops. The default value is 1.
t_monteCarloSeed	Specifies a different seed for Monte Carlo runs. Default for `monteCarloSeed` is `12345`.
t_mcStartingRunNumber	Specifies a starting run number for Monte Carlo runs. Default for `mcStartingRunNumber` is `1`.
t_dutSummary	Specifies a list of design under test (DUT) instances for Monte Carlo runs. In this list, you can specify the instances and devices to which mismatch variations must be applied. The format to specify the list is as given below: `<testname%instances%Libname/Cellname/Viewname%Master#testname%instances%modelname%Subcircuit#testname%instances%Schematic%Schematic>` where two DUT instances in the list are separated by a `#` (hash). For example: *"opamp090:full_diff_opamp_AC:2:1%/I21%acOpenDiff%Subcircuit#opamp090:full_diff_opamp_AC:2:1%/I0/I1%opamp090/ampn/schematic%Master#opamp090:full_diff_opamp_AC:2:1%/I0/M5A, /I0/M3A%Schematic%Schematic"* Default for `dutSummary` is "".
t_ignoreFlag	Optional argument to specify if the user wants to apply mismatch variations to instances specified with `dutSummary`. Default value is `0`. Set it to `1` if you do not want to apply mismatch variations to instances.

t_mcNumBins	Optional argument to specify the number of bins. Set this value when `samplingMode` is `lhs`. Default for `mcNumBins` is "". If not set, simulator uses its own default number of bins. For example, Spectre calculates the number of bins as given below: `numBins = max(t_mcNumBins, (t_mcNumPoints + t_mcStartingRunNumber -1))`

Value Returned

`t`	Returns `t` if options for montecarlo run are specified.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlMonteCarloOptions(?mcmethod "all" ?mcNumPoints “100” ?samplingMode “lhs”
?saveAllPilots “0” ?monteCarloSeed "" ?mcStartingRunNumber "" ?dutSummary ""
?saveProcess “1” ?saveMismatch “0” ?useReference “0” ?doNominal “1” ?mcNumBins
"100")
```

ocnxlPutInfoSpec

```
ocnxlPutToleranceSpec(
     t_testName
     t_resultName
     )
     => t / nil
```

Description

Specifies an info spec for a result. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

t_testName	The name of the test.
t_resultName	The name of the result.

Value Returned

`t`	Returns `t` if the specification is specified.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlPutInfoSpec("test_one" "VT(\"/out\")") => t

# An info spec “test_one” is set for the expression “VT(\"/out\”.
```

ocnxlPutToleranceSpec

```
ocnxlPutToleranceSpec(t_testName t_resultName t_value
     s_type t_tolerance g_weight)
     => t / nil
```

Description

Lets you specify a tolerance spec for a result. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

t_testName	The name of the test.
t_resultName	The name of the result.
t_value	The target value.
s_type	The type of tolerance.
t_value	The tolerance value.
g_weight	The weighting factor for the spec.

Value Returned

`t`	Returns `t` if the specifications are specified.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlPutToleranceSpec("test_one" "VT('/out')" "5.0" 'percent "10" 4) => t
# Spec is defined that transient voltage for /out signal must be 5.0 volts with
tolerance 10%. The weighting factor for the spec is 4.
```

ocnxlPutMinSpec

```
ocnxlPutMinSpec(t_testName t_resultName t_minValue g_weight)
     => t / nil
```

Description

Lets you specify a minimum spec for a result. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

t_testName	Name of the test.
t_resultName	Name of the result on which you specify the test.
t_minValue	The minimum value.
g_weight	The weighting factor for the spec.

Value Returned

`t`	Returns `t` if the specification is set.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlPutMinSpec("test_one" "VT('/out')" "3.5" 4) => t
Spec is defined that minimum transient voltage for /out signal
must be 3.5 volts. The weighting factor for the spec is 4.
```

ocnxlPutMaxSpec

```
ocnxlPutMaxSpec(t_testName t_resultName t_maxValue g_weight)
     => t / nil
```

Description

Lets you specify a maximum spec for a result. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

t_testName	Name of the test.
t_resultName	Name of the result on which you specify the test.
t_maxValue	The maximum value.
g_weight	The weighting factor for the spec.

Value Returned

`t`	Returns `t` if the specification is set.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlPutMaxSpec("test_one" "VT('/out')" "6.5" 4) => t
Spec is defined that maximum transient voltage for /out signal
must be 6.5 volts. The weighting factor for the spec is 4.
```

ocnxlPutGreaterthanSpec

```
ocnxlPutGreaterthanSpec(t_testName t_resultName t_Value g_weight)
     => t / nil
```

Description

Lets you specify that value for a result must be greater than a threshold value. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

t_testName	Name of the test.
t_resultName	Name of the result on which you specify the test.
t_Value	The threshold value.
g_weight	The weighting factor for the spec.

Value Returned

`t`	Returns `t` if the specification is set.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlPutGreaterthanSpec("test_one" "VT('/out')" "3.5" 4) => t
Spec is defined that transient voltage for /out signal
must always be greater than 3.5 volts.The weighting factor for the spec is 4.
```

ocnxlPutLessthanSpec

```
ocnxlPutLessthanSpec(t_testName t_resultName t_Value g_weight)
     => t / nil
```

Description

Lets you specify that value for a result must be less than a threshold value. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

t_testName	Name of the test.
t_resultName	Name of the result on which you specify the test.
t_Value	The threshold value.
g_weight	The weighting factor for the spec.

Value Returned

`t`	Returns `t` if the specification is set.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlPutLessthanSpec("test_one" "VT('/out')" "6.5" 4) => t
Spec is defined that the transient voltage for /out signal
must always be less than 6.5 volts. The weighting factor for the spec is 4.
```

ocnxlPutRangeSpec

```
ocnxlPutRangeSpec(t_testName t_resultName t_maxValue t_minValue g_weight)
     => t / nil
```

Description

Lets you specify a range spec for a result. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

t_testName	Name of the test.
t_resultName	Name of the result on which you specify the test.
t_maxValue	The maximum value.
t_minValue	The minimum value.
g_weight	The weighting factor for the spec.

Value Returned

`t`	Returns `t` if the specification is set.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlPutRangeSpec("test_one" "VT('/out')" "6.5" "3.5" 4) => t
Spec is defined that maximum transient voltage for /out signal
must be 6.5 volts and minimum must be 3.5 volts. The weighting factor for the spec
is 4.
```

ocnxlPutTargetSpec

```
ocnxlPutTargetSpec(t_testName t_resultName t_value)
     => t / nil
```

Description

Lets you specify a target spec for a result. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

t_testName	Name of the test.
t_resultName	Name of the result on which you specify the test.
t_value	The target value.

Value Returned

t	Returns `t` if the specification is set.
nil	Returns `nil` otherwise.

Example

```
ocnxlPutTargetSpec("test_one" "VT('/out')" "5.0") => t
Spec is defined that transient voltage for /out signal
must be 5.0 volts.
```

ocnxlResultsLocation

```
ocnxlResultsLocation(t_resultsDir)
     => t / nil
```

Description

Sets the results directory to the specified location. All results database and log files are saved in the /`libraryName/cellName/<target-view>/results/data/` directory at this location.

By default, data is saved at the `<target-view>/results/data` directory. See help on ocnxlTargetCellView.

Note: If you do not specify the results directory and you open the ADE XL view in read-only mode or do not have write permissions in the ADE XL view, the program writes results databaseinformation and run log files to `libraryName/cellName/adexl/results/data/<history_item>` in the project directory, set by ocnxlProjectDir.

This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

t_resultsDir	Location of the results directory.

Value Returned

`t`	Returns `t` if the results location is set.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlResultsLocation("/home/ocnuser")
```

ocnxlRunSetupSummary

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

Description

Generates the run setup summary. It shows how many tests, sweeps and corners are there and whether they are enabled. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

None.

Value Returned

`t`	Returns `t` if the summary is generated.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlRunSetupSummary()
```

ocnxlSamplingOptions

```
ocnxlSamplingOptions ( [ ?points t_numberOfPoints ])
     => t / nil
```

Description

Lets you specify options for sampling run. See help on ocnxlRun () for run modes. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

t_numberOfPoints	Specifies the number of points. The default value for points is `200`.

Value Returned

`t`	Returns `t` if the options for the run are specified.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlSamplingOptions(?points "500")
# Sets 500 as the number of points for sampling run.
```

ocnxlSetupLocation

```
ocnxlSetupLocation(t_setupDir)
     => t / nil
```

Description

Sets the setup directory to the specified location. All setup data goes into this location. By default data goes into the target cell view. See help on `ocnxlTargetCellView()`. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

t_setupDir	Location of the setup directory.

Value Returned

`t`	Returns `t` if the location of the setup directory is set.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlSetupLocation("/home/ocnuser")
```

ocnxlSizeOverCornersOptions

```
ocnxlSizeOverCornersOptions(
      [(soclazy )]
      socoptmethod
      socreferencepoint
      soceffort
      soctimelimit
      socmaxpoints
      sociterations
      socstopifnoimprovement
      )
      => nil
```

Description

Provides run options for the Size Over Corners run.

Arguments

soclazy	Specifies the evaluation type for the sizing run. between `Conditional` (`1`) and `Full` (`0`) evaluation types for the sizing run. The default value is `1`.
socoptmethod	Specifies between Optimization algorithms for iteration sizing runs. The default value is `neocircuitGlobal`. Other values are `brentPowell`, `hookeJeeves`, and `conjugateGradient`.
socreferencepoint	Specifies if reference point should be used. The default value is `0`.
soceffort	This argument is currently not supported.
soctimelimit	Specifies the time limit for the run.
socmaxpoints	Specifies the maximum number of points processed per iteration. The default value is `3000`.
sociterations	Specifies the maximum number of sizing iterations for the Size Over Corners run. The default value is `3`.

socstopifnoimprovement

Specifies if the optimization run should stop if there is no improvement. The default value is `0`.

Value Returned

`nil` Returns `nil`.

ocnxlOutputExpr

```
ocnxlOutputExpr
     ( t_expr
     [ ?name t_outputName ]
     [ ?plot plot ]
     [ ?save save ]
     [ ?evalType t_evaltype]
     )
     => t / nil
```

Description

This command adds an output expression in the current test being specified. Specification od a test specification begins with `ocnxlBeginTest()`. See help on `ocnxlBeginTest()`. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

t_expr	The expression that you want to add.
t_outputName	The name of the expression. This is an optional argument.
plot	Whether to plot or not. This is an optional argument.
save	Whether to save or not. This is an optional argument.
t_evaltype	Whether to evaluate the expression for a design point or across all corners for a design points. This is an optional argument. Valid Values: `corners`, `point` Default Value: `point`

Value Returned

`t`	Returns `t` if the output expression is set.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlOutputExpr( "ymax(VT("/out"))" ?name "MAX" ?plot t ?save t )
# Adds "/out" in the outputs.
```

ocnxlOutputSignal

```
ocnxlOutputSignal( t_signal [ ?plot plot ] [ ?save save ] )
    => t / nil
```

Description

This command adds an output signal in the current test being specified. A test's specification begins with `ocnxlBeginTest()`. See help on `ocnxlBeginTest()`. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

t_signal	The name of the signal.
plot	Whether to plot or not. This is an optional argument.
save	Whether to save or not. This is an optional argument.

Value Returned

`t`	Returns `t` if the output signal is set.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlOutputSignal( "/out" ?plot t ?save t )
# Adds "/out" in the outputs.
```

ocnxlOutputTerminal

```
ocnxlOutputTerminal( t_term [ ?plot plot ] [ ?save save ] )
     => t / nil
```

Description

This command adds an output terminal in the current test being specified. A test's specification begins with `ocnxlBeginTest()`. See help on `ocnxlBeginTest()`. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

t_term	The name of the terminal.
plot	Whether to plot or not. This is an optional argument.
save	Whether to save or not. This is an optional argument.

Value Returned

`t`	Returns `t` if the output terminal is set.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlOutputTerminal( "/I8/inp" ?plot t ?save t )
# Adds "/I8/inp" in the outputs.
```

ocnxlOutputSummary

```
ocnxlOutputSummary(
     ?exprSummary g_exprSummary
     ?specSummary g_specSummary
     ?yieldSummary g_yieldSummary
     ?detailed g_detailed
     ?wave g_wave
     ?forRun runID)
     ?fileName t_filePath
     ?printHeader g_printHeader
     => t/nil
```

Description

Generates output summary. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

g_exprSummary	Default value for `exprSummary` is `t`. When *exprSummary* is `t`, then expressions summary is printed. If `detailed` is also set to `t`, then detailed expression summary is also printed. If you do not want to see the expressions summary, set this argument to `nil`.
g_specSummary	Default value for `specSummary` is `t`. When `specSummary` is `t`, then spec summary is printed. If `detailed` is also set to `t`, then detailed spec summary is also printed. If you do not want to see the spec summary, set this argument to `nil`.
g_yieldSummary	Default value for `yieldSummary` is `t`. When `yieldSummary` is `t`, then yield summary is printed. If `detailed` is also set to `t`, then detailed yield summary is also printed. If you do not want to see the yield summary, set this argument to `nil`.
g_detailed	Default value for `detailed` is `t`. When `detailed` is `t`, then details of expr/spec are printed. Otherwise only summary is printed.
g_wave	Default value of `wave` is `t`. When `wave` is `t`, then the value of expressions evaluating to a waveform is printed as "wave". If you do not want to see the expressions that are evaluated to waveforms in the output, set this argument to `nil`.

x_runID	Default value of forRun is `nil`. You can specify a runId for which you want to display the output summary. **Note:** Use this argument when the *?waitUntilDone* argument of the ocnxlRun command is set to `nil`.
t_filePath	Default value of fileName is `nil`. You can specify a file path and name to write the output summary to the specified file.
g_printHeader	Prints a default header before the output. Valid values: `t`: Always prints a header `WhenFileName`: Prints a header only when fileName is provided by using the fileName argument. This is the default value. `nil`: Does not print a header.

Value Returned

`t`	Returns `t` if the summary is generated.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlOutputSummary()
# This will print the details of expressions and specs for each sweep point and
each corner. It will also print the summary of expressions (minimum and maximum
values) and for specs (pass/fail, minimum and maximum values) for each sweep point
across all corners.

ocnxlOutputSummary(?exprSummary nil)
# This will print the details of specs for each sweep point and each corner. It
will also print the summary of specs (pass/fail, minimum and maximum values) for
each sweep point across all corners. This will not print any expression details/
summary.

ocnxlOutputSummary(?specSummary nil ?detailed nil)
# This will print only the summary of expressions (minimum and maximum values) for
each sweep point across all corners. This will not print any details for
expressions. This will also not print any spec details/summary.

ocnxlOutputSummary(?wave nil)
# This will print the details of expressions and specs for each sweep point and
each corner. It will also print the summary of expressions (minimum and maximum
values) and for specs (pass/fail, minimum and maximum values) for each sweep point
across all corners. This will skip printing the outputs that evaluate to waveforms.
```

```
ocnxlOutputSummary(?exprSummary t ?specSummary t ?detailed t ?wave t ?fileName
"myoutputfile")

#Writes all the summary to the file myoutputfile with default header as following
#
#Ocean XL Output Summary for run Ocean.<runNumber> on <DateTime>

(ocnxlOutputSummary ?forRun runid2 ?detailed nil)
#
#Ocean XL Output Summary for the run with runID as runid2
```

Related Functions

ocnxlRun

ocnxlTargetCellView

```
ocnxlTargetCellView(t_lib t_cell t_view ?mode)
     => t / nil
```

Description

Specifies target cellview where ADE XL data will be created. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

t_lib	Name of the library.
t_cell	Name of the cell.
t_view	Name of the view.
mode	(Optional) Specifies the mode of the target cellview. Valid values: `r` - Opens the target cellview in read mode `a` - Opens the target cellview in edit mode. Default value: `a`

Value Returned

`t`	Returns `t` if it is able to use the lib:cell:view as the target.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlTargetCellView("opamplib" "ampTest" "adexl" ?mode "r")
```

ocnxlYieldImprovementOptions

```
ocnxlYieldImprovementOptions( [ ?iymethod t_iymethod ]
     [ ?refPoint t_refPoint ] [ ?algorithm t_algorithm ] [ ?timeLimit t_timeLimit ]
     [ ?iterations t_iterations ] [ ?numPoints t_numPoints ]
     [ ?sigmaTarget t_sigmaTarget ] [?stopIfNoImprovement t_stopIfNoImprovement]
     [ ?runFullEvaluation t_runFullEvaluation ]
     [ ?optimizationMethod t_optimizationMethod ] [ ?effort t_effort ]
     [ ?iysamplingmethod t_iysamplingmethod ]
     [ ?iymontecarlodonominal t_iymontecarlodonominal ]
     [ ?iymontecarloseed t_iymontecarloseed ] [ ?iymcnumpoints t_iymcnumpoints ]
     [ ?iymontecarlostartingrun t_iymontecarlostartingrun ]
     [ ?WYCmethod t_WYCmethod ]
     [ ?dutSummary t_dutSummary ]
     [ ?ignoreFlag t_ignoreFlag ]
     )
     => t / nil
```

Description

Lets you specify options for improve yield runs. See help on `ocnxlRun()` for run modes. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

t_iymethod	The yield improvement method to be used. The default value is `all`. The possible values are `global`, `mismatch` and `all`.
t_refPoint	Specifies whether to use a schematic point or a reference point that you have created as a starting place for sizing. Possible values are `0` and `1`. The default value is `0`.
t_algorithm	The default value is `0`. The possible values are `0` and `1`.
t_timeLimit	Sets a time limit for the improve yield run. The default value is "". The `timeLimit` is in minutes.
t_iterations	Specifies the number of sizing/Monte Carlo iterations, The default value is 3.
t_numPoints	Specifies the maximum number of points processed per iteration. The default value is 3000.

t_sigmaTarget	Allows you to increase the mean of the goal distribution to target (of goal) value even after achieving 100% yield. ADE GXL allows you to achieve 4, 5, or even 6 sigma designs. The default value is `6`. The possible values are `4`, `5`, and `6`.
t_stopIfNoImprovement	Specifies if the yield improvement run must be stopped if there is no yield improvement. The default value is `0`. The possible values are `0 and 1`.
t_runFullEvaluation	Sets the program to run full optimization. Default for `runFullEvaluation` is `0`. Possible values are `0` and `1`.
t_optimizationMethod	Sets the optimization method. Default for `optimizationMethod` is `global`. Possible values are `global` and `local`.
t_effort	Specifies the effort level if you are using local optimization. Default for effort is `fine`. Possible values are `fine` and `coarse`.
t_iysamplingmethod	Sets the default statistical sampling method for improve yield runs. The default value is `random`. Possible values are `random` and `lhs`.
t_iymontecarlodonominal	Specifies whether to run a simulation at the reference point prior to beginning the improve yield process. Possible values are `0` and `1`. If set to 1, Spectre will run a simulation at the reference point, and, if this fails, then the sampling process is not initiated and the simulation stops. The default value is 1.
t_iymontecarloseed	Specifies a different seed for Monte Carlo runs. Default for `monteCarloSeed` is `12345`.
t_iymcnumpoints	Sets the number of Monte Carlo points you want to simulate. The default value is `nil`.
t_iymontecarlostartingrun	Specifies the run that Monte Carlo begins with. The default value is 1.

t_dutSummary

Specifies a list of design under test (DUT) instances for improve yield runs. In this list, you can specify the instances and devices to which mismatch variations must be applied. The format to specify the list is as given below:

```
<testname%instances%Libname/Cellname/
Viewname%Master#testname%instances%modelname%Subcircuit#
testname%instances%Schematic%Schematic>
```

where two DUT instances in the list are separated by a `#(hash).`

```
For example:
"opamp090:full_diff_opamp_AC:2:1%/
I21%acOpenDiff%Subcircuit#opamp090:full_diff_opamp_AC:2:
1%/I0/I1%opamp090/ampn/
schematic%Master#opamp090:full_diff_opamp_AC:2:1%/I0/
M5A, /I0/M3A%Schematic%Schematic"
```

Default for `dutSummary` is "".

t_ignoreFlag

Optional argument to specify if the user wants to apply mismatch variations to instances specified with `dutSummary`.

Default for `ignoreFlag` is `0`. Set it to `1` if you do not want to apply mismatch variations to instances.

Value Returned

`t` Returns `t` if the options are specified.

`nil` Returns `nil` otherwise.

Example

```
ocnxlYieldImprovementOptions(?iymethod "mismatch" )
# Sets mismatch as the method for yield improvement run.
```

ocnxlEnableCornerForTest

```
ocnxlEnableCornerForTest(t_cornerName t_testName)
     => t / nil
```

Description

Enables a corner for a test. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

`t_cornerName`	Name of the corner.
`t_testName`	Name of the test.

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlEnableCornerForTest("C0" "test_one") => t
```

ocnxlEnableSweepParam

```
ocnxlEnableSweepParam(t_paramPath)
     => t / nil
```

Description

Enables a sweep parameter. A disabled sweep parameter is not run when the `ocnxlRun()` command is run. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

`t_paramPath`	Name of the sweep parameter.

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlEnableSweepParam("solutions/ampTest/schematic/R1/r") => t
```

ocnxlEnableSweepVar

```
ocnxlEnableSweepVar(t_varName)
     => t / nil
```

Description

Enables a sweep variable. A disabled sweep variable is not run when the `ocnxlRun()` command is run. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

`t_varName`	Name of the sweep variable.

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlEnableSweepVar("CAP") => t
```

ocnxlEnableTest

```
ocnxlEnableTest(t_testName)
     => t / nil
```

Description

Enables a test. A disabled test will not be run when the `ocnxlRun()` command is run. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

`t_testName`	Name of the test.

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlEnableTest("test_two") => t
```

ocnxlGetBestPointParams

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

Description

Returns a list of best design points. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

None

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlGetBestPointParams()
```

ocnxlGetCorners

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

Description

Returns a list of corners names. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

None

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlGetCorners()
```

ocnxlGetCurrentHistory

```
ocnxlGetCurrentHistory()
     => historyName |nil
```

Description

Returns the current history (checkpoint) name. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

None

Value Returned

`historyName`	Returns the name of the current history.
`nil`	Returns `nil` in case of an error.

Example

```
ocnxlLoadSetupState( "ac_state1" 'retain ?tests t ?vars t ?parameters t
?currentMode t ?runOptions t ?specs t ?corners t ?extensions t
    ?modelGroups nil ?relxanalysis nil )

runId = ocnxlRun( ?mode 'sweepsAndCorners ?nominalCornerEnabled t
?allCornersEnabled t ?allSweepsEnabled t ?waitUntilDone nil)

; The following function returns the handle to the results database for the current
history.
rdb=axlReadHistoryResDB(ocnxlGetCurrentHistory() ?session ocnxlGetSession())

; The following function returns the point object for design point 1.
pt = rdb->point(1)
; The following code prints corner name, test name, output name and its value
; for each output of type expression
foreach(out pt->outputs(?type 'expr ?sortBy 'corner)
printf("corner=%s, test=%s, output=%s, value=%L\n" out->cornerName out->testName
out->name out->value))
```

When the above script is run, the results are displayed as shown below:

```
corner=C4_0, test=AC, output=gainBwProd(VF("/OUT")), value=1.068285e+09
corner=C4_0, test=AC, output=Current, value=0.0007904204
corner=C4_0, test=AC, output=Gain, value=49.76433
```

```
corner=C4_0, test=AC, output=UGF, value=5.639488e+08
corner=C4_0, test=TRAN, output=SettlingTime, value=5.911562e-09
corner=C4_0, test=TRAN, output=Swing, value=1.26184
corner=C4_1, test=AC, output=gainBwProd(VF("/OUT")), value=5.452747e+08
corner=C4_1, test=AC, output=Current, value=0.0004168163
corner=C4_1, test=AC, output=Gain, value=46.60983
corner=C4_1, test=AC, output=UGF, value=2.736042e+08
corner=C4_1, test=TRAN, output=SettlingTime, value=7.762304e-09
corner=C4_1, test=TRAN, output=Swing, value=1.05484
corner=nominal, test=AC, output=gainBwProd(VF("/OUT")), value=1.068285e+09
corner=nominal, test=AC, output=Current, value=0.0007904204
corner=nominal, test=AC, output=Gain, value=49.76433
corner=nominal, test=AC, output=UGF, value=5.639488e+08
corner=nominal, test=TRAN, output=SettlingTime, value=5.911562e-09
corner=nominal, test=TRAN, output=Swing, value=1.26184
```

ocnxlGetCurrentHistoryId

```
ocnxlGetCurrentHistoryId()
    => historyID | nil
```

Description

Returns the ID of the current history (checkpoint). This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

None

Value Returned

`t`	Returns the ID of the current history.
`nil`	Returns `nil` in case of an error.

Example

```
ocnxlGetCurrentHistoryId()
```

ocnxlGetHistory

```
ocnxlGetHistory(
     x_runID)
     => x_setupdbHandle / nil
```

Description

Returns the handle to the history setup database associated with a particular run. You can use this handle to work with the history results.

This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

`x_runID`	ID of the run for which the handle to the history setup database is to be returned.

Value Returned

`x_setupdbHandle`	Returns handle to the history setup database.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlLoadSetupState( "ac_state1" 'retain ?tests t ?vars t ?parameters t
?currentMode t
    ?runOptions t ?specs t ?corners t ?extensions t
    ?modelGroups nil ?relxanalysis nil )

runid2 = ocnxlRun(?waitUntilDone nil)
ocnxlLoadSetupState( "tran_state2" 'retain ?tests t ?vars t ?parameters t
?currentMode t
    ?runOptions t ?specs t ?corners t ?extensions t
    ?modelGroups nil ?relxanalysis nil )

runId = ocnxlRun( ?mode 'sweepsAndCorners ?nominalCornerEnabled t
?allCornersEnabled t ?allSweepsEnabled t ?waitUntilDone nil)

histId = ocnxlGetHistory(runId)

ocnxlWaitUntilDone('All)

psfDir = axlGetPointPsfDir(histId "mdltest:testinv:1" ?cornerName ""
```

```
?designPointId 1)
```

Related Functions

ocnxlRun, ocnxlSetRunDistributeOptions

ocnxlGetOverwriteHistory

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

Description

Returns the status of overwrite history. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

None

Value Returned

`t`	Returns `t` if overwrite history is enabled.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlGetOverwriteHistory()
t
```

ocnxlGetOverwriteHistoryName

```
ocnxlGetOverwriteHistoryName()
     => t_historyName / nil
```

Description

Returns name of the history to be overwritten. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

None

Value Returned

`t_historyName`	Returns name of the history to be overwritten.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlGetOverwriteHistoryName()
Interactive.4
```

ocnxlGetRunDistributeOptions

```
ocnxlGetRunDistributeOptions()
     => l_runOptions | nil
```

Description

Returns the run options set for the current setup database. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

None

Value Returned

l_runOptions	List of run options specified for the current setup database. This list contains the following three values: RunIn: Describes how multiple simulations need to run. Valid values are `Parallel` or `Serial`. DivideJobs: Describes how the ICRPs can be divided among the simulation runs. Valid values are `Specify` or `Equally`. JobLimit: Describes the maximum number of jobs that can run when Divide Jobs is set to `Specify`.
nil	Unsuccessful operation

Example

```
runOpt = ocnxlGetRunDistributeOptions()
runOpt~>??
(JobLimit 2 DivideJobs Specify RunIn
    Parallel
)
```

Related Functions

ocnxlSetRunDistributeOptions

ocnxlGetSession

```
ocnxlGetSession()
     => t_sessionName / nil
```

Description

Returns the session name. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

None

Value Returned

`t_sessionName`	Returns the name of the session.
`nil`	Returns `nil` and prints an error message if there is no OCEAN XL session.

Example

The following example shows that this function returns the default session name assigned by OCEAN.

```
ocnSetXLMode()
t
ocnxlTargetCellView("myLib" "ampTest" "adexl")
t
ocnxlGetSession()
"ocnXLSession_Apr_18_10_11_38_2013"
```

ocnxlGetSpecs

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

Description

Returns a list of parameter specification names. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

None

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlGetSpecs()
```

ocnxlGetTests

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

Description

Returns a list of test names. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

None

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlGetTests()
```

ocnxlRemoveSpec

```
ocnxlRemoveSpec(t_specName)
     => t / nil
```

Description

Removes the specified parameter specification. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

`t_specName`	Name of the spec.

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlRemoveSpec("MAX")
```

ocnxlRenameCurrentHistory

```
ocnxlRenameCurrentHistory(t_newNameForHistory)
     => t / nil
```

Description

Renames the current history (checkpoint). This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

`t_newNameForHistory`	New name for the current history.

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlRenameCurrentHistory("myHistory")
```

ocnxlRun

```
ocnxlRun( [ ?mode s_mode ]
     [ ?nominalCornerEnabled g_nominalCornerEnabled ]
     [ ?allCornersEnabled g_allCornersEnabled ]
     [ ?allSweepsEnabled g_allSweepsEnabled ]
     [ ?verboseMode g_verboseMode ]
     [ ?waitUntilDone g_waitUntilDone ]
     )
     => t/nil/runID
```

Description

Specifies the run mode for simulation and whether to run the nominal corner, corners and sweeps during simulation. Also specifies whether to report completion of points during simulation. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

s_mode Lets you run simulations in one of following modes:

- ’sweepsAndCorners
- ’localOptimization
- ’globalOptimization
- ’monteCarlo
- ’yieldImprovement
- ’sampling
- ’sensitivity
- 'feasibilityAnalysis
- 'Size Over Corners

g_nominalCornerEnabled Accepts boolean values `t` or `nil`. The default value is `t`. If set to `nil`, ADE XL excludes nominal corners from the simulation run.

g_allCornersEnabled Accepts boolean values `t` or `nil`. The default value is `t`. If set to `nil`, ADE XL excludes all corners from the simulation run.

g_allSweepsEnabled	Accepts boolean values `t` or `nil`. The default value is `t`. If set to `nil`, ADE XL excludes all sweeps from the simulation run.
g_waitUntilDone	Specifies if OCEAN should wait for this run to complete before executing the next command in the script. **Valid values:** ■ `t`: Specifies that OCEAN should wait for the completion of this run. This is the default value. ■ `nil`: Specifies that the you intend to run multiple OCEAN runs in parallel. In this case, ocean assigns a run id to each ocean run. **Note:** Set this argument to `nil` to run multiple OCEAN runs in parallel.
g_verboseMode	Accepts boolean values `t` or `nil`. The default value is `t`. If set to `nil`, ADE XL does not report the progress in the simulation of points in the simulation run. **Note:** It is recommended that you specify the value `nil` if you have set up a large number of points.

Value Returned

`t`	Returns t when the run is successful.
`runID`	When the OCEAN runs are run in parallel, that is, when *?waitUntilDone* is set to `nil`, returns the run ID on success.
`nil`	Returns nil when the run is unsuccessful.

Examples

Example 1:

The following command runs an already loaded setup and also specifies that no corner should be run.

```
ocnxlRun(?allCornersEnabled nil)
```

No corner will be run but rest of the setup will be run.

Example 2:

This example runs two setups in parallel.

```
ocnxlJobSetup( '( "blockemail" "1" "configuretimeout" "300" "distributionmethod"
"Local" "lingertimeout" "300" "maxjobs" "8" "name" "ADE XL Default"
"preemptivestart" "1" "reconfigureimmediately" "1" "runtimeout" "-1"
"showerrorwhenretrying" "1" "showoutputlogerror" "0" "startmaxjobsimmed" "1"
"starttimeout" "300" ) )
ocnxlLoadSetupState( "C1" 'retain ?tests t ?vars t ?parameters t ?currentMode t
    ?runOptions t ?specs t ?corners t ?extensions t
    ?modelGroups nil ?relxanalysis nil )

runid1 = ocnxlRun(?waitUntilDone nil)
ocnxlLoadSetupState( "C4" 'retain ?tests t ?vars t ?parameters t ?currentMode t
    ?runOptions t ?specs t ?corners t ?extensions t
    ?modelGroups nil ?relxanalysis nil )

runid2 = ocnxlRun(?waitUntilDone nil)
(ocnxlWaitUntilDone 'All)
(ocnxlOutputSummary ?forRun runid2 ?detailed nil)
; The previous command displays run summary for the second run
```

Related Functions

ocnxlWaitUntilDone, ocnxlLoadSetupState, ocnxlJobSetup, ocnxlGetHistory, ocnxlSetRunDistributeOptions

ocnxlHistoryPrefix

```
ocnxlHistoryPrefix(t_prefixName)
     => t / nil
```

Description

Sets the prefix used in the names of history items created by OCEAN XL runs. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

t_prefixName	The prefix to be used in the names of history items.

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlHistoryPrefix("check")
```

Creates history items with names like `check.0`, `check.1`, and so on.

ocnxlSetReferenceHistory

```
ocnxlSetReferenceHistory(
     t_historyName
     [ ?reuseNetlist t_reuseNetlist ]
     [ ?useReferenceResults t_useReferenceResults ]
     )
     => t_referenceHistoryName | nil
```

Description

Sets a reference history for incremental simulation runs in OCEAN. This command works only in XL mode. See help on `ocnSetXLMode()`.

Argument

t_historyName	Name of reference history that you want to use. Use the ocnxlGetCurrentHistory() function to use the current history or give the name of the reference history that you want to use.
t_reuseNetlist	(Optional) Specifies whether to reuse the netlist in the subsequent runs. If the design has not changes, you can reuse the netlist. Possible values: `t`: Creates an incremental netlist for the new design points. However, for same design points, netlist from the reference history is reused. `nil`: Always creates a new netlist for the design. Default value: `nil`

t_useReferenceResults (Optional) Specifies whether to reuse the results from the reference history for the incremental simulation run.

Possible values:

`new`: Creates a new resultset for the incremental simulation run. Does not use the results from the reference history.

`copy`: Copies the simulation results of the reference history to the new history item that is created during the incremental simulation run. With this option, OCEAN XL displays the results for only the updated values.

`move`: Moves the simulation results of the reference history to the new history item that is created during the incremental simulation run.

Default value: `copy`

Value Returned

`t_referenceHistoryName`	Returns `t` if the name of the reference history is set.
`nil`	Returns `nil` if unsuccessful.

Example

In the following example, during the first OCEAN run, the variable `var1` is sweeped for values `1` and `2`. For the next run, the current history is set as the reference history. The default options specify that the netlist will not be reused, but the reference results will be copied to the incremental run. As a result, in the subsequent run, netlist will be created for the entire design, but the results will be generated only for the new design points with the value of `var1` set to 3, 4, and 5. Results for design points with ABC set to `1` or `2` will be copied to the new history.

```
ocnxlSetXLMode()
...
...
ocnxlSweepVar("var1" "1 2")
...
...
ocnxlRun(...)
...
```

```
...
ocnxlSetReferenceHistory(ocnxlGetCurrentHistory())
...
ocnxlSweepVar("var1" "1 2 3 4 5")
...
ocnxlRun() <--- This will only run three (3 4 5) additional points for netlisting

...
...
ocnxlEndXLMode()
```

ocnxlGetReferenceHistory

```
ocnxlGetReferenceHistory(
     )
     => t_referenceHistoryName | nil
```

Description

Gets the name of the reference history currently set in the OCAEN XL. This command works only in XL mode. See help on `ocnSetXLMode()`.

Argument

None

Value Returned

`t_referenceHistoryName`	Name of the reference history currently set.
`nil`	If no reference history is set.

Example

```
ocnxlSetXLMode()
...
...
ocnxlSetReferenceHistory(ocnxlGetCurrentHistory())
t
ocnxlGetReferenceHistory()
"Interactive.1"
```

ocnxlExportOutputView

```
ocnxlExportOutputView(
     t_fileName
     t_viewType
     )
     => t | t_error
```

Description

Exports the results view to the specified .csv or .html file.

Argument

t_fileName	Path and name of file to which the results need to be exported. Append .htm or .html to the filename to write in HTML format and .csv to write in CSV format. If no file extension is specified, a .csv file is created by default. The file is saved in the current working directory.

t_viewType	Name of the view type that needs to be exported.
	Possible values:
	`""` or `Current`: writes the currently visible view
	`Detail`
	`Detail-Transpose`
	`Optimization`
	`Summary`
	`Yield`
	Default value: `""`

Value Returned

`t`	Successful export of output view.
`t_error`	If unsuccessful, returns an error message.

Example

```
ocnxlExportOutputView( "./abc.csv" "Yield")

ocnxlExportOutputView( "./abc.html" "Detail-Transpose")

ocnxlExportOutputView("./abcd.html" "Yield" )
```

ocnxlSetOverwriteHistory

```
ocnxlSetOverwriteHistory(g_historyStatus)
    => t | nil
```

Description

Sets the status of overwrite history. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

g_historyStatus	Enables or disables overwrite history. Default value: `t`

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlSetOverwriteHistory(t)t
```

ocnxlSetOverwriteHistoryName

```
ocnxlSetOverwriteHistoryName(t_historyName)
     => t | nil
```

Description

Sets name of the history to be overwritten. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

`t_historyName`	Name of the history to be overwritten.

Value Returned

`t_historyName`	Returns t is successful.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlSetOverwriteHistoryName("Interactive.4")
```

ocnxlSetRunDistributeOptions

```
ocnxlSetRunDistributeOptions(
     ?RunIn t_runIn
     ?DivideJobs t_divideJobs
     ?JobLimit n_jobLimit
     )
     => t | nil
```

Description

Sets the specified run option settings for the current setup database. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

t_runIn	Describes how multiple simulations need to run. Valid values: `Parallel`, `Serial`.
t_divideJobs	Specifies how the ICRPs are divided among the simulation runs. Valid values: `Specify`, `Equally`.
n_jobLimit	Specifies the maximum number of jobs that can run when ?DivideJobs is set to `Specify`. **Note:** This value is not considered when ?DivideJobs is set to `Equally`.

Value Returned

`t`	Returns `t` when the options are successfully set.
`nil`	Returns `nil` otherwise.

Example

The following example sets the run options to run ICRPs in parallel with a maximum of three jobs per run:

```
ocnxlSetRunDistributeOptions( ?RunIn 'Parallel ?DivideJobs 'Specify ?JobLimit 3)
```

`t`

Related Functions

ocnxlGetRunDistributeOptions

ocnxlLoadSetupState

```
ocnxlLoadSetupState(
     t_state t_state
     t_mode t_mode
     [ ?tests t_tests ]
     [ ?vars t_vars ]
     [ ?parameters t_parameters ]
     [ ?currentMode t_currentMode ]
     [ ?runOptions t_runOptions ]
     [ ?specs t_specs ]
     [ ?corners t_corners ]
     [ ?modelGroups t_modelGroups ]
     [ ?extensions t_extensions]
     [ ?relxanalysis t_relxanalysis ]
     )
     => t / nil
```

Description

Restores the settings in the specified setup state to the active setup. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

t_stateName	The name of the setup state to be restored.
t_mode	Specifies the mode for restoring the settings in the setup state to the active setup. Valid values: `'retain`, `'merge`, `'overwrite`
t_tests	Specifies whether the tests in the setup state should be restored to the active setup. Valid values: `t`, `nil` Default Value: `t`
t_vars	Specifies whether the global variables in the setup state should be restored to the active setup. Valid values: `t`, `nil` Default Value: `t`

t_parameters	Specifies whether the parameters in the setup state should be restored to the active setup. Valid values: `t`, `nil` Default Value: `t`
t_currentMode	Specifies whether the run mode in the setup state should be restored to the active setup Valid values: `t`, `nil` Default Value: `t`
t_runOptions	Specifies whether the run options in the setup state should be restored to the active setup Valid values: `t`, `nil` Default Value: `t`
t_specs	Specifies whether the parameter specifications in the setup state should be restored to the active setup Valid values: `t`, `nil` Default Value: `t`
t_corners	Specifies whether the corners in the setup state should be restored to the active setup Valid values: `t`, `nil` Default Value: `t`
t_modelGroups	Specifies whether the model groups in the setup state should be restored to the active setup. Valid values: `t`, `nil` Default Value: `t`
t_extensions	Specifies whether the extensions in the setup state should be restored to the active setup. Valid values: `t`, `nil` Default Value: `t`

t_relxanalysis	Specifies whether the details of reliability analysis should be restored to the active setup. Valid values: `t`, `nil` Default Value: `t`

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlLoadSetupState("optimize")
```

Loads the setup state named `optimize`.

ocnxlStartingPoint

```
ocnxlStartingPoint(l_startingPointDetails)
     => t / nil
```

Description

Lets you specify a reference point—a starting place for sizing—for Improve Yield, Global Optimization, Feasibility Analysis or Monte Carlo runs. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

l_startingPointDetails	A list of elements where each element is: (*t_type t_varName t_value*) Where: ■ *t_type* can be a `variable` or `parameter`. ■ *t_varName* is the name of the variable or parameter ■ *t_value* is the value of the variable or parameter.

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlStartingPoint(’(("variable" "CAP" "2p")
     (“parameter” "ether_adcflash/adc_cascode_opamp/schematic/M2/fw" "16.3u")))
```

ocnxlOutputAreaGoal

```
ocnxlOutputAreaGoal( t_expr
     [ ?name t_outputName ]
     [ ?plot plot ]
     [ ?save save ]
     )
     => t / nil
```

Description

Adds an area goal output expression in the current test being specified. A test's specification begins with `ocnxlBeginTest()`. See help on `ocnxlBeginTest()`. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

`t_expr`	The expression that you want to add.
`t_outputName`	The name of the expression. This is an optional argument.
`plot`	Whether to plot or not. This is an optional argument.
`save`	Whether to save or not. This is an optional argument.

Value Returned

`t`	Returns `t` if the output expression is set.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlOutputAreaGoal( "(('I8/R4' ('res' 'w*l' 'default' 'enabled'))...)" ?name
"MAX" ?plot t ?save t ) => t
```

Adds an area goal output expression named `MAX` for the current test.

ocnxlConjugateGradientOptions

```
ocnxlConjugateGradientOptions (
     [?runFullEvaluation t_runFullEvaluation]
     [?meetAllGoals t_meetAllGoals]
     [?timeLimit t_timeLimit]
     [?numPoints t_numPoints]
     )
     => t | nil
```

Description

Sets options for conjugate gradient runs. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

`t_runFullEvaluation`	Sets to run full evaluation. Possible Values: “1” and “0”. Default Value: “1”
`t_meetAllGoals`	Sets to run only until all goals are met. Possible Values: “1” and “0”. Default Value: “1”
`t_timeLimit`	Sets a time limit (in seconds) for the run.
`t_numPoints`	Sets the limit in the number of points to be run.

Value Returned

`t`	Returns `t` if options are specified for conjugate gradient run.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlConjugateGradientOptions(?numPoints "3000")
Sets to run for 3000 points.
```

ocnxlMTSEnable

```
ocnxlMTSEnable( g_enable)
     => t / nil
```

Description

Enables or disables multi-technology simulation (MTS) mode for the current test. The current test's specification begins with `ocnxlBeginTest()`. See help on `ocnxlBeginTest()`. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

g_enable	Enables or disables MTS mode for the current test. Valid values: `t`, `nil` Default Value: `nil`

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlMTSEnable(t)
```

Enables MTS mode for the current test.

ocnxlMTSBlock

```
ocnxlMTSBlock( s_blockName
     [?isMtsBlock  g_isMtsBlock]
     [?includeFile l_includeFile]
     [?modelFiles  l_modelFiles]
     )
     => t/nil
```

Description

Enables a block for multi-technology simulation (MTS) and specifies the include files and model files associated with the block. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

s_blockName	Specifies the name of the block that needs to be enabled for multi-technology simulation. Valid values: a string
g_isMtsBlock	Specifies whether the block is enabled or disabled for multi-technology simulation. Valid values: `t`, `nil` Default Value: `nil`
l_includeFile	Specifies the include files associated with the block. Valid values: a list of strings or `nil` Default value: `nil`
l_modelFiles	Specifies the model files associated with the block. Valid values: a list of strings or `nil` Default value: `nil`

Value Returned

`t`	Returns `t` if successful.

`nil`	Returns `nil` otherwise.

Example

```
ocnxlMTSBlock('digLib\ inv_usim
?isMtsBlock t
?modelFiles '(("Models/myModels.scs" "ss")
("Models/spectre_cl013lv.scs" "aa"))
```

Enables the `inv_usim` cell in the `digLib` library for multi-technology simulation and specifies the model files (and the sections of the model files) associated with the block.

ocnxlProjectDir

```
ocnxlProjectDir(t_projectDir)
     => t / nil
```

Description

Sets the project directory to the specified location. All simulation data goes into this location by default, if the simulation results or results directories are not set. By default, the project directory is set as `$HOME/simulation`.

This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

`t_projectDir`	Sets the location of the project directory.

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlProjectDir("/tmp/simulation")
```

Related Commands

ocnxlResultsLocation, ocnxlSimResultsLocation

ocnxlSimResultsLocation

```
ocnxlSimResultsLocation(t_simResultsDir)
     => t / nil
```

Description

Sets the simulation results directory to the specified location. All simulation data goes into this location. If the simulation results directory is not set using this function, the simulation results are saved at any one of the following locations:

- In the /`libraryName/cellName/<target-view>/results/data/<history_item>` directory at the location set by ocnxlResultsLocation, if set.
- Otherwise, in the `libraryName/cellName/<target-view>/results/data/<history_item>` directory at the location set by ocnxlProjectDir, if set.
- Otherwise, in the `$HOME/simulation/libraryName/cellName/<target-view>/results/data` directory.

This command works only in XL mode. See help on ocnSetXLMode.

Arguments

`t_simResultsDir`	Sets the location of the simulation results directory.

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlSimResultsLocation("/home/ocnuser")
```

ocnxlDisableCorner

```
ocnxlDisableCorner(t_cornerName)
     => t/nil
```

Description

Lets you disable a corner. A disabled corner will not be run when the ocnxlRun() command is run. This command works only in XL mode. See help on `ocnSetXLMode()`

Arguments

`t_cornerName`	The name of the corner to be disabled.

Value Returned

`t`	Returns `t` if the corner is disabled.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlDisableCorner("C0")  => t
```

ocnxlEnableCorner

```
ocnxlEnableCorner(t_cornerName)
     => t/nil
```

Description

Lets you enable a corner. An enabled corner will be run when ocnxlRun() command is run. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

`t_cornerName`	The name of the corner to be enabled.

Value Returned

`t`	Returns `t` if the corner is disabled.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlEnableCorner("C0") => t
```

ocnxlSaveSetupAs

```
ocnxlSaveSetupAs(t_lib t_cell t_view)
    => t/nil
```

Description

Saves the current setup to a different adexl view. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

`t_lib`	The name of the library in which the new adexl view is to be saved.
`t_cell`	The name of the cell in which the new adexl view is to be saved.
`t_view`	The name of the new adexl view.

Value Returned

`t`	Returns `t` if the save is successful.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlSaveSetupAs("solution" "ampTest" "newView")
```

ocnxlParametricSet

`ocnxlParametricSet( l_paramList )`

Description

Creates a parametric set by using the given list of parameters.

Arguments

l_paramList	List of parameter names to be included in the parametric set.

Example

The following example creates a parametric set with two parameters, *vin_ac* and *vdd*:

```
ocnxlParametricSet('("vin_ac" "vdd"))
```

ocnxlSetAllParametersDisabled

```
ocnxlSetAllParametersDisabled(g_disabled)
     => t/nil
```

Description

Enables or disables all parameters. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

`g_disabled`	Specify `t` to disable all parameters, and `nil` to enable all parameters.

Value Returned

`t`	Returns `t` if all parameters are enabled or disabled.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlSetAllParametersDisabled(t) => t
```

ocnxlSetAllVariablePSetsDisabled

```
ocnxlSetAllVariablePSetsDisabled(g_disabled)
     => t/nil
```

Description

Enables or disables all variable parametric sets. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

`g_disabled`	Specify `t` to disable all variable parameteric sets. Specify `nil` to enable all variable parameteric sets.

Example

```
ocnxlSetAllVariablePSetsDisabled(t)
```

ocnxlSetAllParameterPSetsDisabled

```
ocnxlSetAllParameterPSetsDisabled(g_disabled)
     => t/nil
```

Description

Enables or disables all parameter parametric sets. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

`g_disabled`	Specify `t` to disable all parameter parameteric sets. Specify `nil` to enable all parameter parameteric sets.

Example

```
ocnxlSetAllParameterPSetsDisabled(true)
```

ocnxlSetAllVarsDisabled

```
ocnxlSetAllVarsDisabled(g_disabled)
     => t/nil
```

Description

Lets you enable or disable all global variables. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

`g_disabled`	Specify `t` to disable all variables, and `nil` to enable all variables.

Value Returned

`t`	Returns `t` if all variables are enabled or disabled.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlSetAllVarsDisabled(t) => t
```

ocnxlPreRunScript

```
ocnxlPreRunScript(t_fileName)
     => t / nil
```

Description

Specifies the pre-run script file containing the OCEAN commands that need to be run before the simulation starts. This function must be used within a test setup block (starting the ocnxlBeginTest and ending with the ocnxlEndTest command) in your OCEAN script file. This command works only in the XL mode. See help on `ocnSetXLMode()`.

For more information, see Executing Pre-run Scripts before Simulation Runs.

Arguments

`t_fileName`	Path to the pre-run script file.

Value Returned

`t`	Returns `t` if file exists.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlBeginTest("myTest")
...
ocnxlPreRunScriptEnabled(t)
ocnxlPreRunScript("/net/scripts/myPreRunScript")
...
ocnxlEndTest()
```

Related Functions

ocnxlSetPreRunScriptEnabled

ocnxlSetPreRunScriptEnabled

```
ocnxlSetPreRunScriptEnabled(g_enabled)
     => t | nil
```

Description

Specifies if running the pre-run scripts through OCEAN scripts should be enabled.

This command works only in the XL mode. See help on `ocnSetXLMode()`.

For more information, see Executing Pre-run Scripts before Simulation Runs.

Arguments

`g_enabled`	Specifies if running pre-run scripts should be enabled. Default value: `t` Possible values: `t`, `nil`

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlBeginTest("myTest")
...
ocnxlSetPreRunScriptEnabled(t)
ocnxlPreRunScript("/net/scripts/myPreRunScript")
...
ocnxlEndTest()
```

ocnxlLoadCurrentEnvironment

```
ocnxlLoadCurrentEnvironment(g_noAnalysis)
     => t / nil
```

Description

Reads the test's simulation setup. It also sets the results and netlist directory for the pre-calibration simulations (simulations that are run for calibrating the simulation setup) based on the results directory for the current point. For example, if the results directory of point `4` is `$AXL_PROJECT_DIR/myLib/myCell/myView/results/data/Interactive.1/4/myTest`, the netlist directory for the pre-calibration simulation run will be `$AXL_PROJECT_DIR/myLib/myCell/myView/results/data/Interactive.1/4/myTest/preSim/netlist` and the results directory will be `$AXL_PROJECT_DIR/myLib/myCell/myView/results/data/Interactive.1/4/myTest/preSim/psf`. You can specify a different results directory for the pre-calibration simulation run by using the `resultsDir` OCEAN command. This command must be used only in a pre-run script.

For more information, see Executing Pre-run Scripts before Simulation Runs.

Arguments

`g_noAnalysis`	Specifies whether the test's simulation setup must be read before calibrating the simulation setup for a point. Default value: `nil` Possible values: `t` and `nil`.

Value Returned

`t`	Returns `t` if test's simulation setup is successfully read
`nil`	Returns `nil` otherwise.

Example

```
ocnxlLoadCurrentEnvironment(t)

analysis('tran ?stop 10u)
```

ocnxlSetCalibration

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

Description

Sets up a single iteration Monte Carlo calibration run by inheriting statistical parameter information from the main Monte Carlo simulation run. The starting iteration number for the calibration run is set to the current iteration number of the main Monte Carlo simulation run. This command must be used only in a pre-run script.

For more information, see Executing Pre-run Scripts before Simulation Runs.

Arguments

None

Value Returned

`t`	Returns `t` if successful
`nil`	Returns `nil` otherwise

Example

```
ocnxlSetCalibration()
t
```

ocnxlSetMCdut

```
ocnxlSetMCdut(t_instName)
     => t / nil
```

Description

Sets a design instance to be used in a pre-run script for Monte Carlo calibration. If set, the specified subcircuit instance has process and mismatch variations applied to it and the unspecified instances only have process variations. All subcircuits instantiated under the specified instance also have process and mismatch enabled. By default, mismatch variations are applied to all the subcircuit instances in the design and process variations are applied globally. This allows the testbench to change and not affect the variations seen by the actual design.

Note: This function is to be used in a pre-run script and only applies to Monte Carlo analysis. Execute this function after ocnxlSetCalibration.

For more information, see Executing Pre-run Scripts before Simulation Runs.

Arguments

t_instName	Specifies the name of design instance.

Value Returned

`t`	Returns `t` if the design instance is set successfully
`nil`	Returns `nil` otherwise

Example

```
;Set the MC iteration number, etc. to match the main simulation
ocnxlSetCalibration()

;Set the DUT instance of the pre run design
ocnxlSetMCdut("I0")
```

ocnxlRunCalibration

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

Description

Starts the simulation required to calibrate the simulation setup. This command must be used only in a pre-run script.

For more information, see Executing Pre-run Scripts before Simulation Runs.

Arguments

None

Value Returned

`t`	Returns `t` if the simulation run is successful
`nil`	Returns `nil` otherwise

Example

```
ocnxlRunCalibration()
t
```

ocnxlAddOrUpdateOutput

```
ocnxlAddOrUpdateOutput(t_outName t_outVal)
     => t / nil
```

Description

Adds the specified scalar output to the simulation setup so that the results for the output can be viewed on the Results tab in ADE XL. If the specified output name already exists, only its value is updated with the specified value. This command must be used only in a pre-run script.

For more information, see Executing Pre-run Scripts before Simulation Runs.

Arguments

`t_outName`	Output name
`t_outVal`	Output value

Value Returned

`t`	Returns `t` if output is successfully added or updated
`nil`	Returns `nil` otherwise

Example

```
ocnxlAddOrUpdateOutput("Calibrated_ParamName" CalResult)
```

ocnxlUpdatePointVariable

```
ocnxlUpdatePointVariable(t_paramName t_paramValue)
     => t / nil
```

Description

Updates the value of a parameter or variable in the simulation setup. This command must be used only in a pre-run script.

For more information, see Executing Pre-run Scripts before Simulation Runs.

Arguments

`t_paramName`	Parameter name
`t_paramValue`	Parameter value

Value Returned

`t`	Returns `t` if parameter value is successfully updated
`nil`	Returns `nil` otherwise

Example

```
ocnxlUpdatePointVariable("ParamName" "6.7")
```

ocnxlGetJobId

```
ocnxlGetJobId()
     => x_jobID / nil
```

Description

Returns the ID of the current simulation job. This command must be used only in a pre-run script.

For more information, see Executing Pre-run Scripts before Simulation Runs.

Arguments

None

Value Returned

`x_jobId`	Returns the job ID of the current simulation job
`nil`	Returns `nil` otherwise

Example

```
id=ocnxlGetJobId()
```

ocnxlGetPointId

```
ocnxlGetPointId()
     => x_pointID / nil
```

Description

Returns the ID of the current simulation point. This command must be used only in a pre-run script.

For more information, see Executing Pre-run Scripts before Simulation Runs.

Arguments

None

Value Returned

`x_pointId`	Returns the ID of current simulation point
`nil`	Returns `nil` otherwise

Example

```
id=ocnxlGetPointId()
```

ocnxlMCIterNum

```
ocnxlMCIterNum()
     => x_iterNum / nil
```

Description

Returns the current iteration number of the main Monte Carlo simulation run. This command must be used only in a pre-run script.

For more information, see Executing Pre-run Scripts before Simulation Runs.

Arguments
None

Value Returned

`x_iterNum`	Returns the iteration number of main Monte Carlo simulation run
`nil`	Returns `nil` otherwise

Example

```
x=ocnxlMCIterNum()
when( equal(x 1) then
  initialize();
)
```

ocnxlMainSimSession

```
ocnxlMainSimSession()
     => g_session / nil
```

Description

Returns the session object for the main simulation session. This command must be used only in a pre-run script.

For more information, see Executing Pre-run Scripts before Simulation Runs.

Arguments

None

Value Returned

`g_session`	Returns the session object for the main simulation session
`nil`	Returns `nil` otherwise

Example

```
ocnxlMainSimSession()
```

ocnxlWaitUntilDone

```
ocnxlWaitUntilDone( x_runID | 'All)
    => t |nil
```

Description

This command waits for an active OCEAN XL run to complete. This command works only in XL mode. See help on `ocnSetXLMode()`.

Important

Use this function only when you are running multiple OCEAN runs in parallel, that is, when you have specified the *?waitUntilDone* argument of the ocnxlRun command to `nil`. You can enable parallel run in OCEAN XL scripts by using the ocnxlSetRunDistributeOptions function.

Arguments

x_runID	Specifies ID of the OCEAN XL run for which OCEAN needs to wait to complete before starting execution of the next command. You can specify the runId returned by the `ocnxlRun` function or the history name or the handle to the setup database for a run.
`'All`	Specify `'All` if you want to wait for all the OCEAN runs that are currently running.

Value Returned

`t`	Returns `t` if the specified runID is found.
`nil`	Returns `nil` otherwise.

Examples

Example 1

In this example, the ocnxlWaitUntilDone command waits for all OCEAN XL runs that are currently running to complete before moving to the next command in the script.

```
ocnxlLoadSetupState( "C1" 'retain ?tests t ?vars t ?parameters t ?currentMode t
    ?runOptions t ?specs t ?corners t ?extensions t
    ?modelGroups nil ?relxanalysis nil )

runid1 = ocnxlRun(?waitUntilDone nil)

ocnxlLoadSetupState( "C4" 'retain ?tests t ?vars t ?parameters t ?currentMode t
    ?runOptions t ?specs t ?corners t ?extensions t
    ?modelGroups nil ?relxanalysis nil )

runid2 = ocnxlRun(?waitUntilDone nil)

(ocnxlWaitUntilDone 'All)

ocnxlOutputSummary()
```

Example 2

In this example, the ocnxlWaitUntilDone command waits for the OCEAN XL run with runID as `runid2` to complete before moving to the next command in the script.

```
runid2 = ocnxlRun(?waitUntilDone nil)

(ocnxlWaitUntilDone runid2)

ocnxlOutputSummary()
```

Related Function

ocnxlRun, ocnxlSetRunDistributeOptions

ocnxlWriteDatasheet

```
ocnxlWriteDatasheet(
     [?name t_datasheetName ]
     [?directory t_directory ]
     [?resultsSummary g_resultsSummary]
     [?testsSummary g_testsSummary]
     [?detailedResults g_detailedResults]
     [?plots g_plots]
     [?designVarsSummary g_designVarsSummary]
     [?paramsSummary g_paramsSummary]
     [?cornersSummary g_cornersSummary]
     [?launchBrowser g_launchBrowser]
     ))
     => t |nil
```

Description

This command writes a datasheet from the latest OceanXL run. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

t_directory	Directory where the datasheet should be created. If unspecified datasheet will be created in the <lib>/<cell>/adexl/documents directory.
g_resultsSummary	Optional boolean argument that controls whether a results summary sheet will be printed or not. Results summary contains spec sheet pass/fail table. Default is t.
g_testsSummary	Optional boolean argument that controls whether a tests summary sheet will be printed or not. Tests summary contains details about the tests sweeps and corners. Default is t.
g_detailedResults	Optional boolean argument that controls whether results for all the points will be generated or not. Default is t.
g_designVarsSummary	Optional boolean argument that controls whether design variable information will be generated or not. Default is t.
g_paramsSummary	Optional boolean argument that controls whether parameters information will be generated or not. Default is t.

g_cornersSummary	Optional boolean argument that controls whether corners information will be generated or not. Default is t.
g_launchBrowser	Optional boolean argument that controls whether the generated datasheet will be displayed in a browser window. Default is t.
t_datasheetName	Optional argument that specified a title for the datasheet.
g_plots	Optional boolean argument that controls whether the generated datasheet will include all the plots. Default is t.

Value Returned

`t`	Returns `t` if the datasheet is created successfully.
`nil`	Returns `nil` otherwise.

Example

```
ocnxlWriteDatasheet(?name "My datasheet")
=> t
```

ocnxlYieldEstimationOptions

```
ocnxlYieldEstimationOptions(
     [ ?useReference t_useReference ]
     [ ?mcMethod t_mcMethod ]
     [ ?samplingMethod t_samplingMethod]
     [ ?mcNumPoints t_mcNumPoints ]
     [ ?mcNumBins t_mcNumBins]
     [ ?monteCarloSeed  t_monteCarloSeed ]
     [ ?haveYieldToStart t_yieldToStart ]
     [ ?yisToStart t_yisToStart ]
     [ ?varReductionBy t_varReductionBy ]
     [ ?iterations t_iterations ]
     [ ?designUnderTest t_designUnderTest ]
     [ ?dutInstances t_dutInstances]
     [ ?dutSummary t_dutSummary]
     [ ?ignoreFlag t_ignoreFlag ]
     [ ?yeMethod t_yeMethod ]
     [ ?yeSpecTolerance t_yeSpecTolerance ]
     [ ?yeAngleTolerance t_yeAngleTolerance ])

     => t | nil
```

Description

Lets you specify options for High Yield Estimation run mode. See help on `ocnxlRun()` for run modes. This command works only in XL mode. See help on `ocnSetXLMode()`.

Arguments

useReference	Specifies whether to use a schematic point or a reference point that you have created as a starting place for sizing. The possible values are `0` and `1`. The default value is `0`.
mcMethod	Optional argument to specify the yield estimation method to be used. The default value is `all`. The possible values are `global`, `mismatch` and `all`.
samplingMethod	Optional argument to set the default statistical sampling method for improve yield runs. The default value is `random`. Possible values are `random`, `orthogonal`, and `lhs` (Latin Hypercube).
mcNumPoints	Optional argument to set the number of Monte Carlo points you want to simulate. The default value is `200`.

mcNumBins	If the selected sampling method is `lhs`, this argument specifies the number of bins (or subdivisions) for `lhs`. Set this value when `samplingMode` is `lhs`. Default for `mcNumBins` is "". If not set, simulator uses its own default number of bins. For example, Spectre calculates the number of bins as given below: `numBins = max(t_mcNumBins, (t_mcNumPoints + t_mcStartingRunNumber -1))`
monteCarloSeed	Optional argument to specify a different seed for Monte Carlo runs. Default for `monteCarloSeed` is `12345`.
haveYieldToStart	Specifies if there is a yield in sigma value from which the high yield estimation method should be applied. The default value is `1`. Possible values `0` and `1`.
yisToStart	Specifies a yield in sigma value from which the high yield estimation method should be applied. The default value is `3.0`.
varReductionBy	Enables or disables the statistical variable reduction method. Default value `auto`, enables this method. Possible values `auto` and `disabled`.
iterations	Optional argument to specify the number of sizing/Monte Carlo iterations run for each specification. The default value is `10`.

dutSummary	Optional argument to specify a list of design under test (DUT) instances for improve yield runs. In this list, you can specify the instances and devices to which mismatch variations must be applied. The format to specify the list is as given below:
	`<testname%instances%Libname/Cellname/ Viewname%Master#testname%instances%modelname%Subcircuit#testname%instances%Schematic%Schematic>`
	where two DUT instances in the list are separated by a `#(hash).`
	`For example: “opamp090:full_diff_opamp_AC:2:1%/ I21%acOpenDiff%Subcircuit#opamp090:full_diff_opamp_AC:2:1%/I0/I1%opamp090/ampn/ schematic%Master#opamp090:full_diff_opamp_AC:2:1%/ I0/M5A, /I0/M3A%Schematic%Schematic”`
	Default for `dutSummary` is "".

ignoreFlag	Optional argument to specify if the user wants to apply mismatch variations to instances specified with `dutSummary`. Default for `ignoreFlag` is `0`. Set it to `1` if you do not want to apply mismatch variations to instances.
yeMethod	Specifies the estimation method to be used. This argument takes only one value `Worst Case Distance`.

yeSpecTolerance	Specifies the specification tolerance value. This argument is used to modify the convergence criteria to be used for WCD calculation. Default value: 0.02 Valid value range: `0.01` to `0.1` **Note:** The WCD algorithm converges only when the following two conditions are met: ■ Specification value error ratio is less than the value specified by *yeSpecTolerance*, where the specification value error ratio is calculated as: `spec_value_error_ratio = abs(spec_value - spec_target) / abs(nominal_spec_value - spec_target)` ■ Angle between the WCD point vector and gradient vector is less than the value specified by *yeAngleTolerance*.
yeAngleTolerance	Specifies the angle tolerance value between WCD point vector and gradient vector. Default value: `8.0` Valid value range: `1.0` to `15.0` For more details, refer to the note given for *yeSpecTolerance*.

Value Returned

`t`	Returns `t` if the options are specified.
`nil`	Returns `nil` otherwise

Example

```
ocnxlYieldEstimationOptions( ?useReference "0" ?mcMethod "all" ?samplingMode
"random" ?mcNumPoints "300" ?mcNumBins "" ?monteCarloSeed "" ?haveYieldToStart "1"
?yisToStart "3.0" ?varReductionBy "auto" ?iterations "20" ?yeMethod "Worst Case
Distance" )
```

ocnxlSetRelxAnalysisEnabled

```
ocnxlSetRelxAnalysisEnabled(
     g_enable)
     => t/nil
```

Description

Enables or disables reliability analysis for the setup based on the input argument

Arguments

`g_enable`	Enables reliability analysis if the value specified is `t`. Disables reliability analysis if the value specified is `nil`.

Value Returned

`t`	Returns `t` if the analysis is successfully enabled
`nil`	Returns `nil` otherwise

Example

```
ocnxlSetRelxAnalysisEnabled(t)
```

ocnxlAddRelxSetup

```
ocnxlAddRelxSetup(
     t_relxSetupName
     t_freshTest
     t_stressTest
     t_agedTest
     @key stressVarList
     agedVarList)
     =>t/nil
```

Description

Adds a new reliability analysis setup with the specified fresh, stress, and aged tests and any variables for which the values need to be overridden.

Arguments

`t_relxSetupName`	Specifies a unique name for the new reliability analysis setup
`t_freshTest`	Specifies the name of the test you want to use for running fresh simulation
`t_stressTest`	Specifies the name of the test you want to use for running stress simulation
`t_agedTest`	Specifies the name of the test you want to use for running aging simulation
`l_stressVarList`	(Optional) Provides a list of variables for which you want to modify the values. This list specifies values only for a stress simulation.
`l_agedVarList`	(Optional) Provides the list of variables for which you want to modify the values. This list specifies values only for an aging simulation.

Value Returned

`t`	Returns `t` if the analysis is successfully added

`nil`	Returns `nil` otherwise

Example

```
ocnxlAddRelxSetup("my_relx" "fresh_test" "stress_test" "aged_test"
?stressVarList '(("CAP" "100f") ("RES" "10K")))
```

ocnxlDisableRelxSetup

```
ocnxlDisableRelxSetup(
     t_relxSetupName)
     =>t/nil
```

Description

Disables the specified reliability analysis setup.

Arguments

`t_relxSetupName`	Specifies the name of the reliability analysis setup that is to be disabled

Value Returned

`t`	Returns `t` if the analysis is successfully disabled
`nil`	Returns `nil` otherwise

Example

```
ocnxlDisableRelxSetup("my_relx")
```

16

OCEAN 4.4.6 Issues

For the 4.4.6 release of OCEAN, there are some restrictions and requirements.

The netlist file that you specify for the Spectre® circuit simulator interface with the design command must be `netlist`. The full path can be specified. For example, `/usr/netlist` is acceptable. The `netlistHeader` and `netlistFooter` files are searched in the same directory where the netlist is located. Cadence recommends that you use the netlist generated from the Virtuoso® Analog Design Environment. Netlists from other sources can also be used, as long as they contain only connectivity. You might be required to make slight modifications.

- Cadence recommends full paths for the Spectre simulator model files, definition files, and stimulus files.
- The Cadence SPICE circuit simulator is still used to parse netlists for socket interfaces (spectreS and cdsSpice, for example). Therefore, the netlist that you specify with the design command must be in Cadence SPICE syntax. Cadence recommends that you use the raw netlist generated from the Virtuoso® Analog Design Environment. Netlists from other sources can also be used, as long as they can pass through Cadence SPICE. You might be required to make slight modifications.
- Any presimulation commands that you specify are appended to the final netlist (as is currently the case in the design environment). Therefore, if you have control cards already in your netlist, and specify simulation setup commands, you might duplicate control cards, which causes a warning or an error from the simulator. You might want to remove control cards from your netlist file to avoid the warnings.
- Models, include files, stimulus files, and PWLF files must be found according to the path specified with the path command.

Mixed-Signal in OCEAN 4.4.6

All of the analog OCEAN features are available in mixed-signal. This means you can set up analyses, change options, change the path, and so forth.

There are limitations in the area of mixed-signal simulation.

- If mixed-signal simulation is run using a standalone OCEAN tool, then the complete netlist must be created before running the simulation. The netlist can be created using Affirma Analog Design Environment or by specifying the design as lib-cell-view using the ocean command `design` in CIW of the workbench followed by the OCEAN commands `createNetlist` and `run`.

  For example:

  ```
  design("mylib" "ampTest" "schematic")

  ; design using lib-cell-view can only be specified in CIW of
  workbench

  createNetlist()

  run()
  ```

- If mixed-signal simulation is run using OCEAN commands in the CIW of the workbench, then the design should be specified as lib-cell-view.

  Otherwise, if the design is specified as the path to the netlist, for example as `design( "./simulation/ampTest/specter/netlist"`, then the complete netlist should be created before running the simulation using the procedure specified above.

In the 4.4.6 release, there are no commands that operate on Verilog-XL final netlists. If you need to change anything in the final netlist, you must make those changes by hand.

However, you can change any of the command line arguments that are sent to the Verilog-XL simulator. This means you can change any of the digital options or any of the mixed-signal options. To see what these options are, choose *Simulation – Options – Digital* in the Virtuoso® Analog Design Environment window.

For example, you can change acceleration, keep nodes, and library files.

Index

Symbols

Numerics

A

B

C

D

E

F

M

N

O